```
#|
 Copyright (C) 1994 by Natarajan Shankar.  All Rights Reserved.

 This script is hereby placed in the public domain, and therefore unlimited
 editing and redistribution is permitted.

 NO WARRANTY

 Natarajan Shankar PROVIDES ABSOLUTELY NO WARRANTY.  THE EVENT SCRIPT IS
 PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED,
 INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND
 FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND
 PERFORMANCE OF THE SCRIPT IS WITH YOU.  SHOULD THE SCRIPT PROVE DEFECTIVE, YOU
 ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

 IN NO EVENT WILL Natarajan Shankar BE LIABLE TO YOU FOR ANY DAMAGES, ANY LOST
 PROFITS, LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
 ARISING OUT OF THE USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT
 LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED
 BY THIRD PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF SUCH
 DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

|#

;   The Mechanical Proof of Goedel's Incompleteness Theorem
;                             by
;                         N. Shankar




;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;  I:  Defining the Proof-checker for Z2         ;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;BOOT-STRAP loads basic library of axioms about lists, litatoms, and numbers.
;;;;;;;
```

EVENT: Start with the initial **thm** theory.

EVENT: For efficiency, compile those definitions not yet compiled.

```
;;;;;;;Turn off the use of the compiled lisp definitions for the non-BOOT-STRAP
```

`;;;;;;;functions.`

EVENT: Enable defined functions; name this event 'turn-off-lisp-defns'.

`;;;;;;;Recognizers for function, variable, and predicate symbols.`

DEFINITION: function $(fn) = (\text{car}(fn) = 2)$

DEFINITION: variable $(x) = (x \in \mathbf{N})$

DEFINITION: predicate $(p) = (\text{car}(p) = 3)$

`;;;;;;;Degree (arity) and index (subscript) of a function/predicate symbol.`

DEFINITION: degree $(fn) = \text{fix}(\text{cddr}(fn))$

DEFINITION: index $(fn) = \text{fix}(\text{cadr}(fn))$

`;;;;;;;(V X) generates the Xth variable`

DEFINITION: $\text{v}(x) = \text{fix}(x)$

THEOREM: numberp-fix
$\text{fix}(x) \in \mathbf{N}$

THEOREM: variable-v
$\text{variable}(\text{v}(x))$

`;;;;;;;FN generates function symbol with index X and arity Y.`

DEFINITION: $\text{fn}(x, y) = \text{cons}(2, \text{cons}(\text{fix}(x), \text{fix}(y)))$

`;;;;;;;P generates the predicate symbol with index X and arity Y.`

DEFINITION: $\text{p}(x, y) = \text{cons}(3, \text{cons}(\text{fix}(x), \text{fix}(y)))$

THEOREM: function-fn
$\text{function}(\text{fn}(x, y))$

THEOREM: predicate-p
$\text{predicate}(\text{p}(x, y))$

`;;;;;;;F-NOT is the negation shell.`

EVENT: Add the shell *f-not*, with recognizer function symbol *f-notp* and 1 accessor: *arg*, with type restriction (none-of) and default value zero.

```
;;;;;;;F-OR is the disjunction shell.
```

EVENT: Add the shell *f-or*, with recognizer function symbol *f-orp* and 2 accessors: *arg1*, with type restriction (none-of) and default value zero; *arg2*, with type restriction (none-of) and default value zero.

```
;;;;;;;FORSOME is the existential quantification shell.
```

EVENT: Add the shell *forsome*, with recognizer function symbol *forsomep* and 2 accessors: *bind*, with type restriction (one-of numberp) and default value zero; *body*, with type restriction (none-of) and default value zero.

```
;;;;;;;EQL is the equality predicate for Z2 expressions.  Strict equality EQUAL cannot
;;;;;;;be used since expressions will be Goedel-encoded, and it is possible
;;;;;;;two different expressions to have the same Goedel-encoding.  EQL essentially
;;;;;;;checks that its two arguments have the same encoding.
```

DEFINITION:
fix1 (x)
$=$ **if** $x \in \mathbf{N}$ **then** x
  else nil endif

THEOREM: bind-count
forsomep $(x) \rightarrow (\text{count}(\text{bind}(x)) < \text{count}(x))$

DEFINITION:
eql (x, y)
$=$ **if** listp (x)
  then if listp (y) **then** eql (car (x), car (y)) $\wedge$ eql (cdr (x), cdr (y))
    else f endif
  elseif listp (y) **then f**
  elseif f-notp (x)
  then if f-notp (y) **then** eql (arg (x), arg (y))
    else f endif
  elseif f-notp (y) **then f**
  elseif f-orp (x)
  then if f-orp (y) **then** eql (arg1 (x), arg1 (y)) $\wedge$ eql (arg2 (x), arg2 (y))
    else f endif
  elseif f-orp (y) **then f**
  elseif forsomep (x)
  then if forsomep (y)
    then (bind (x) = bind (y)) $\wedge$ eql (body (x), body (y))
    else f endif
  elseif forsomep (y) **then f**
  else fix1 (x) = fix1 (y) **endif**

THEOREM: eql-comm
$\text{eql}(x,\ y) = \text{eql}(y,\ x)$

THEOREM: equal-eql
$(x = y) \rightarrow \text{eql}(x,\ y)$

THEOREM: numberp-eql
$((x \in \mathbf{N}) \vee (y \in \mathbf{N})) \rightarrow (\text{eql}(x,\ y) = (x = y))$

THEOREM: listp-eql
$\text{eql}(\text{cons}(x,\ y),\ \text{cons}(u,\ v)) = (\text{eql}(x,\ u) \wedge \text{eql}(y,\ v))$

THEOREM: f-notp-eql
$\text{eql}(\text{f-not}(x),\ \text{f-not}(y)) = \text{eql}(x,\ y)$

THEOREM: f-orp-eql
$\text{eql}(\text{f-or}(x,\ y),\ \text{f-or}(u,\ v)) = (\text{eql}(x,\ u) \wedge \text{eql}(y,\ v))$

THEOREM: forsomep-eql
$\text{eql}(\text{forsome}(x,\ y),\ \text{forsome}(u,\ v)) = ((\text{fix}(x) = \text{fix}(u)) \wedge \text{eql}(y,\ v))$

```
;;;;;;;MEMB checks if X is a member of Y, using EQL.
```

DEFINITION:
memb $(x,\ y)$
= **if** listp (y)
 then if eql $(x,$ car $(y))$ **then t**
  else memb $(x,$ cdr $(y))$ **endif**
 else f endif

THEOREM: degree-fn
$\text{degree}(\text{fn}(x,\ y)) = \text{fix}(y)$

THEOREM: degree-p
$\text{degree}(\text{p}(x,\ y)) = \text{fix}(y)$

THEOREM: index-fn
$\text{index}(\text{fn}(x,\ y)) = \text{fix}(x)$

THEOREM: index-p
$\text{index}(\text{p}(x,\ y)) = \text{fix}(x)$

THEOREM: not-function
$((x \simeq \mathbf{nil}) \vee \text{predicate}(x)) \rightarrow (\neg\ \text{function}(x))$

THEOREM: not-predicate
$((x \simeq \mathbf{nil}) \vee \text{function}(x)) \rightarrow (\neg\ \text{predicate}(x))$

EVENT: Enable index; name this event 'g0616'.

EVENT: Enable degree; name this event 'g0617'.

EVENT: Enable v; name this event 'g0618'.

EVENT: Enable fn; name this event 'g0619'.

EVENT: Enable p; name this event 'g0620'.

EVENT: Enable function; name this event 'g0621'.

EVENT: Enable predicate; name this event 'g0622'.

```
;;;;;;;X is a function or a predicate.
```

DEFINITION: func-pred (x) = (function (x) $\vee$ predicate (x))

```
;;;;;;;APPEND concatenates two lists.
```

DEFINITION:
append (x, y)
= **if** listp (x) **then** cons (car (x), append (cdr (x), y))
else y **endif**

```
;;;;;;;DEL deletes all occurrences of X from list Y, using EQL.
```

DEFINITION:
del (x, y)
= **if** listp (y)
then if eql $(x$, car $(y))$ **then** del $(x$, cdr $(y))$
else cons (car (y), del $(x$, cdr $(y)))$ **endif**
else y **endif**

THEOREM: not-memb-del
$\neg$ memb $(x$, del $(x, y))$

```
;;;;;;;COLLECT-FREE collects a list of all the free variables in the
;;;;;;;expression/list of expressions EXP, depending on (ZEROP FLG).
```

DEFINITION:
collect-free (exp, flg)

$=$ **if** *flg* $\simeq$ `0`
then if variable (*exp*) **then** cons (*exp*, **nil**)
elseif f-notp (*exp*) **then** collect-free (arg (*exp*), `0`)
elseif f-orp (*exp*)
then append (collect-free (arg1 (*exp*), `0`), collect-free (arg2 (*exp*), `0`))
elseif forsomep (*exp*)
then del (bind (*exp*), collect-free (body (*exp*), `0`))
elseif listp (*exp*) **then** collect-free (cdr (*exp*), `1`)
else nil endif
elseif listp (*exp*)
then append (collect-free (car (*exp*), `0`), collect-free (cdr (*exp*), `1`))
else nil endif

DEFINITION: sentence (*exp*) $=$ (collect-free (*exp*, `0`) $=$ **nil**)

```
;;;;;;;COVERING collects a list of all bound variables that govern free occurrences
;;;;;;;of VAR in EXP.
```

DEFINITION:
covering (*exp*, *var*, *flg*)
$=$ **if** *flg* $\simeq$ `0`
then if f-notp (*exp*) **then** covering (arg (*exp*), *var*, `0`)
elseif f-orp (*exp*)
then append (covering (arg1 (*exp*), *var*, `0`), covering (arg2 (*exp*), *var*, `0`))
elseif forsomep (*exp*)
then if memb (*var*, collect-free (*exp*, `0`))
then cons (bind (*exp*), covering (body (*exp*), *var*, `0`))
else nil endif
elseif listp (*exp*) **then** covering (cdr (*exp*), *var*, `1`)
else nil endif
elseif listp (*exp*)
then append (covering (car (*exp*), *var*, `0`), covering (cdr (*exp*), *var*, `1`))
else nil endif

```
;;;;;;;X and Y have no common elements.
```

DEFINITION:
nil-intersect (x, y)
$=$ **if** listp (x) **then** ($\neg$ memb (car (x), y)) $\wedge$ nil-intersect (cdr (x), y)
else t endif

```
;;;;;;;TERM is FREE-FOR VAR in EXP, if no free occurrence of VAR in EXP is governed
;;;;;;;by a variable which occurs free in TERM.
```

DEFINITION:
free-for (*exp*, *var*, *term*, *flg*)
$=$ nil-intersect (covering (*exp*, *var*, *flg*), collect-free (*term*, `0`))

```
;;;;;;;(F-EQUAL X Y) is the Z2 expression for (x=y).
```

DEFINITION: f-equal $(x, y) = \text{list}(\text{p}(\mathtt{0}, \mathtt{2}), x, y)$

THEOREM: or-memb-append
$\text{memb}(x, \text{append}(y, z)) = (\text{memb}(x, y) \vee \text{memb}(x, z))$

THEOREM: memb-del
$(\neg\ \text{eql}(x, z)) \rightarrow (\text{memb}(x, \text{del}(z, y)) = \text{memb}(x, y))$

```
;;;;;;;VAR-LIST checks that LIST is a list of N variables.
```

DEFINITION:
var-list$(list, n)$
$=$ **if** $(n \simeq \mathtt{0}) \vee (list \simeq \textbf{nil})$ **then** $(n \simeq \mathtt{0}) \wedge (list \simeq \textbf{nil})$
else variable$(\text{car}(list)) \wedge$ var-list$(\text{cdr}(list), n - 1)$ **endif**

```
;;;;;;;SYMB checks that X is either the equality symbol, or appears in list SYMBOLS.
```

DEFINITION:
$\text{symb}(x, symbols) = ((x = \text{p}(\mathtt{0}, \mathtt{2})) \vee \text{memb}(x, symbols))$

```
;;;;;;;TERMP checks that EXP is either a variable or an n-ary function symbol (in SYMBOLS)
;;;;;;;applied to n smaller terms.
```

DEFINITION:
termp$(exp, flg, symbols)$
$=$ **if** $flg \simeq \mathtt{0}$
then if listp(exp)
then function$(\text{car}(exp))$
$\wedge$ symb$(\text{car}(exp), symbols)$
$\wedge$ termp$(\text{cdr}(exp), \mathtt{1}, symbols)$
$\wedge$ $(\text{length}(\text{cdr}(exp)) = \text{degree}(\text{car}(exp)))$
else variable(exp) **endif**
elseif $exp \simeq \textbf{nil}$ **then t**
else termp$(\text{car}(exp), \mathtt{0}, symbols) \wedge$ termp$(\text{cdr}(exp), \mathtt{1}, symbols)$ **endif**

```
;;;;;;;ATOMP checks that EXP is an n-ary predicate symbol applied to n terms.
```

DEFINITION:
atomp$(exp, symbols)$
$=$ (predicate$(\text{car}(exp))$
$\wedge$ symb$(\text{car}(exp), symbols)$
$\wedge$ $(\text{length}(\text{cdr}(exp)) = \text{degree}(\text{car}(exp)))$
$\wedge$ termp$(\text{cdr}(exp), \mathtt{1}, symbols)$)

THEOREM: atomp-reduc
atomp (cons (x, y), *symbols*)
= (predicate (x)
∧ symb (x, *symbols*)
∧ (length (y) = degree (x))
∧ termp (y, 1, *symbols*))

THEOREM: atomp-f-equal
atomp (f-equal (x, y), *symbols*)
= (termp (x, 0, *symbols*) ∧ termp (y, 0, *symbols*))

```
;;;;;;;SUBST substitutes TERM for all free occurrences of VAR in EXP.
```

DEFINITION:
subst (*exp*, *var*, *term*, *flg*)
= **if** *flg* ≃ 0
then if variable (*exp*)
then if *exp* = *var* **then** *term*
else *exp* **endif**
elseif f-notp (*exp*) **then** f-not (subst (arg (*exp*), *var*, *term*, 0))
elseif f-orp (*exp*)
then f-or (subst (arg1 (*exp*), *var*, *term*, 0),
subst (arg2 (*exp*), *var*, *term*, 0))
elseif forsomep (*exp*)
then if eql (bind (*exp*), *var*) **then** *exp*
else forsome (bind (*exp*), subst (body (*exp*), *var*, *term*, 0)) **endif**
elseif listp (*exp*)
then cons (car (*exp*), subst (cdr (*exp*), *var*, *term*, 1))
else *exp* **endif**
elseif listp (*exp*)
then cons (subst (car (*exp*), *var*, *term*, 0), subst (cdr (*exp*), *var*, *term*, 1))
else *exp* **endif**

```
;;;;;;;SUB checks if every element of X occurs in Y, using EQL.
```

DEFINITION:
sub (x, y)
= **if** listp (x) **then** memb (car (x), y) ∧ sub (cdr (x), y)
else t endif

```
;;;;;;;SUBSET checks if every element of X occurs in Y, using EQUAL.
```

DEFINITION:
subset (x, y)
= **if** listp (x) **then** (car (x) ∈ y) ∧ subset (cdr (x), y)
else t endif

THEOREM: sub-subset
subset $(x,\ y) \rightarrow$ sub $(x,\ y)$

THEOREM: subst-f-equal
subst (f-equal $(x,\ y)$, *var*, *term*, 0)
$=$ f-equal (subst $(x$, *var*, *term*, 0), subst $(y$, *var*, *term*, 0))

EVENT: Enable f-equal; name this event 'g0623'.

EVENT: Enable atomp; name this event 'g0253'.

```
;;;;;;;FORMULA checks if EXP is a well-formed formula.
```

DEFINITION:
formula (*exp*, *symbols*)
$=$ **if** f-notp (*exp*) **then** formula (arg (*exp*), *symbols*)
elseif f-orp (*exp*)
then formula (arg1 (*exp*), *symbols*) $\wedge$ formula (arg2 (*exp*), *symbols*)
elseif forsomep (*exp*) **then** formula (body (*exp*), *symbols*)
else atomp (*exp*, *symbols*) **endif**

THEOREM: formula-f-equal
formula (f-equal $(x,\ y)$, *symbols*)
$=$ (termp $(x$, 0, *symbols*) $\wedge$ termp $(y$, 0, *symbols*))

```
;;;;;;;(F-AND X Y) denotes (X ^ Y).
```

DEFINITION: f-and $(x,\ y) =$ f-not (f-or (f-not (x), f-not (y)))

```
;;;;;;;(F-IMPLIES X Y) denotes (X => Y).
```

DEFINITION: f-implies $(x,\ y) =$ f-or (f-not $(x),\ y)$

```
;;;;;;;(FORALL X EXP) denotes (Ax exp).
```

DEFINITION: $\forall$ *var* *exp* $=$ f-not (forsome (*var*, f-not (*exp*)))

```
;;;;;;;PAIREQUALS returns a formula of the form
;;;;;;;((x1=y1) -> (...-> ((xn=yn) -> EXP))).
```

DEFINITION:
pairequals (*vars1*, *vars2*, *exp*)
$=$ **if** listp (*vars1*) $\wedge$ listp (*vars2*)
then f-implies (f-equal (car (*vars1*), car (*vars2*)),
pairequals (cdr (*vars1*), cdr (*vars2*), *exp*))
else *exp* **endif**

Definition: second (x) = cadr (x)

Definition: third (x) = caddr (x)

Definition: fourth (x) = cadddr (x)

```
;;;;;;;FOL-AXIOM-PROOF checks if CONC is a logical axiom (in a theory with
;;;;;;;SYMBOLS), using hints from HINTS.  The axioms are: [1] propositional
;;;;;;;axioms of the form, "((not A) or A)";  [2] substitution axioms of
;;;;;;;the form, "[t/x]A -> (forsome x A)";  [3] Identity axioms of the form,
;;;;;;;"(x=x)";  [4] Equality axioms for functions, of the form,
;;;;;;;"((x1=y1) -> (...-> ((xn=yn) -> ((fx1...xn)=(fy1...yn)))))";
;;;;;;;[5] Equality axioms for predicates, of the form,
;;;;;;;"((x1=y1) -> (...-> ((xn=yn) -> ((px1...xn) -> (py1...yn)))))".
```

Definition:
fol-axiom-proof(*conc*, *hints*, *symbols*)
= **if** car (*hints*) = 1
then formula (second (*hints*), *symbols*)
∧ eql (*conc*, f-or (f-not (second (*hints*)), second (*hints*)))
elseif car (*hints*) = 2
then formula (second (*hints*), *symbols*)
∧ variable (third (*hints*))
∧ termp (fourth (*hints*), 0, *symbols*)
∧ free-for (second (*hints*), third (*hints*), fourth (*hints*), 0)
∧ eql (*conc*,
f-implies (subst (second (*hints*),
third (*hints*),
fourth (*hints*),
0),
forsome (third (*hints*), second (*hints*))))
elseif car (*hints*) = 3
then termp (second (*hints*), 0, *symbols*)
∧ eql (*conc*, f-equal (second (*hints*), second (*hints*)))
elseif car (*hints*) = 4
then termp (third (*hints*), 1, *symbols*)
∧ termp (fourth (*hints*), 1, *symbols*)
∧ function (second (*hints*))
∧ (length (third (*hints*)) = degree (second (*hints*)))
∧ (length (fourth (*hints*)) = degree (second (*hints*)))
∧ eql (*conc*,
pairequals (third (*hints*),
fourth (*hints*),
f-equal (cons (second (*hints*), third (*hints*)),
cons (second (*hints*), fourth (*hints*)))))

elseif car (*hints*) = `5`
then termp (third (*hints*), `1`, *symbols*)
∧ termp (fourth (*hints*), `1`, *symbols*)
∧ predicate (second (*hints*))
∧ (length (third (*hints*)) = degree (second (*hints*)))
∧ (length (fourth (*hints*)) = degree (second (*hints*)))
∧ eql (*conc*,
pairequals (third (*hints*),
fourth (*hints*),
f-implies (cons (second (*hints*), third (*hints*)),
cons (second (*hints*), fourth (*hints*)))))
else f endif

```
;;;;;;;FOL-RULES checks if CONC follows from the SUB-GOALS by an application
;;;;;;;of one of the logical rules of inference, using hints from HINTS.
;;;;;;;The rules of inference are:  [1] Expansion: derive "(A or B)" from "B";
;;;;;;;[2] Contraction: derive "A" from "(A or A)";
;;;;;;;[3] Associativity: derive "((A or B) or C)" from "(A or (B or C))";
;;;;;;;[4] Cut: derive "(B or C)" from "(A or B)" and "((not B) or C)";
;;;;;;;[5] E-introduction: derive "((forsome x A) -> B)" from "(A -> B)",
;;;;;;;provided "x" does not occur free in "B".
```

DEFINITION:
fol-rules (*conc*, *hints*, *sub-goals*, *symbols*)
= **if** car (*hints*) = `1`
then formula (second (*hints*), *symbols*)
∧ formula (car (*sub-goals*), *symbols*)
∧ eql (*conc*, f-or (second (*hints*), car (*sub-goals*)))
elseif car (*hints*) = `2`
then formula (second (*hints*), *symbols*)
∧ eql (*sub-goals*, list (f-or (second (*hints*), second (*hints*))))
∧ eql (*conc*, second (*hints*))
elseif car (*hints*) = `3`
then formula (second (*hints*), *symbols*)
∧ formula (third (*hints*), *symbols*)
∧ formula (fourth (*hints*), *symbols*)
∧ eql (*sub-goals*,
list (f-or (second (*hints*),
f-or (third (*hints*), fourth (*hints*)))))
∧ eql (*conc*,
f-or (f-or (second (*hints*), third (*hints*)), fourth (*hints*)))
elseif car (*hints*) = `4`
then formula (second (*hints*), *symbols*)
∧ formula (third (*hints*), *symbols*)

$\wedge$ formula (fourth (*hints*), *symbols*)
$\wedge$ eql (*sub-goals*,
  list (f-or (second (*hints*), third (*hints*)),
    f-or (f-not (second (*hints*)), fourth (*hints*))))
$\wedge$ eql (*conc*, f-or (third (*hints*), fourth (*hints*)))
elseif car (*hints*) = 5
then variable (second (*hints*))
$\wedge$ formula (third (*hints*), *symbols*)
$\wedge$ formula (fourth (*hints*), *symbols*)
$\wedge$ ($\neg$ memb (second (*hints*), collect-free (fourth (*hints*), 0)))
$\wedge$ eql (*sub-goals*, list (f-implies (third (*hints*), fourth (*hints*))))
$\wedge$ eql (*conc*,
  f-implies (forsome (second (*hints*), third (*hints*)),
    fourth (*hints*)))
else f endif

```
;;;;;;;Some notation for: [1] Set-theoretic membership: "(x in y)";
;;;;;;;[2] Null Set: "(phi)";  [3] Pairing: "{x, y}";
;;;;;;;[4] Union: "(x U y)";  [5] Integerhood:  "(I x)" defined as
;;;;;;;"x is transitive and linearly ordered";
;;;;;;;[6] Successor: "(S x)" defined as "(x U {x})", where "{x}" is "{x, x}";
;;;;;;;[7] If-and-only-if: "(A <-> B)".
```

DEFINITION: isin (x, y) = list (p (1, 2), x, y)

DEFINITION: PHI = list (fn (0, 0))

DEFINITION: z-pair (x, y) = list (fn (1, 2), x, y)

DEFINITION: z-union (x, y) = list (fn (2, 2), x, y)

DEFINITION: z-int (x) = list (p (2, 1), x)

DEFINITION: z-succ (x) = list (fn (3, 1), x)

DEFINITION: f-iff (x, y) = f-and (f-implies (x, y), f-implies (y, x))

```
;;;;;;;Z2-AXIOMS checks if CONC is a set-theoretic axiom from Z2.
;;;;;;;These axioms are:
;;;;;;;[1] Extensionality: "(x=y) <-> (forall z ((z in x) <-> (z in y)))".
;;;;;;;[2] Null set: "(not (x in (phi)))".
;;;;;;;[3] Pairing: "(z in {x, y}) <-> ((z=x) v (z=y))".
;;;;;;;[4] Union: "(z in (x U y)) <-> ((z in x) v (z in y))".
;;;;;;;[5] Induction:  "A[(phi)] ^ (forall x (((I x) ^ A[x]) -> A[(S x)])
;;;;;;;                     -> (forall x ((I x) -> A[x]))"
```

DEFINITION:

```
z2-axioms (conc, hints, symbols)
=   if car (hints) = 1
    then variable (fourth (hints))
         ∧  termp (second (hints), 0, symbols)
         ∧  termp (third (hints), 0, symbols)
         ∧  (¬ memb (fourth (hints), collect-free (second (hints), 0)))
         ∧  (¬ memb (fourth (hints), collect-free (third (hints), 0)))
         ∧  eql (conc,
                 f-iff (f-equal (second (hints), third (hints)),
                        ∀ fourth, hints f-iff (isin (fourth (hints),
                                                     second (hints)),
                                               isin (fourth (hints),
                                                     third (hints)))))
    elseif car (hints) = 2
    then termp (second (hints), 0, symbols)
         ∧  eql (conc, f-not (isin (second (hints), PHI)))
    elseif car (hints) = 3
    then termp (list (second (hints), third (hints), fourth (hints)), 1, symbols)
         ∧  eql (conc,
                 f-iff (isin (fourth (hints),
                              z-pair (second (hints), third (hints))),
                        f-or (f-equal (fourth (hints), second (hints)),
                              f-equal (fourth (hints), third (hints)))))
    elseif car (hints) = 4
    then termp (list (second (hints), third (hints), fourth (hints)), 1, symbols)
         ∧  eql (conc,
                 f-iff (isin (fourth (hints),
                              z-union (second (hints), third (hints))),
                        f-or (isin (fourth (hints), second (hints)),
                              isin (fourth (hints), third (hints)))))
    elseif car (hints) = 5
    then variable (third (hints))
         ∧  formula (second (hints), symbols)
         ∧  eql (conc,
                 f-implies (f-and (subst (second (hints),
                                          third (hints),
                                          PHI,
                                          0),
                                   ∀ third, hints f-implies (f-and (z-int (third (hints)),
                                                                    second (hints)),
                                                             subst (second (hints),
                                                                    third (hints),
                                                                    z-succ (third (hints)),
```

$0))),$

$\forall$ *third*, *hints* f-implies (z-int (third (*hints*)),
second (*hints*))))

else f endif

```
;;;;;;;X is a list with no duplicates.
```

DEFINITION:
set (x)
= **if** listp (x) **then** ($\neg$ memb (car (x), cdr (x))) $\wedge$ set (cdr (x))
else t endif

```
;;;;;;;Returns the element paired to X in the alist Y.
```

DEFINITION:
assoc (x, y)
= **if** listp (y)
then if eql (caar (y), x) **then** cdar (y)
else assoc (x, cdr (y)) **endif**
else 0 **endif**

```
;;;;;;;Returns the list of symbols following the last occurrence of SYM in SYMBOLS.
```

DEFINITION:
rest-of (*sym*, *symbols*)
= **if** listp (*symbols*)
then if eql (car (*symbols*), *sym*) $\wedge$ ($\neg$ memb (*sym*, cdr (*symbols*)))
then cdr (*symbols*)
else rest-of (*sym*, cdr (*symbols*)) **endif**
else nil endif

```
;;;;;;;SYMB-DEFN-PROOF checks if CONC is admissible as the definition of a new
;;;;;;;function or predicate symbol.  The definition of a predicate symbol
;;;;;;;must have the form, "(px1...xn) <-> A", where the "xi" are distinct
;;;;;;;variables, and "A" is a formula w.r.t. the symbols that follow "p" in SYMBOLS
;;;;;;;which contains no free variables other than the "xi".
;;;;;;;The definition of a function symbol must have the form, "[(fx1...xn)/x]A",
;;;;;;;where the "x" and the "xi" are distinct, "A" is a formula w.r.t. the
;;;;;;;symbols that follow "f" in SYMBOLS, and "(forsome-unique x A)" is
;;;;;;;provable in a theory which only contains the symbols that follow "f" in SYMBOLS.
```

DEFINITION:
symb-defn-proof (*conc*, *hints*, *sub-goals*, *defn*, *symbols*)
= **if** predicate (caar (*hints*))
then set (cdar (*hints*))

```
        ∧   var-list (cdar (hints), degree (caar (hints)))
        ∧   eql (conc, f-iff (car (hints), defn))
        ∧   sub (collect-free (defn, 0), cdar (hints))
        ∧   formula (defn, rest-of (caar (hints), symbols))
    elseif function (caar (hints))
    then set (cons (fourth (hints), cons (second (hints), cdar (hints))))
        ∧   var-list (cdar (hints), degree (caar (hints)))
        ∧   variable (second (hints))
        ∧   variable (fourth (hints))
        ∧   formula (third (hints), rest-of (caar (hints), symbols))
        ∧   sub (collect-free (third (hints), 0),
                     cons (second (hints), cdar (hints)))
        ∧   free-for (third (hints), second (hints), car (hints), 0)
        ∧   free-for (third (hints), second (hints), fourth (hints), 0)
        ∧   eql (conc, subst (third (hints), second (hints), car (hints), 0))
        ∧   eql (conc, defn)
        ∧   eql (sub-goals,
                 list (forsome (second (hints), third (hints)),
                       f-implies (f-and (third (hints),
                                         subst (third (hints),
                                                second (hints),
                                                fourth (hints),
                                                0)),
                                  f-equal (second (hints), fourth (hints)))))
    else f endif
```

```
;;;;;;;A formal proof is represented as 4-tuple: <step hints conclusion sub-proofs>,
;;;;;;;where the conclusion is derived from the conclusions of the sub-proofs
;;;;;;;by means of a proof-step given by the combination of ‘step’ and ‘hints’.
;;;;;;;CONC returns the conclusion of a proof/list of proofs.
```

Definition:

```
conc (pf, flg)
=   if flg ≃ 0 then third (pf)
    elseif listp (pf) then cons (conc (car (pf), 0), conc (cdr (pf), 1))
    else nil endif
```

```
;;;;;;;PRF checks if PF is well-formed proof in a theory whose function/predicate
;;;;;;;symbols appear in SYMBOLS, where the definitions of the defined symbols
;;;;;;;appear in DEFNS, and GIVEN is a list of assumptions.  The argument GIVEN is
;;;;;;;there merely to facilitate the proof of essential incompleteness, and is never
;;;;;;;actually used to demonstrate the existence of a proof.  One can obviously
;;;;;;;introduce an inconsistency through GIVEN.
;;;;;;;PRF recurses down PF and checks if the conclusion is either [1] a logical axiom;
;;;;;;;[2] follows from the conclusion of the sub-proofs (which are themselves
```

```
;;;;;;;well-formed) by means of one of the rules of inference; [3] a set-theoretic
;;;;;;;axiom; [4] a function/predicate definition; [5] an assumption from GIVEN.
```

DEFINITION:
prf (pf, $given$, $defns$, flg, $symbols$)
$=$ **if** $flg \simeq$ `0`
then if $pf \simeq$ **nil then f**
elseif $\neg$ formula (conc (pf, `0`), $symbols$) **then f**
elseif car (pf) $=$ `1`
then fol-axiom-proof (conc (pf, `0`), second (pf), $symbols$)
elseif car (pf) $=$ `2`
then fol-rules (conc (pf, `0`), second (pf), conc (fourth (pf), `1`), $symbols$)
$\wedge$ prf (fourth (pf), $given$, $defns$, `1`, $symbols$)
elseif car (pf) $=$ `3`
then z2-axioms (conc (pf, `0`), second (pf), $symbols$)
elseif car (pf) $=$ `4`
then symb-defn-proof (conc (pf, `0`),
second (pf),
conc (fourth (pf), `1`),
assoc (caar (second (pf)), $defns$),
$symbols$)
$\wedge$ **if** function (caar (cadr (pf)))
then prf (cadddr (pf),
$given$,
$defns$,
`1`,
rest-of (caar (cadr (pf)), $symbols$))
else t endif
elseif car (pf) $=$ `5` **then** memb (conc (pf, `0`), $given$)
else f endif
elseif $pf \simeq$ **nil then t**
else prf (car (pf), $given$, $defns$, `0`, $symbols$)
$\wedge$ prf (cdr (pf), $given$, $defns$, `1`, $symbols$) **endif**

```
;;;;;;;PROVES checks that PF is a well-formed proof whose conclusion is EXP.
```

DEFINITION:
proves (pf, exp, $given$, $defns$, $symbols$)
$=$ (eql (conc (pf, `0`), exp)
$\wedge$ formula (exp, $symbols$)
$\wedge$ prf (pf, $given$, $defns$, `0`, $symbols$))

```
;;;;;;;What follows is a series of rewrite rules which convert the proof-checker above
;;;;;;;into a set of rewrite rules.  Functions like PROP-AXIOM construct axioms
;;;;;;;whose proofs are constructed by 'proof-constructors' like PROP-AXIOM-PROOF.
```

DEFINITION: prop-axiom (exp) = f-or (f-not (exp), exp)

DEFINITION:
subst-axiom (exp, var, $term$)
= f-implies (subst (exp, var, $term$, 0), forsome (var, exp))

DEFINITION: ident-axiom (var) = f-equal (var, var)

DEFINITION:
prop-axiom-proof (exp) = list (1, list (1, exp), prop-axiom (exp))

DEFINITION:
subst-axiom-proof (exp, var, $term$)
= list (1, list (2, exp, var, $term$), subst-axiom (exp, var, $term$))

DEFINITION:
ident-axiom-proof (var) = list (1, list (3, var), f-equal (var, var))

DEFINITION:
equal-axiom1 ($vars1$, $vars2$, fn)
= pairequals ($vars1$, $vars2$, f-equal (cons (fn, $vars1$), cons (fn, $vars2$)))

DEFINITION:
equal-axiom1-proof (fn, $vars1$, $vars2$)
= list (1, list (4, fn, $vars1$, $vars2$), equal-axiom1 ($vars1$, $vars2$, fn))

DEFINITION:
equal-axiom2 ($vars1$, $vars2$, pr)
= pairequals ($vars1$, $vars2$, f-implies (cons (pr, $vars1$), cons (pr, $vars2$)))

DEFINITION:
equal-axiom2-proof (pr, $vars1$, $vars2$)
= list (1, list (5, pr, $vars1$, $vars2$), equal-axiom2 ($vars1$, $vars2$, pr))

THEOREM: formula-f-not
formula (f-not (x), $symbols$) = formula (x, $symbols$)

THEOREM: formula-f-or
formula (f-or (x, y), $symbols$) = (formula (x, $symbols$) $\wedge$ formula (y, $symbols$))

THEOREM: formula-forsome
formula (forsome (x, y), $symbols$) = formula (y, $symbols$)

```
;;;;;;;The lemma below establishes that PROP-AXIOM-PROOF constructs a well-formed proof
;;;;;;;of a propositional axiom.  Once this is proved, the definition of PROP-AXIOM-PROOF
;;;;;;;is irrelevant and is disabled.
```

THEOREM: prop-axiom-proves
formula $(exp,\ symbols)$
$\rightarrow$ proves (prop-axiom-proof (exp), f-or (f-not (exp), exp), $given,\ defns,\ symbols$)

THEOREM: subst-length
$(flg \not\simeq \texttt{0}) \rightarrow$ (length (subst $(exp,\ var,\ term,\ flg)$) $=$ length (exp))

THEOREM: subst-termp
(termp $(exp,\ flg,\ symbols)$ $\wedge$ termp $(term,\ \texttt{0},\ symbols)$)
$\rightarrow$ termp (subst $(exp,\ var,\ term,\ flg)$, flg, $symbols$)

EVENT: Disable atomp; name this event 'g0606'.

THEOREM: subst-formula
(formula $(exp,\ symbols)$ $\wedge$ termp $(term,\ \texttt{0},\ symbols)$)
$\rightarrow$ formula (subst $(exp,\ var,\ term,\ \texttt{0})$, $symbols$)

EVENT: Enable atomp; name this event 'g0607'.

THEOREM: subst-axiom-proves
((lhs = subst $(exp,\ var,\ term,\ \texttt{0})$)
$\wedge$ variable (var)
$\wedge$ termp $(term,\ \texttt{0},\ symbols)$
$\wedge$ formula $(exp,\ symbols)$
$\wedge$ free-for $(exp,\ var,\ term,\ \texttt{0})$)
$\rightarrow$ proves (subst-axiom-proof $(exp,\ var,\ term)$,
f-implies (lhs, forsome $(var,\ exp)$),
$given$,
$defns$,
$symbols$)

THEOREM: formula-pairequals
(termp $(vars1,\ \texttt{1},\ symbols)$ $\wedge$ termp $(vars2,\ \texttt{1},\ symbols)$ $\wedge$ formula $(exp,\ symbols)$)
$\rightarrow$ formula (pairequals $(vars1,\ vars2,\ exp)$, $symbols$)

THEOREM: equal-axiom1-proves
(function (fn)
$\wedge$ symb $(fn,\ symbols)$
$\wedge$ (length $(vars1)$ = degree (fn))
$\wedge$ (length $(vars2)$ = degree (fn))
$\wedge$ termp $(vars1,\ \texttt{1},\ symbols)$
$\wedge$ termp $(vars2,\ \texttt{1},\ symbols)$
$\wedge$ ($concl$ = equal-axiom1 $(vars1,\ vars2,\ fn)$))
$\rightarrow$ proves (equal-axiom1-proof $(fn,\ vars1,\ vars2)$, $concl,\ given,\ defns,\ symbols$)

Theorem: equal-axiom2-proves

$$
\begin{aligned}
&(\text{predicate}\,(pr)\\
&\ \ \wedge\ \ \text{symb}\,(pr,\ symbols)\\
&\ \ \wedge\ \ (\text{length}\,(vars1) = \text{degree}\,(pr))\\
&\ \ \wedge\ \ (\text{length}\,(vars2) = \text{degree}\,(pr))\\
&\ \ \wedge\ \ \text{termp}\,(vars1,\ \mathtt{1},\ symbols)\\
&\ \ \wedge\ \ \text{termp}\,(vars2,\ \mathtt{1},\ symbols)\\
&\ \ \wedge\ \ (concl = \text{equal-axiom2}\,(vars1,\ vars2,\ pr)))\\
&\rightarrow\ \ \text{proves}\,(\text{equal-axiom2-proof}\,(pr,\ vars1,\ vars2),\ concl,\ given,\ defns,\ symbols)
\end{aligned}
$$

Theorem: ident-axiom-proves

$$
\begin{aligned}
&\text{termp}\,(var,\ \mathtt{0},\ symbols)\\
&\rightarrow\ \ \text{proves}\,(\text{ident-axiom-proof}\,(var),\ \text{f-equal}\,(var,\ var),\ given,\ defns,\ symbols)
\end{aligned}
$$

Definition: $\text{assum-proof}\,(exp) = \text{list}\,(\mathtt{5},\ \mathbf{nil},\ exp)$

Theorem: assum-proof-proves

$$
\begin{aligned}
&((exp \in given) \wedge \text{formula}\,(exp,\ symbols))\\
&\rightarrow\ \ \text{proves}\,(\text{assum-proof}\,(exp),\ exp,\ given,\ defns,\ symbols)
\end{aligned}
$$

Theorem: memb-member

$$(x \in y) \rightarrow \text{memb}\,(x,\ y)$$

Definition:

$$
\begin{aligned}
&\text{fun-defn-proof}\,(fn,\ vars,\ x,\ y,\ exp,\ pf1,\ pf2)\\
=\ \ &\text{list}\,(\mathtt{4},\\
&\qquad \text{list}\,(\text{cons}\,(fn,\ vars),\ x,\ exp,\ y),\\
&\qquad \text{subst}\,(exp,\ x,\ \text{cons}\,(fn,\ vars),\ \mathtt{0}),\\
&\qquad \text{list}\,(pf1,\ pf2))
\end{aligned}
$$

Theorem: fun-defn-proof-proves

$$
\begin{aligned}
&(\text{var-list}\,(vars,\ \text{degree}\,(fn))\\
&\ \ \wedge\ \ \text{variable}\,(x)\\
&\ \ \wedge\ \ \text{variable}\,(y)\\
&\ \ \wedge\ \ \text{set}\,(\text{cons}\,(y,\ \text{cons}\,(x,\ vars)))\\
&\ \ \wedge\ \ \text{function}\,(fn)\\
&\ \ \wedge\ \ \text{symb}\,(fn,\ symbols)\\
&\ \ \wedge\ \ \text{formula}\,(a\text{-}prime,\ symbols)\\
&\ \ \wedge\ \ \text{subset}\,(\text{collect-free}\,(exp,\ \mathtt{0}),\ \text{cons}\,(x,\ vars))\\
&\ \ \wedge\ \ \text{free-for}\,(exp,\ x,\ \text{cons}\,(fn,\ vars),\ \mathtt{0})\\
&\ \ \wedge\ \ \text{free-for}\,(exp,\ x,\ y,\ \mathtt{0})\\
&\ \ \wedge\ \ (a\text{-}prime = \text{assoc}\,(fn,\ defns))\\
&\ \ \wedge\ \ \text{proves}\,(pf1,\ \text{forsome}\,(x,\ exp),\ given,\ defns,\ \text{rest-of}\,(fn,\ symbols))\\
&\ \ \wedge\ \ \text{proves}\,(pf2,\\
&\qquad\qquad \text{f-implies}\,(\text{f-and}\,(exp,\ \text{subst}\,(exp,\ x,\ y,\ \mathtt{0})),\ \text{f-equal}\,(x,\ y)),
\end{aligned}
$$

$given$,
$defns$,
rest-of (fn, $symbols$))
$\land$ ($a\text{-}prime$ = subst (exp, x, cons (fn, $vars$), `0`)))
$\rightarrow$ proves (fun-defn-proof (fn, $vars$, x, y, exp, $pf1$, $pf2$),
$a\text{-}prime$,
$given$,
$defns$,
$symbols$)

EVENT: Enable fun-defn-proof; name this event 'g0591'.

DEFINITION:
pred-defn-proof (pr, $vars$, $defn$)
= list (`4`, list (cons (pr, $vars$)), f-iff (cons (pr, $vars$), $defn$))

THEOREM: var-list-termp
var-list ($vars$, length ($vars$)) $\rightarrow$ termp ($vars$, `1`, $symbols$)

THEOREM: var-list-length
(length ($vars$) $\neq$ fix (n)) $\rightarrow$ ($\neg$ var-list ($vars$, n))

THEOREM: memb-rest-of
($\neg$ memb (x, z)) $\rightarrow$ ($\neg$ memb (x, rest-of (y, z)))

THEOREM: termp-rest-of
($\neg$ termp (exp, flg, $symbols$)) $\rightarrow$ ($\neg$ termp (exp, flg, rest-of ($symb$, $symbols$)))

EVENT: Disable atomp; name this event 'g0594'.

THEOREM: formula-rest-of
($\neg$ formula (exp, $symbols$)) $\rightarrow$ ($\neg$ formula (exp, rest-of ($symb$, $symbols$)))

EVENT: Enable atomp; name this event 'g0595'.

THEOREM: pred-defn-proof-proves
(var-list ($vars$, degree (pr))
$\land$ set ($vars$)
$\land$ predicate (pr)
$\land$ symb (pr, $symbols$)
$\land$ ($defn$ = assoc (pr, $defns$))
$\land$ formula ($defn$, rest-of (pr, $symbols$))
$\land$ subset (collect-free ($defn$, `0`), $vars$)
$\land$ ($concl$ = f-iff (cons (pr, $vars$), $defn$)))
$\rightarrow$ proves (pred-defn-proof (pr, $vars$, $defn$), $concl$, $given$, $defns$, $symbols$)

EVENT: Enable pred-defn-proof; name this event 'g0592'.

DEFINITION:
expan-proof $(a,\ b,\ pf)$ = list (2, list (1, a), f-or ($a,\ b$), list (pf))

DEFINITION: contrac-proof $(a,\ pf)$ = list (2, list (2, a), a, list (pf))

DEFINITION:
assoc-proof $(a,\ b,\ c,\ pf)$ = list (2, list (3, $a,\ b,\ c$), f-or (f-or ($a,\ b$), c), list (pf))

DEFINITION:
cut-proof $(a,\ b,\ c,\ pf1,\ pf2)$ = list (2, list (4, $a,\ b,\ c$), f-or ($b,\ c$), list ($pf1,\ pf2$))

DEFINITION:
forsome-intro-proof ($var,\ a,\ b,\ pf$)
= list (2, list (5, $var,\ a,\ b$), f-implies (forsome ($var,\ a$), b), list (pf))

EVENT: Enable fol-axiom-proof; name this event 'g0624'.

EVENT: Enable fun-defn-proof; name this event 'g0625'.

THEOREM: expan-proof-proves
(formula (a, $symbols$) ∧ proves ($pf,\ b,\ given,\ defns,\ symbols$))
→ proves (expan-proof ($a,\ b,\ pf$), f-or ($a,\ b$), $given,\ defns,\ symbols$)

THEOREM: assoc-proof-proves
proves (pf, f-or (a, f-or ($b,\ c$)), $given,\ defns,\ symbols$)
→ proves (assoc-proof ($a,\ b,\ c,\ pf$), f-or (f-or ($a,\ b$), c), $given,\ defns,\ symbols$)

THEOREM: contrac-proof-proves
proves (pf, f-or ($a,\ a$), $given,\ defns,\ symbols$)
→ proves (contrac-proof ($a,\ pf$), $a,\ given,\ defns,\ symbols$)

THEOREM: cut-proof-proves
(proves ($pf1$, f-or ($a,\ b$), $given,\ defns,\ symbols$)
∧ proves ($pf2$, f-or (f-not (a), c), $given,\ defns,\ symbols$)
∧ formula (f-or ($b,\ c$), $symbols$))
→ proves (cut-proof ($a,\ b,\ c,\ pf1,\ pf2$), f-or ($b,\ c$), $given,\ defns,\ symbols$)

THEOREM: forsome-intro-proves
(proves (pf, f-implies ($a,\ b$), $given,\ defns,\ symbols$)
∧ (¬ memb (var, collect-free (b, 0)))
∧ variable (var)
∧ formula (f-implies (forsome ($var,\ a$), b), $symbols$))

$\rightarrow$ proves (forsome-intro-proof (var, a, b, pf),
f-implies (forsome (var, a), b),
$given$,
$defns$,
$symbols$)

EVENT: Enable prop-axiom-proof; name this event 'g2752'.

EVENT: Enable subst-axiom-proof; name this event 'g2753'.

EVENT: Enable equal-axiom1-proof; name this event 'g2754'.

EVENT: Enable equal-axiom2-proof; name this event 'g2755'.

EVENT: Enable ident-axiom-proof; name this event 'g2756'.

EVENT: Enable assum-proof; name this event 'g2757'.

EVENT: Enable fun-defn-proof; name this event 'g2758'.

EVENT: Enable expan-proof; name this event 'g2759'.

EVENT: Enable contrac-proof; name this event 'g2760'.

EVENT: Enable assoc-proof; name this event 'g2761'.

EVENT: Enable cut-proof; name this event 'g2762'.

EVENT: Enable forsome-intro-proof; name this event 'g2763'.

EVENT: Enable fol-rules; name this event 'g0626'.

THEOREM: nlistp-proves
($pf \simeq$ **nil**) $\rightarrow$ ($\neg$ proves (pf, exp, $given$, $defns$, $symbols$))

DEFINITION:
ext-axiom (x, $terma$, $termb$)
= f-iff (f-equal ($terma$, $termb$), $\forall$ x f-iff (isin (x, $terma$), isin (x, $termb$)))

Definition:
ext-axiom-proof $(x,\ terma,\ termb)$
$=$ list $(\mathtt{3},\ \text{list}\,(\mathtt{1},\ terma,\ termb,\ x),\ \text{ext-axiom}\,(x,\ terma,\ termb))$

Theorem: ext-axiom-proof-proves
$((\neg\ \text{memb}\,(x,\ \text{collect-free}\,(terma,\ \mathtt{0})))$
$\land\ (\neg\ \text{memb}\,(x,\ \text{collect-free}\,(termb,\ \mathtt{0})))$
$\land\ \text{variable}\,(x)$
$\land\ \text{memb}\,(\text{p}\,(\mathtt{1},\ \mathtt{2}),\ symbols)$
$\land\ \text{termp}\,(terma,\ \mathtt{0},\ symbols)$
$\land\ \text{termp}\,(termb,\ \mathtt{0},\ symbols))$
$\rightarrow\ \text{proves}\,(\text{ext-axiom-proof}\,(x,\ terma,\ termb),$
$\quad\quad \text{f-iff}\,(\text{f-equal}\,(terma,\ termb),$
$\quad\quad\quad \forall\ x\ \text{f-iff}\,(\text{isin}\,(x,\ terma),\ \text{isin}\,(x,\ termb))),$
$\quad\quad given,$
$\quad\quad defns,$
$\quad\quad symbols)$

Event: Enable ext-axiom-proof; name this event 'g0374'.

Definition: null-axiom $(term)$ $=$ f-not $(\text{isin}\,(term,\ \text{PHI}))$

Definition:
null-axiom-proof $(term)$ $=$ list $(\mathtt{3},\ \text{list}\,(\mathtt{2},\ term),\ \text{null-axiom}\,(term))$

Theorem: null-axiom-proof-proves
$(\text{subset}\,(\text{list}\,(\text{fn}\,(\mathtt{0},\ \mathtt{0}),\ \text{p}\,(\mathtt{1},\ \mathtt{2})),\ symbols) \land \text{termp}\,(term,\ \mathtt{0},\ symbols))$
$\rightarrow\ \text{proves}\,(\text{null-axiom-proof}\,(term),$
$\quad\quad \text{f-not}\,(\text{isin}\,(term,\ \text{PHI})),$
$\quad\quad given,$
$\quad\quad defns,$
$\quad\quad symbols)$

Event: Enable null-axiom-proof; name this event 'g0375'.

Definition:
pair-axiom $(x,\ terma,\ termb)$
$=$ f-iff $(\text{isin}\,(x,\ \text{z-pair}\,(terma,\ termb)),$
$\quad\quad \text{f-or}\,(\text{f-equal}\,(x,\ terma),\ \text{f-equal}\,(x,\ termb)))$

Definition:
pair-axiom-proof $(x,\ terma,\ termb)$
$=$ list $(\mathtt{3},\ \text{list}\,(\mathtt{3},\ terma,\ termb,\ x),\ \text{pair-axiom}\,(x,\ terma,\ termb))$

THEOREM: pair-axiom-proof-proves

$$
\begin{array}{ll}
 & (\mathrm{termp}\,(x,\ \mathtt{0},\ symbols) \\
\land & \mathrm{termp}\,(terma,\ \mathtt{0},\ symbols) \\
\land & \mathrm{termp}\,(termb,\ \mathtt{0},\ symbols) \\
\land & \mathrm{subset}\,(\mathrm{list}\,(\mathrm{fn}\,(\mathtt{1},\ \mathtt{2}),\ \mathrm{p}\,(\mathtt{1},\ \mathtt{2})),\ symbols)) \\
\to & \mathrm{proves}\,(\text{pair-axiom-proof}\,(x,\ terma,\ termb), \\
 & \qquad\quad \text{f-iff}\,(\mathrm{isin}\,(x,\ \text{z-pair}\,(terma,\ termb)), \\
 & \qquad\qquad\quad \text{f-or}\,(\text{f-equal}\,(x,\ terma),\ \text{f-equal}\,(x,\ termb))), \\
 & \qquad\quad given, \\
 & \qquad\quad defns, \\
 & \qquad\quad symbols)
\end{array}
$$

EVENT: Enable pair-axiom-proof; name this event ‘g0376’.

DEFINITION:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{union-axiom}\,(x,\ terma,\ termb)} \\
= & \text{f-iff}\,(\mathrm{isin}\,(x,\ \text{z-union}\,(terma,\ termb)),\ \text{f-or}\,(\mathrm{isin}\,(x,\ terma),\ \mathrm{isin}\,(x,\ termb)))
\end{array}
$$

DEFINITION:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{union-axiom-proof}\,(x,\ terma,\ termb)} \\
= & \mathrm{list}\,(\mathtt{3},\ \mathrm{list}\,(\mathtt{4},\ terma,\ termb,\ x),\ \text{union-axiom}\,(x,\ terma,\ termb))
\end{array}
$$

THEOREM: union-axiom-proof-proves

$$
\begin{array}{ll}
 & (\mathrm{termp}\,(x,\ \mathtt{0},\ symbols) \\
\land & \mathrm{termp}\,(terma,\ \mathtt{0},\ symbols) \\
\land & \mathrm{termp}\,(termb,\ \mathtt{0},\ symbols) \\
\land & \mathrm{memb}\,(\mathrm{p}\,(\mathtt{1},\ \mathtt{2}),\ symbols) \\
\land & \mathrm{memb}\,(\mathrm{fn}\,(\mathtt{2},\ \mathtt{2}),\ symbols)) \\
\to & \mathrm{proves}\,(\text{union-axiom-proof}\,(x,\ terma,\ termb), \\
 & \qquad\quad \text{f-iff}\,(\mathrm{isin}\,(x,\ \text{z-union}\,(terma,\ termb)), \\
 & \qquad\qquad\quad \text{f-or}\,(\mathrm{isin}\,(x,\ terma),\ \mathrm{isin}\,(x,\ termb))), \\
 & \qquad\quad given, \\
 & \qquad\quad defns, \\
 & \qquad\quad symbols)
\end{array}
$$

EVENT: Enable union-axiom-proof; name this event ‘g0377’.

DEFINITION:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{ind-axiom}\,(form,\ x)} \\
= & \text{f-implies}\,(\text{f-and}\,(\mathrm{subst}\,(form,\ x,\ \textsc{phi},\ \mathtt{0}), \\
 & \qquad\qquad\qquad \forall\ x\ \text{f-implies}\,(\text{f-and}\,(\text{z-int}\,(x),\ form), \\
 & \qquad\qquad\qquad\qquad\qquad \mathrm{subst}\,(form,\ x,\ \text{z-succ}\,(x),\ \mathtt{0}))), \\
 & \qquad\quad \forall\ x\ \text{f-implies}\,(\text{z-int}\,(x),\ form))
\end{array}
$$

DEFINITION:
ind-axiom-proof$(form,\ x)$ = list (3, list (5, $form,\ x$), ind-axiom $(form,\ x)$)

EVENT: Disable predicate; name this event 'g0627'.

THEOREM: ind-axiom-proof-proves
(formula $(form,\ symbols)$
$\land$ variable (x)
$\land$ subset (list (fn (0, 0), fn (3, 1), p (2, 1)), $symbols$)
$\land$ $(concl$ = ind-axiom $(form,\ x)))$
$\to$ proves (ind-axiom-proof$(form,\ x)$, $concl,\ given,\ defns,\ symbols$)

EVENT: Enable predicate; name this event 'g0629'.

EVENT: Enable ind-axiom-proof; name this event 'g0378'.

EVENT: Enable prf; name this event 'g0379'.

THEOREM: proves-is-formula
proves $(pf,\ exp,\ given,\ defns,\ symbols) \to$ formula $(exp,\ symbols)$

THEOREM: proves-is-formula-again
$(\neg$ formula $(exp,\ symbols)) \to (\neg$ proves $(pf,\ exp,\ given,\ defns,\ symbols))$

EVENT: Enable proves; name this event 'g0380'.

```
;;;;;;;The commutativity of disjunction is the first example of a 'derived inference rule'.
;;;;;;;COMMUT-PROOF constructs a proof of "(B or A)" given a proof of "(A or B)".
```

DEFINITION:
commut-proof$(a,\ b,\ pf)$ = cut-proof $(a,\ b,\ a,\ pf$, prop-axiom-proof$(a))$

```
;;;;;;;The lemma below establishes that this is the required proof.  The definition
;;;;;;;of COMMUT-PROOF then becomes irrelevant and is disabled.
```

THEOREM: commut-proof-proves
proves $(pf$, f-or $(a,\ b)$, $given,\ defns,\ symbols$)
$\to$ proves (commut-proof$(a,\ b,\ pf)$, f-or $(b,\ a)$, $given,\ defns,\ symbols$)

EVENT: Enable commut-proof; name this event 'g2766'.

DEFINITION:

detach-proof (a, b, *pf1*, *pf2*)
= contrac-proof (b,
  cut-proof (a,
    b,
    b,
    commut-proof (b, a, expan-proof (b, a, *pf1*)),
    pf2))

EVENT: Enable proves-is-formula; name this event 'g0503'.

;;;;;;;Modus Ponens is proved as a derived inference rule below.

THEOREM: detach-proof-proves
(proves (*pf1*, a, *given*, *defns*, *symbols*)
$\land$ proves (*pf2*, f-implies (a, b), *given*, *defns*, *symbols*))
$\rightarrow$ proves (detach-proof (a, b, *pf1*, *pf2*), b, *given*, *defns*, *symbols*)

EVENT: Enable detach-proof; name this event 'g2767'.

THEOREM: formula-f-implies
formula (f-implies (x, y), *symbols*)
= (formula (x, *symbols*) $\land$ formula (y, *symbols*))

THEOREM: covering-f-implies
covering (f-implies (x, y), *var*, 0)
= append (covering (x, *var*, 0), covering (y, *var*, 0))

THEOREM: member-append
($x \in$ append (y, z)) = (($x \in y$) $\lor$ ($x \in z$))

THEOREM: collect-free-f-implies
collect-free (f-implies (x, y), 0)
= append (collect-free (x, 0), collect-free (y, 0))

EVENT: Enable f-implies; name this event 'g0344'.

DEFINITION:
list-implies (*list*, *conc*)
= **if** *list* $\simeq$ **nil** **then** *conc*
elseif cdr (*list*) $\simeq$ **nil** **then** f-implies (car (*list*), *conc*)
else f-implies (car (*list*), list-implies (cdr (*list*), *conc*)) **endif**

DEFINITION:
list-detach-proof (*alist*, b, *pflist*, *pf2*)

$=$ **if** *alist* $\simeq$ **nil then** *pf2*
elseif cdr (*alist*) $\simeq$ **nil then** detach-proof (car (*alist*), *b*, car (*pflist*), *pf2*)
else list-detach-proof (cdr (*alist*),
b,
cdr (*pflist*),
detach-proof (car (*alist*),
list-implies (cdr (*alist*), *b*),
car (*pflist*),
pf2)) **endif**

THEOREM: formula-list-implies
(formula (list-implies (*alist*, *b*), *symbols*) $\land$ listp (*alist*))
$\rightarrow$ formula (list-implies (cdr (*alist*), *b*), *symbols*)

```
;;;;;;;PROVES-LIST checks that PFLIST is a list of proofs of the elements of
;;;;;;;the list EXPLIST.
```

DEFINITION:
proves-list (*pflist*, *explist*, *given*, *defns*, *symbols*)
$=$ **if** *explist* $\simeq$ **nil then** *pflist* $=$ **nil**
else proves (car (*pflist*), car (*explist*), *given*, *defns*, *symbols*)
$\land$ proves-list (cdr (*pflist*),
cdr (*explist*),
given,
defns,
symbols) **endif**

THEOREM: detach-rule-corr
(proves-list (*pflist*, *alist*, *given*, *defns*, *symbols*)
$\land$ proves (*pf2*, list-implies (*alist*, *b*), *given*, *defns*, *symbols*)
$\land$ formula (*b*, *symbols*))
$\rightarrow$ proves (list-detach-proof (*alist*, *b*, *pflist*, *pf2*), *b*, *given*, *defns*, *symbols*)

EVENT: Enable list-detach-proof; name this event ‘g0220’.

DEFINITION:
rt-expan-proof (*a*, *b*, *pf*) $=$ commut-proof (*b*, *a*, expan-proof (*b*, *a*, *pf*))

THEOREM: rt-expan-proof-proves
(proves (*pf*, *a*, *given*, *defns*, *symbols*) $\land$ formula (*b*, *symbols*))
$\rightarrow$ proves (rt-expan-proof (*a*, *b*, *pf*), f-or (*a*, *b*), *given*, *defns*, *symbols*)

EVENT: Enable rt-expan-proof; name this event ‘g0227’.

```
;;;;;;;MAKE-DISJUNCT returns a disjunction of a given list of formulas.
```

DEFINITION:
make-disjunct ($flist$)
$=$ **if** $flist \simeq$ **nil then nil**
elseif cdr ($flist$) $\simeq$ **nil then** car ($flist$)
else f-or (car ($flist$), make-disjunct (cdr ($flist$))) **endif**

DEFINITION:
m1-proof (exp, $flist$, pf)
$=$ **if** $flist \simeq$ **nil then nil**
elseif cdr ($flist$) $\simeq$ **nil then** pf
elseif $exp =$ car ($flist$)
then rt-expan-proof (car ($flist$), make-disjunct (cdr ($flist$)), pf)
else expan-proof (car ($flist$),
make-disjunct (cdr ($flist$)),
m1-proof (exp, cdr ($flist$), pf)) **endif**

```
;;;;;;;The rule below asserts that if "A" is provable, so is any disjunction
;;;;;;;which contains "A" as a disjunct.
```

THEOREM: m1-proof-proves
(formula (make-disjunct ($flist$), $symbols$)
$\wedge$ ($exp \in flist$)
$\wedge$ proves (pf, exp, $given$, $defns$, $symbols$))
$\rightarrow$ proves (m1-proof (exp, $flist$, pf),
make-disjunct ($flist$),
$given$,
$defns$,
$symbols$)

EVENT: Enable m1-proof; name this event 'g0228'.

```
;;;;;;;Note that the lemma in the above form contains MAKE-DISJUNCT in the
;;;;;;;conclusion which might prevent a match with a disjunction of the
;;;;;;;form "(A or B)".  This is remedied by moving it to the hypothesis in
;;;;;;;the rephrased form below.  The lemma had to first be proved in the
;;;;;;;above form to get the induction right.
```

THEOREM: m1-proof-proves1
(formula (make-disjunct ($flist$), $symbols$)
$\wedge$ ($exp \in flist$)
$\wedge$ proves (pf, exp, $given$, $defns$, $symbols$)
$\wedge$ ($concl =$ make-disjunct ($flist$)))
$\rightarrow$ proves (m1-proof (exp, $flist$, pf), $concl$, $given$, $defns$, $symbols$)

EVENT: Enable m1-proof-proves; name this event 'g0504'.

DEFINITION:
rt-assoc-proof (a, b, c, pf)
= commut-proof (f-or (b, c),
a,
assoc-proof (b,
c,
a,
commut-proof (f-or (c, a),
b,
assoc-proof (c,
a,
b,
commut-proof (f-or (a,
b),
c,
pf)))))

THEOREM: rt-assoc-proof-proves
proves (pf, f-or (f-or (a, b), c), $given$, $defns$, $symbols$)
$\rightarrow$ proves (rt-assoc-proof (a, b, c, pf), f-or (a, f-or (b, c)), $given$, $defns$, $symbols$)

EVENT: Enable rt-assoc-proof; name this event 'g0231'.

DEFINITION:
insert-proof (a, b, c, pf)
= commut-proof (f-or (a, c),
b,
assoc-proof (a,
c,
b,
expan-proof (a, f-or (c, b), commut-proof (b, c, pf))))

THEOREM: insert-proof-proves
(proves (pf, f-or (b, c), $given$, $defns$, $symbols$) $\land$ formula (a, $symbols$))
$\rightarrow$ proves (insert-proof (a, b, c, pf), f-or (b, f-or (a, c)), $given$, $defns$, $symbols$)

EVENT: Enable insert-proof; name this event 'g0232'.

DEFINITION:
m2-proof-step ($exp1$, $exp2$, $flist$, pf)
= **if** $flist \simeq$ **nil then nil**
elseif cdr ($flist$) $\simeq$ **nil**
then if $exp2$ = car ($flist$) **then** pf
else nil endif

elseif *exp2* = car (*flist*)
then rt-assoc-proof (*exp1*,
exp2,
make-disjunct (cdr (*flist*)),
rt-expan-proof (f-or (*exp1*, *exp2*),
make-disjunct (cdr (*flist*)),
pf))
else insert-proof (car (*flist*),
exp1,
make-disjunct (cdr (*flist*)),
m2-proof-step (*exp1*, *exp2*, cdr (*flist*), *pf*)) **endif**

THEOREM: m2-proof-step-proves
(formula (make-disjunct (*flist*), *symbols*)
∧ (*exp2* ∈ *flist*)
∧ proves (*pf*, f-or (*exp1*, *exp2*), *given*, *defns*, *symbols*))
→ proves (m2-proof-step (*exp1*, *exp2*, *flist*, *pf*),
f-or (*exp1*, make-disjunct (*flist*)),
given,
defns,
symbols)

THEOREM: m2-proof-step-proves1
(formula (make-disjunct (*flist*), *symbols*)
∧ (*exp2* ∈ *flist*)
∧ proves (*pf*, f-or (*exp1*, *exp2*), *given*, *defns*, *symbols*)
∧ (*concl* = f-or (*exp1*, make-disjunct (*flist*))))
→ proves (m2-proof-step (*exp1*, *exp2*, *flist*, *pf*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable m2-proof-step; name this event ‘g0233’.

EVENT: Enable m2-proof-step-proves; name this event ‘g0234’.

DEFINITION:
m2-proof (*exp1*, *exp2*, *flist*, *pf*)
= **if** *flist* ≃ **nil then nil**
elseif *exp1* = *exp2* **then** m1-proof (*exp1*, *flist*, contrac-proof (*exp1*, *pf*))
elseif *exp1* = car (*flist*)
then m2-proof-step (*exp1*, *exp2*, cdr (*flist*), *pf*)
elseif *exp2* = car (*flist*)
then m2-proof-step (*exp2*, *exp1*, cdr (*flist*), commut-proof (*exp1*, *exp2*, *pf*))
else expan-proof (car (*flist*),
make-disjunct (cdr (*flist*)),
m2-proof (*exp1*, *exp2*, cdr (*flist*), *pf*)) **endif**

```
;;;;;;;The rule below asserts that if "(A or B)" is provable, so is any
;;;;;;;disjunction which contains "A" and "B" as disjuncts.
```

THEOREM: m2-proof-proves
(formula (make-disjunct (*flist*), *symbols*)
 ∧ (*exp1* ∈ *flist*)
 ∧ (*exp2* ∈ *flist*)
 ∧ proves (*pf*, f-or (*exp1*, *exp2*), *given*, *defns*, *symbols*))
→ proves (m2-proof (*exp1*, *exp2*, *flist*, *pf*),
 make-disjunct (*flist*),
 given,
 defns,
 symbols)

THEOREM: m2-proof-proves1
(formula (make-disjunct (*flist*), *symbols*)
 ∧ (*exp1* ∈ *flist*)
 ∧ (*exp2* ∈ *flist*)
 ∧ proves (*pf*, f-or (*exp1*, *exp2*), *given*, *defns*, *symbols*)
 ∧ (*concl* = make-disjunct (*flist*)))
→ proves (m2-proof (*exp1*, *exp2*, *flist*, *pf*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable m2-proof; name this event 'g0235'.

EVENT: Enable m2-proof-proves; name this event 'g0236'.

DEFINITION:
m3-proof (*exp1*, *exp2*, *flist2*, *pf*)
= contrac-proof (make-disjunct (*flist2*),
 contrac-proof (f-or (make-disjunct (*flist2*),
 make-disjunct (*flist2*)),
 m2-proof (f-or (make-disjunct (*flist2*),
 make-disjunct (*flist2*)),
 exp1,
 cons (f-or (make-disjunct (*flist2*),
 make-disjunct (*flist2*)),
 cons (make-disjunct (*flist2*),
 flist2)),
 assoc-proof (make-disjunct (*flist2*),
 make-disjunct (*flist2*),
 exp1,
 commut-proof (f-or (make-disjunct (*flist2*),
 exp1),
 make-disjunct (*flist2*),

```
m2-proof (f-or (make-disjunct (flist2),
                exp1),
          exp2,
          cons (f-or (make-disjunct (flist2),
                      exp1),
                flist2),
          assoc-proof (make-disjunct (flist2),
                       exp1,
                       exp2,
                       commut-proof (f-or (exp1,
                                           exp2),
                                     make-disjunct (flist2),
                                     pf))))))))
```

DEFINITION:
form-list (*flist*, *symbols*)
= **if** listp (*flist*)
then formula (car (*flist*), *symbols*) $\wedge$ form-list (cdr (*flist*), *symbols*)
else t endif

THEOREM: subset-cons
subset (x, y) $\rightarrow$ subset (x, cons (z, y))

THEOREM: formlist-formula-make-disj
(listp (*flist*) $\wedge$ form-list (*flist*, *symbols*))
$\rightarrow$ formula (make-disjunct (*flist*), *symbols*)

THEOREM: m3-proof-proves
(proves (*pf*, make-disjunct (cons (f-or (*exp1*, *exp2*), *flist2*)), *given*, *defns*, *symbols*)
$\wedge$ (*exp1* $\in$ *flist2*)
$\wedge$ (*exp2* $\in$ *flist2*))
$\rightarrow$ proves (m3-proof (*exp1*, *exp2*, *flist2*, *pf*),
make-disjunct (*flist2*),
given,
defns,
symbols)

EVENT: Enable m3-proof; name this event ‘g0222’.

THEOREM: m3-proof-proves1
(proves (*pf*, make-disjunct (cons (f-or (*exp1*, *exp2*), *flist2*)), *given*, *defns*, *symbols*)
$\wedge$ (*exp1* $\in$ *flist2*)
$\wedge$ (*exp2* $\in$ *flist2*)
$\wedge$ (*concl* = make-disjunct (*flist2*)))
$\rightarrow$ proves (m3-proof (*exp1*, *exp2*, *flist2*, *pf*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable m3-proof-proves; name this event 'g0229'.

DEFINITION:
m-proof (*flist1*, *flist2*, *pf*)
= **if** *flist1* ≃ **nil** **then nil**
elseif cdr (*flist1*) ≃ **nil** **then** m1-proof (car (*flist1*), *flist2*, *pf*)
elseif cddr (*flist1*) ≃ **nil**
then m2-proof (car (*flist1*), cadr (*flist1*), *flist2*, *pf*)
else m3-proof (car (*flist1*),
cadr (*flist1*),
flist2,
m-proof (cons (f-or (car (*flist1*), cadr (*flist1*)),
cddr (*flist1*)),
cons (f-or (car (*flist1*), cadr (*flist1*)),
flist2),
assoc-proof (car (*flist1*),
cadr (*flist1*),
make-disjunct (cddr (*flist1*)),
pf))) **endif**

```
;;;;;;;The rule below is the important lemma on disjuncts which asserts that
;;;;;;;if a disjunction of formulas {"A1",...,"Am"} is provable, so is any
;;;;;;;disjunction of formulas {"B1",...,"Bn"} which contains all the "Ai".
```

THEOREM: m-proof-proves
(form-list (*flist2*, *symbols*)
∧ listp (*flist1*)
∧ listp (*flist2*)
∧ subset (*flist1*, *flist2*)
∧ proves (*pf*, make-disjunct (*flist1*), *given*, *defns*, *symbols*))
→ proves (m-proof (*flist1*, *flist2*, *pf*),
make-disjunct (*flist2*),
given,
defns,
symbols)

DEFINITION:
termp1 (*exp*, *flg*)
= **if** *flg* ≃ `0`
then if listp (*exp*)
then function (car (*exp*))
∧ (length (cdr (*exp*)) = degree (car (*exp*)))
∧ termp1 (cdr (*exp*), `1`)
else variable (*exp*) **endif**

elseif *exp* $\simeq$ **nil then t**
else termp1 (car (*exp*), `0`) $\wedge$ termp1 (cdr (*exp*), `1`) **endif**

THEOREM: termp-termp1
termp (*exp*, *flg*, *symbols*) $\rightarrow$ termp1 (*exp*, *flg*)

DEFINITION:
atomp1 (*exp*)
= (predicate (car (*exp*))
  $\wedge$ (length (cdr (*exp*)) = degree (car (*exp*)))
  $\wedge$ termp1 (cdr (*exp*), `1`))

EVENT: Disable atomp; name this event 'g0345'.

THEOREM: atomp-atomp1
atomp (*exp*, *symbols*) $\rightarrow$ atomp1 (*exp*)

DEFINITION:
dble-neg-type (*exp*) = (f-notp (*exp*) $\wedge$ f-notp (arg (*exp*)))

DEFINITION:
neg (*exp1*, *exp2*) = ((*exp1* = f-not (*exp2*)) $\vee$ (*exp2* = f-not (*exp1*)))

DEFINITION:
neg-list (*exp*, *list*)
= **if** f-notp (*exp*) **then** arg (*exp*) $\in$ *list*
else f-not (*exp*) $\in$ *list* **endif**

DEFINITION:
elem-form (*exp*)
= (atomp1 (*exp*)
  $\vee$ (forsomep (*exp*) $\wedge$ (*exp* = forsome (bind (*exp*), body (*exp*)))))

DEFINITION:
neg-elem-form (*exp*) = (f-notp (*exp*) $\wedge$ elem-form (arg (*exp*)))

DEFINITION:
prop-atomp (*exp*) = (elem-form (*exp*) $\vee$ neg-elem-form (*exp*))

DEFINITION:
form-count (*exp*)
= **if** f-notp (*exp*) **then** 1 + form-count (arg (*exp*))
elseif f-orp (*exp*)
then 1 + (form-count (arg1 (*exp*)) + form-count (arg2 (*exp*)))
else `1` **endif**

DEFINITION:
list-count (*list*)
$=$ **if** listp (*list*) **then** form-count (car (*list*)) $+$ list-count (cdr (*list*))
else 0 **endif**

DEFINITION: nor-type (*exp*) $=$ (f-notp (*exp*) $\wedge$ f-orp (arg (*exp*)))

THEOREM: lessp-list-count
listp (*flist*) $\rightarrow$ (list-count (cdr (*flist*)) $<$ list-count (*flist*))

THEOREM: f-orp-list-count
(f-orp (car (*flist*)) $\wedge$ listp (*flist*))
$\rightarrow$ (list-count (cons (arg1 (car (*flist*)), cons (arg2 (car (*flist*)), cdr (*flist*))))
$<$ list-count (*flist*))

THEOREM: nor-type-list-count1
(listp (*flist*) $\wedge$ nor-type (car (*flist*)))
$\rightarrow$ (list-count (cons (f-not (arg1 (arg (car (*flist*)))), cdr (*flist*)))
$<$ list-count (*flist*))

THEOREM: nor-type-list-count2
(listp (*flist*) $\wedge$ nor-type (car (*flist*)))
$\rightarrow$ (list-count (cons (f-not (arg2 (arg (car (*flist*)))), cdr (*flist*)))
$<$ list-count (*flist*))

THEOREM: dble-neg-list-count
(listp (*flist*) $\wedge$ dble-neg-type (car (*flist*)))
$\rightarrow$ (list-count (cons (arg (arg (car (*flist*))), cdr (*flist*)))
$<$ list-count (*flist*))

EVENT: Enable list-count; name this event ‘g0240’.

```
;;;;;;;TAUTOLOGYP1 is the tautology-checker.  It recurses through a list of
;;;;;;;formulas FLIST and builds up a list of literals AUXLIST.  If AUXLIST
;;;;;;;is ever to contain a literal and its negation, TAUTOLOGYP1 returns T.
;;;;;;;Otherwise, it returns F.
```

DEFINITION:
tautologyp1 (*flist*, *auxlist*)
$=$ **if** listp (*flist*)
then if f-orp (car (*flist*))
then tautologyp1 (cons (arg1 (car (*flist*)),
cons (arg2 (car (*flist*)), cdr (*flist*))),
auxlist)
elseif nor-type (car (*flist*))

then tautologyp1 (cons (f-not (arg1 (arg (car (*flist*)))), cdr (*flist*)),
auxlist)
∧ tautologyp1 (cons (f-not (arg2 (arg (car (*flist*)))),
cdr (*flist*)),
auxlist)
elseif dble-neg-type (car (*flist*))
then tautologyp1 (cons (arg (arg (car (*flist*))), cdr (*flist*)), *auxlist*)
else neg-list (car (*flist*), *auxlist*)
∨ tautologyp1 (cdr (*flist*),
cons (car (*flist*), *auxlist*)) **endif**
else f endif

THEOREM: form-list-append
(form-list (x, *symbols*) ∧ form-list (y, *symbols*))
→ form-list (append (x, y), *symbols*)

DEFINITION:
neg-proof (*exp1*, *exp2*)
= **if** *exp1* = f-not (*exp2*) **then** prop-axiom-proof (*exp2*)
else commut-proof (*exp2*, *exp1*, prop-axiom-proof (*exp1*)) **endif**

THEOREM: neg-proof-proves
(formula (*exp1*, *symbols*)
∧ formula (*exp2*, *symbols*)
∧ neg (*exp1*, *exp2*)
∧ (*concl* = f-or (*exp1*, *exp2*)))
→ proves (neg-proof (*exp1*, *exp2*), *concl*, *given*, *defns*, *symbols*)

DEFINITION:
neg-list-proof (*exp*, *flist*)
= **if** f-not (*exp*) ∈ *flist*
then m2-proof (*exp*,
f-not (*exp*),
cons (*exp*, *flist*),
neg-proof (*exp*, f-not (*exp*)))
else m2-proof (*exp*,
arg (*exp*),
cons (*exp*, *flist*),
neg-proof (*exp*, arg (*exp*))) **endif**

THEOREM: neg-list-proof-proves
(formula (*exp*, *symbols*)
∧ form-list (*flist*, *symbols*)
∧ neg-list (*exp*, *flist*)
∧ (*concl* = make-disjunct (cons (*exp*, *flist*))))
→ proves (neg-list-proof (*exp*, *flist*), *concl*, *given*, *defns*, *symbols*)

THEOREM: subset-ident
subset $(x,\ x)$

THEOREM: subset-car
subset $(x,$ cons $(y,\ x))$

THEOREM: subset-append
subset (cons (*exp*, *list2*), append (cons (*exp*, *list1*), *list2*))

THEOREM: m-proof-proves1
(form-list (*flist2*, *symbols*)
$\land$ subset (*flist1*, *flist2*)
$\land$ proves (*pf*, make-disjunct (*flist1*), *given*, *defns*, *symbols*)
$\land$ (*concl* = make-disjunct (*flist2*)))
$\rightarrow$ proves (m-proof (*flist1*, *flist2*, *pf*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable m-proof-proves; name this event 'g0505'.

EVENT: Enable neg-list-proof; name this event 'g0256'.

DEFINITION:
prop-atom-proof1 (*flist1*, *flist2*)
= m-proof (cons (car (*flist1*), *flist2*),
append (*flist1*, *flist2*),
neg-list-proof (car (*flist1*), *flist2*))

THEOREM: prop-atom-proof1-proves
(form-list (*flist1*, *symbols*)
$\land$ listp (*flist1*)
$\land$ form-list (*flist2*, *symbols*)
$\land$ neg-list (car (*flist1*), *flist2*)
$\land$ (*concl* = make-disjunct (append (*flist1*, *flist2*))))
$\rightarrow$ proves (prop-atom-proof1 (*flist1*, *flist2*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable prop-atom-proof1; name this event 'g0259'.

THEOREM: subset-append-car
subset (append (*list1*, cons (*exp*, *list2*)), append (cons (*exp*, *list1*), *list2*))

THEOREM: form-list-append-car
(form-list (cons (*exp*, *list1*), *symbols*)
$\land$ ((*list2* = **nil**) $\lor$ form-list (*list2*, *symbols*)))
$\rightarrow$ form-list (append (*list1*, cons (*exp*, *list2*)), *symbols*)

Definition:
prop-atom-proof2 (*flist1*, *flist2*, *pf*)
= m-proof (append (cdr (*flist1*), cons (car (*flist1*), *flist2*)),
append (*flist1*, *flist2*),
pf)

Theorem: prop-atom-proof2-proves
(form-list (*flist1*, *symbols*)
$\land$ listp (*flist1*)
$\land$ form-list (*flist2*, *symbols*)
$\land$ proves (*pf*,
make-disjunct (append (cdr (*flist1*), cons (car (*flist1*), *flist2*))),
given,
defns,
symbols)
$\land$ (*concl* = make-disjunct (append (*flist1*, *flist2*))))
$\rightarrow$ proves (prop-atom-proof2 (*flist1*, *flist2*, *pf*), *concl*, *given*, *defns*, *symbols*)

Event: Enable prop-atom-proof2; name this event 'g0258'.

Definition:
cancel-proof (a, b, *pf1*, *pf2*)
= contrac-proof (b, cut-proof (a, b, b, *pf2*, rt-expan-proof (f-not (a), b, *pf1*)))

Theorem: cancel-proof-proves
(proves (*pf1*, f-not (a), *given*, *defns*, *symbols*)
$\land$ proves (*pf2*, f-or (a, b), *given*, *defns*, *symbols*))
$\rightarrow$ proves (cancel-proof (a, b, *pf1*, *pf2*), b, *given*, *defns*, *symbols*)

Event: Enable cancel-proof; name this event 'g0255'.

Definition:
nlistp-nor-type-proof (a, b, *pf1*, *pf2*)
= cancel-proof (b,
f-not (f-or (a, b)),
pf2,
cancel-proof (a,
f-or (b, f-not (f-or (a, b))),
pf1,
m-proof (list (f-not (f-or (a, b)), a, b),
list (a, b, f-not (f-or (a, b))),
prop-axiom-proof (f-or (a, b)))))

Definition:

```
listp-nor-type-proof(a, b, c, pf1, pf2)
=   m-proof(list(f-not(f-or(a, b)), c, c),
            list(f-not(f-or(a, b)), c),
            rt-assoc-proof(f-not(f-or(a, b)),
                           c,
                           c,
                           cut-proof(b,
                                     f-or(f-not(f-or(a, b)), c),
                                     c,
                                     rt-assoc-proof(b,
                                                    f-not(f-or(a, b)),
                                                    c,
                                                    cut-proof(a,
                                                              f-or(b,
                                                                   f-not(f-or(a,
                                                                              b))),
                                                              c,
                                                              m-proof(list(f-not(f-or(a,
                                                                                      b)),
                                                                           a,
                                                                           b),
                                                                      list(a,
                                                                           b,
                                                                           f-not(f-or(a,
                                                                                      b))),
                                                                      prop-axiom-proof(f-or(a,
                                                                                            b))),
                                                              pf1)),
                                     pf2)))
```

DEFINITION:
nor-proof(a, b, *clist*, *pf1*, *pf2*)
= **if** *clist* $\simeq$ **nil** **then** nlistp-nor-type-proof(a, b, *pf1*, *pf2*)
else listp-nor-type-proof(a, b, make-disjunct(*clist*), *pf1*, *pf2*) **endif**

DEFINITION:

```
nlistp-dble-neg-proof(a, pf)
=   contrac-proof(f-not(f-not(a)),
                  cut-proof(a,
                            f-not(f-not(a)),
                            f-not(f-not(a)),
                            rt-expan-proof(a, f-not(f-not(a)), pf),
                            commut-proof(f-not(f-not(a)),
                                         f-not(a),
                                         prop-axiom-proof(f-not(a)))))
```

Definition:
listp-dble-neg-proof (a, c, pf)
```
=  commut-proof (c,
                 f-not (f-not (a)),
                 cut-proof (a,
                            c,
                            f-not (f-not (a)),
                            pf,
                            commut-proof (f-not (f-not (a)),
                                          f-not (a),
                                          prop-axiom-proof (f-not (a)))))
```

Definition:
dble-neg-proof (a, $clist$, pf)
```
=  if clist ≃ nil then nlistp-dble-neg-proof (a, pf)
   else listp-dble-neg-proof (a, make-disjunct (clist), pf) endif
```

Definition:
f-orp-proof (a, b, $clist$, pf)
```
=  if clist ≃ nil then pf
   else assoc-proof (a, b, make-disjunct (clist), pf) endif
```

```
;;;;;;;TAUT-PROOF1 is the proof-constructor that constructs the proof for a formula
;;;;;;;asserted to be a tautology by TAUTOLOGYP1.  TAUT-PROOF1 and TAUTOLOGYP1 recurse
;;;;;;;in the same way.
```

Definition:
taut-proof1 ($flist$, $auxlist$)
```
=  if listp (flist)
   then if f-orp (car (flist))
        then f-orp-proof (arg1 (car (flist)),
                          arg2 (car (flist)),
                          append (cdr (flist), auxlist),
                          taut-proof1 (cons (arg1 (car (flist)),
                                             cons (arg2 (car (flist)),
                                                   cdr (flist))),
                                       auxlist))
        elseif nor-type (car (flist))
        then nor-proof (arg1 (arg (car (flist))),
                        arg2 (arg (car (flist))),
                        append (cdr (flist), auxlist),
                        taut-proof1 (cons (f-not (arg1 (arg (car (flist)))),
                                           cdr (flist)),
                                     auxlist),
                        taut-proof1 (cons (f-not (arg2 (arg (car (flist)))),
```

```
                                             cdr (flist)),
                                        auxlist))
        elseif dble-neg-type (car (flist))
        then dble-neg-proof (arg (arg (car (flist))),
                             append (cdr (flist), auxlist),
                             taut-proof1 (cons (arg (arg (car (flist))),
                                               cdr (flist)),
                                         auxlist))
        elseif neg-list (car (flist), auxlist)
        then prop-atom-proof1 (flist, auxlist)
        else prop-atom-proof2 (flist,
                               auxlist,
                               taut-proof1 (cdr (flist),
                                            cons (car (flist), auxlist))) endif
    else nil endif
```

```
;;;;;;;The lemma below is a form of the tautology theorem.  If (TAUTOLOGYP1 FLIST AUXLIST)
;;;;;;;is T, then (TAUT-PROOF1 FLIST AUXLIST) constructs a correct proof of
;;;;;;;the disjunction of the formulas in FLIST and AUXLIST.  We still have to
;;;;;;;show that the tautology-checker TAUTOLOGYP1 is correct w.r.t the truth-table
;;;;;;;definition of a tautology.
```

THEOREM: taut-thm1

```
 (form-list (flist, symbols)
  ∧  form-list (auxlist, symbols)
  ∧  tautologyp1 (flist, auxlist))
 →  proves (taut-proof1 (flist, auxlist),
            make-disjunct (append (flist, auxlist)),
            given,
            defns,
            symbols)
```

EVENT: Enable taut-proof1; name this event ‘g0275’.

THEOREM: taut-thm2

```
 (form-list (flist, symbols)
  ∧  form-list (auxlist, symbols)
  ∧  tautologyp1 (flist, auxlist)
  ∧  (concl = make-disjunct (append (flist, auxlist))))
 →  proves (taut-proof1 (flist, auxlist), concl, given, defns, symbols)
```

```
;;;;;;;EVAL evaluates the truth-value of the Boolean expression EXP,
;;;;;;;where the truth-value assigned to atoms are specified in ALIST.
```

DEFINITION:
eval (*exp*, *alist*)
$=$ **if** f-notp (*exp*) **then** $\neg$ eval (arg (*exp*), *alist*)
elseif f-orp (*exp*)
then eval (arg1 (*exp*), *alist*) $\vee$ eval (arg2 (*exp*), *alist*)
else *exp* $\in$ *alist* **endif**

THEOREM: member-eval
((*exp* $\in$ *flist*) $\wedge$ eval (*exp*, *alist*)) $\rightarrow$ eval (make-disjunct (*flist*), *alist*)

THEOREM: eval-make-disjunct
(listp (*list1*) $\wedge$ listp (*list2*))
$\rightarrow$ (eval (make-disjunct (append (*list1*, *list2*)), *alist*)
$=$ (eval (make-disjunct (*list1*), *alist*)
$\vee$ eval (make-disjunct (*list2*), *alist*)))

THEOREM: neg-list-eval
(listp (*flist1*) $\wedge$ neg-list (car (*flist1*), *flist2*))
$\rightarrow$ eval (make-disjunct (append (*flist1*, *flist2*)), *alist*)

THEOREM: eval-prop-atomp
(listp (*flist1*)
$\wedge$ eval (make-disjunct (append (cdr (*flist1*), cons (car (*flist1*), *flist2*))),
alist))
$\rightarrow$ eval (make-disjunct (append (*flist1*, *flist2*)), *alist*)

THEOREM: eval-prop-atomp1
(listp (*flist1*)
$\wedge$ eval (make-disjunct (append (cdr (*flist1*), cons (car (*flist1*), *flist2*))),
alist)
$\wedge$ (*concl* $=$ make-disjunct (append (*flist1*, *flist2*))))
$\rightarrow$ eval (*concl*, *alist*)

THEOREM: neg-list-eval1
(listp (*flist1*)
$\wedge$ neg-list (car (*flist1*), *flist2*)
$\wedge$ (*concl* $=$ make-disjunct (append (*flist1*, *flist2*))))
$\rightarrow$ eval (*concl*, *alist*)

```
;;;;;;;The lemma below is one half of the correctness of the tautology-checker.
;;;;;;;It asserts that any formula checked to be a tautology, evaluates to
;;;;;;;true under any assignment of truth-values to the atoms.
```

THEOREM: taut-eval
tautologyp1 (*flist*, *auxlist*)
$\rightarrow$ eval (make-disjunct (append (*flist*, *auxlist*)), *alist*)

```
;;;;;;;FALSIFY attempts to return a falsifying assignment corresponding to a list of literals.
```

Definition:

falsify (*list*)
= **if** *list* $\simeq$ **nil then nil**
elseif f-notp (car (*list*)) **then** cons (arg (car (*list*)), falsify (cdr (*list*)))
else falsify (cdr (*list*)) **endif**

```
;;;;;;;FALSIFY-TAUT attempts to return an assignment which falsifies a non-tautology.
```

Definition:

```
falsify-taut (flist, auxlist)
=   if listp (flist)
    then if f-orp (car (flist))
         then falsify-taut (cons (arg1 (car (flist)),
                                  cons (arg2 (car (flist)), cdr (flist))),
                            auxlist)
         elseif nor-type (car (flist))
         then if falsify-taut (cons (f-not (arg2 (arg (car (flist)))),
                                     cdr (flist)),
                               auxlist)
              then falsify-taut (cons (f-not (arg2 (arg (car (flist)))),
                                       cdr (flist)),
                                 auxlist)
              else falsify-taut (cons (f-not (arg1 (arg (car (flist)))),
                                       cdr (flist)),
                                 auxlist) endif
         elseif dble-neg-type (car (flist))
         then falsify-taut (cons (arg (arg (car (flist))), cdr (flist)), auxlist)
         elseif neg-list (car (flist), auxlist) then f
         else falsify-taut (cdr (flist), cons (car (flist), auxlist)) endif
    else falsify (auxlist) endif
```

Theorem: not-falsify-taut

tautologyp1 (*flist*, *auxlist*) = ($\neg$ falsify-taut (*flist*, *auxlist*))

Theorem: eval-member

(($\neg$ f-notp (*exp*)) $\land$ ($\neg$ f-orp (*exp*)) $\land$ (*exp* $\in$ *alist*))
$\rightarrow$ eval (*exp*, *alist*)

Definition:

```
prop-atomp-list (flist)
=   if listp (flist)
    then (¬ f-orp (car (flist)))
         ∧  (¬ nor-type (car (flist)))
```

$\wedge$ (¬ dble-neg-type (car (*flist*)))
$\wedge$ (¬ neg-list (car (*flist*), cdr (*flist*)))
$\wedge$ prop-atomp-list (cdr (*flist*))
else t endif

THEOREM: member-falsify
$(exp \in \text{falsify}(list)) = (\text{f-not}(exp) \in list)$

THEOREM: alist-cons-eval2
$((x \notin flist)$
$\wedge$ listp (*flist*)
$\wedge$ prop-atomp-list (*flist*)
$\wedge$ (¬ eval (make-disjunct (*flist*), *alist*)))
$\rightarrow$ (¬ eval (make-disjunct (*flist*), cons (*x*, *alist*)))

THEOREM: prop-atomp-list-eval
(listp (*flist*) $\wedge$ prop-atomp-list (*flist*))
$\rightarrow$ (¬ eval (make-disjunct (*flist*), falsify (*flist*)))

THEOREM: not-eval-prop-atomp
eval (make-disjunct (append (*flist1*, cons (*exp*, *flist2*))), *alist*)
$=$ eval (make-disjunct (cons (*exp*, append (*flist1*, *flist2*))), *alist*)

```
;;;;;;;The lemma below is the second half of the correctness of the tautology-checker.
;;;;;;;It asserts that if AUXLIST does not contain both a literal and its negation,
;;;;;;;then any formula which is checked to not be a tautology is falsifiable.
```

THEOREM: not-taut-false
(prop-atomp-list (*auxlist*) $\wedge$ (¬ tautologyp1 (*flist*, *auxlist*)))
$\rightarrow$ (¬ eval (make-disjunct (append (*flist*, *auxlist*)),
falsify-taut (*flist*, *auxlist*)))

```
;;;;;;;TAUTOLOGYP is TAUTOLOGYP1 with AUXLIST set to NIL.
```

DEFINITION: tautologyp (*flist*) = tautologyp1 (*flist*, **nil**)

DEFINITION: taut-proof (*flist*) = taut-proof1 (*flist*, **nil**)

THEOREM: form-list-append-nil
make-disjunct (append (*flist*, **nil**)) = make-disjunct (*flist*)

THEOREM: tautology-theorem
(form-list (*flist*, *symbols*)
$\wedge$ tautologyp (*flist*)
$\wedge$ (*concl* = make-disjunct (*flist*)))
$\rightarrow$ proves (taut-proof (*flist*), *concl*, *given*, *defns*, *symbols*)

THEOREM: taut-eval2
$(\text{tautologyp1}(\mathit{flist},\ \mathit{auxlist})$
$\ \wedge\ \ (\mathit{concl} = \text{make-disjunct}(\text{append}(\mathit{flist},\ \mathit{auxlist}))))$
$\rightarrow\ \ \text{eval}(\mathit{concl},\ \mathit{alist})$

THEOREM: tautologies-are-true
$(\text{listp}(\mathit{flist}) \wedge \text{tautologyp}(\mathit{flist})) \rightarrow \text{eval}(\text{make-disjunct}(\mathit{flist}),\ \mathit{alist})$

THEOREM: not-taut-falsify2
$(\text{prop-atomp-list}(\mathit{auxlist})$
$\ \wedge\ \ (\neg\ \text{tautologyp1}(\mathit{flist},\ \mathit{auxlist}))$
$\ \wedge\ \ (\mathit{concl} = \text{make-disjunct}(\text{append}(\mathit{flist},\ \mathit{auxlist}))))$
$\rightarrow\ \ (\neg\ \text{eval}(\mathit{concl},\ \text{falsify-taut}(\mathit{flist},\ \mathit{auxlist})))$

THEOREM: truths-are-tautologies
$(\neg\ \text{tautologyp}(\mathit{flist}))$
$\rightarrow\ \ (\neg\ \text{eval}(\text{make-disjunct}(\mathit{flist}),\ \text{falsify-taut}(\mathit{flist},\ \mathbf{nil})))$

DEFINITION:
$\text{lis-not}(\mathit{flist})$
$=\ \ \mathbf{if}\ \mathit{flist} \simeq \mathbf{nil}\ \ \mathbf{then}\ \mathbf{nil}$
$\quad\ \mathbf{else}\ \text{cons}(\text{f-not}(\text{car}(\mathit{flist})),\ \text{lis-not}(\text{cdr}(\mathit{flist})))\ \mathbf{endif}$

DEFINITION:
$\text{taut-conseq}(\mathit{flist},\ \mathit{exp}) = \text{tautologyp}(\text{append}(\text{lis-not}(\mathit{flist}),\ \text{cons}(\mathit{exp},\ \mathbf{nil})))$

DEFINITION:
$\text{tautconseq-proof}(\mathit{flist},\ \mathit{exp},\ \mathit{pflist})$
$=\ \ \text{list-detach-proof}(\mathit{flist},$
$\qquad\qquad \mathit{exp},$
$\qquad\qquad \mathit{pflist},$
$\qquad\qquad \text{taut-proof}(\text{append}(\text{lis-not}(\mathit{flist}),\ \text{cons}(\mathit{exp},\ \mathbf{nil}))))$

EVENT: Disable f-implies; name this event ‘g0873’.

THEOREM: eval-f-implies
$\text{eval}(\text{f-implies}(x,\ y),\ \mathit{alist}) = (\text{eval}(x,\ \mathit{alist}) \rightarrow \text{eval}(y,\ \mathit{alist}))$

THEOREM: list-implies-reduc
$\text{list-implies}(\mathit{flist},\ \mathit{exp})$
$=\ \ \text{make-disjunct}(\text{append}(\text{lis-not}(\mathit{flist}),\ \text{cons}(\mathit{exp},\ \mathbf{nil})))$

EVENT: Enable f-implies; name this event ‘g0874’.

THEOREM: append-exp-form-list
(form-list (*flist*, *symbols*) $\wedge$ formula (*exp*, *symbols*))
$\rightarrow$ form-list (append (lis-not (*flist*), cons (*exp*, **nil**)), *symbols*)

THEOREM: eval-tautologyp
eval (make-disjunct (*flist*), falsify-taut (*flist*, **nil**)) $\rightarrow$ tautologyp (*flist*)

THEOREM: proves-list-form-list
proves-list (*pflist*, *flist*, *given*, *defns*, *symbols*) $\rightarrow$ form-list (*flist*, *symbols*)

THEOREM: taut-conseq-proves
(formula (*exp*, *symbols*)
$\wedge$ taut-conseq (*flist*, *exp*)
$\wedge$ proves-list (*pflist*, *flist*, *given*, *defns*, *symbols*))
$\rightarrow$ proves (tautconseq-proof (*flist*, *exp*, *pflist*), *exp*, *given*, *defns*, *symbols*)

EVENT: Enable tautconseq-proof; name this event 'g0276'.

THEOREM: eval-tautconseq
(form-list (*flist*, *symbols*)
$\wedge$ formula (*exp*, *symbols*)
$\wedge$ eval (make-disjunct (append (lis-not (*flist*), cons (*exp*, **nil**))),
falsify-taut (append (lis-not (*flist*), cons (*exp*, **nil**)), **nil**)))
$\rightarrow$ taut-conseq (*flist*, *exp*)

EVENT: Enable taut-conseq; name this event 'g0277'.

EVENT: Enable falsify-taut; name this event 'g0282'.

```
;;;;;;;The lemma below is the frequently used form of the tautology theorem.
;;;;;;;It asserts that if EXP is a tautological consequence of a list of
;;;;;;;provable formulas FLIST, then EXP is provable.  "B" is the
;;;;;;;tautological consequence of {"A1",...,"An"} if "(A1->(...->(An->B)))" is
;;;;;;;a tautology.  A formula is a tautology if it evaluates to true under its
;;;;;;;falsifying assignment.
```

THEOREM: eval-tautconseq-proof-proves
(formula (*exp*, *symbols*)
$\wedge$ eval (make-disjunct (append (lis-not (*flist*), cons (*exp*, **nil**))),
falsify-taut (append (lis-not (*flist*), cons (*exp*, **nil**)), **nil**))
$\wedge$ proves-list (*pflist*, *flist*, *given*, *defns*, *symbols*))
$\rightarrow$ proves (tautconseq-proof (*flist*, *exp*, *pflist*), *exp*, *given*, *defns*, *symbols*)

EVENT: Enable taut-conseq-proves; name this event 'g0283'.

```
;;;;;;;The rules below are some properties of the various logical connectives which
;;;;;;;follow as straightforward consequences of the tautology theorem.
```

DEFINITION:
f-iff-mp-proof1 (a, b, $pf1$, $pf2$)
$=$ tautconseq-proof (list (f-iff (a, b), a), b, list ($pf1$, $pf2$))

```
;;;;;;;The rule of detachment for ‘<->’ (first form).
```

THEOREM: f-iff-mp-proves1
(proves ($pf1$, f-iff (a, b), $given$, $defns$, $symbols$)
$\land$ proves ($pf2$, a, $given$, $defns$, $symbols$))
$\rightarrow$ proves (f-iff-mp-proof1 (a, b, $pf1$, $pf2$), b, $given$, $defns$, $symbols$)

EVENT: Enable f-iff-mp-proof1; name this event ‘g0284’.

DEFINITION:
f-iff-mp-proof2 (a, b, $pf1$, $pf2$)
$=$ tautconseq-proof (list (f-iff (a, b), b), a, list ($pf1$, $pf2$))

```
;;;;;;;The rule of detachment for ‘<->’ (second form).
```

THEOREM: f-iff-mp-proves2
(proves ($pf1$, f-iff (a, b), $given$, $defns$, $symbols$)
$\land$ proves ($pf2$, b, $given$, $defns$, $symbols$))
$\rightarrow$ proves (f-iff-mp-proof2 (a, b, $pf1$, $pf2$), a, $given$, $defns$, $symbols$)

DEFINITION:
trans-implies-proof (a, b, c, $pf1$, $pf2$)
$=$ tautconseq-proof (list (f-implies (a, b), f-implies (b, c)),
f-implies (a, c),
list ($pf1$, $pf2$))

```
;;;;;;;The transitivity of ‘->’.
```

THEOREM: transitivity-of-implies
(proves ($pf1$, f-implies (a, b), $given$, $defns$, $symbols$)
$\land$ proves ($pf2$, f-implies (b, c), $given$, $defns$, $symbols$))
$\rightarrow$ proves (trans-implies-proof (a, b, c, $pf1$, $pf2$),
f-implies (a, c),
$given$,
$defns$,
$symbols$)

EVENT: Enable trans-implies-proof; name this event ‘g0285’.

`;;;;;;The transitivity of ‘<->’.`

Definition:
trans-f-iff-proof (a, b, c, *pf1*, *pf2*)
= tautconseq-proof (list (f-iff (a, b), f-iff (b, c)), f-iff (a, c), list (*pf1*, *pf2*))

Theorem: transitivity-of-iff
(proves (*pf1*, f-iff (a, b), *given*, *defns*, *symbols*)
$\land$ proves (*pf2*, f-iff (b, c), *given*, *defns*, *symbols*))
$\rightarrow$ proves (trans-f-iff-proof (a, b, c, *pf1*, *pf2*),
f-iff (a, c),
given,
defns,
symbols)

Event: Enable trans-f-iff-proof; name this event ‘g0286’.

Definition:
and-split-proof1 (a, b, *pf1*)
= tautconseq-proof (list (f-and (a, b)), a, list (*pf1*))

`;;;;;;Derive "A" from "(A and B)".`

Theorem: and-split1
proves (*pf1*, f-and (a, b), *given*, *defns*, *symbols*)
$\rightarrow$ proves (and-split-proof1 (a, b, *pf1*), a, *given*, *defns*, *symbols*)

Definition:
and-split-proof2 (a, b, *pf1*)
= tautconseq-proof (list (f-and (a, b)), b, list (*pf1*))

`;;;;;;Derive "B" from "(A and B)".`

Theorem: and-split2
proves (*pf1*, f-and (a, b), *given*, *defns*, *symbols*)
$\rightarrow$ proves (and-split-proof2 (a, b, *pf1*), b, *given*, *defns*, *symbols*)

Definition:
and-join-proof (a, b, *pf1*, *pf2*)
= tautconseq-proof (list (a, b), f-and (a, b), list (*pf1*, *pf2*))

`;;;;;;Derive "(A and B)" from "A" and "B".`

Theorem: and-join
(proves (*pf1*, a, *given*, *defns*, *symbols*) $\land$ proves (*pf2*, b, *given*, *defns*, *symbols*))
$\rightarrow$ proves (and-join-proof (a, b, *pf1*, *pf2*), f-and (a, b), *given*, *defns*, *symbols*)

Event: Enable and-split-proof1; name this event ‘g0287’.

Event: Enable and-split-proof2; name this event ‘g0288’.

Event: Enable and-join-proof; name this event ‘g0289’.

Definition:
iff-split-proof1 (a, b, $pf1$)
= tautconseq-proof (list (f-iff (a, b)), f-implies (a, b), list ($pf1$))

```
;;;;;;;Derive "(A -> B)" from "(A <-> B)".
```

Theorem: iff-split1
proves ($pf1$, f-iff (a, b), $given$, $defns$, $symbols$)
$\rightarrow$ proves (iff-split-proof1 (a, b, $pf1$), f-implies (a, b), $given$, $defns$, $symbols$)

Event: Enable iff-split-proof1; name this event ‘g0290’.

Definition:
iff-split-proof2 (a, b, $pf1$)
= tautconseq-proof (list (f-iff (a, b)), f-implies (b, a), list ($pf1$))

```
;;;;;;;Derive "(B -> A)" from "(A <-> B)".
```

Theorem: iff-split2
proves ($pf1$, f-iff (a, b), $given$, $defns$, $symbols$)
$\rightarrow$ proves (iff-split-proof2 (a, b, $pf1$), f-implies (b, a), $given$, $defns$, $symbols$)

Event: Enable iff-split-proof2; name this event ‘g0291’.

Definition:
iff-join-proof (a, b, $pf1$, $pf2$)
= tautconseq-proof (list (f-implies (a, b), f-implies (b, a)),
f-iff (a, b),
list ($pf1$, $pf2$))

```
;;;;;;;Derive "(A <-> B)" from "(A -> B)" and "(B -> A)".
```

Theorem: iff-join
(proves ($pf1$, f-implies (a, b), $given$, $defns$, $symbols$)
$\wedge$ proves ($pf2$, f-implies (b, a), $given$, $defns$, $symbols$))
$\rightarrow$ proves (iff-join-proof (a, b, $pf1$, $pf2$), f-iff (a, b), $given$, $defns$, $symbols$)

EVENT: Enable iff-join-proof; name this event 'g0292'.

DEFINITION:
contra-pos-proof1 (a, b, *pf1*)
= tautconseq-proof (list (f-implies (a, b)),
f-implies (f-not (b), f-not (a)),
list (*pf1*))

```
;;;;;;;Derive "((not B) -> (not A))" from "(A -> B)".
```

THEOREM: contra-pos1
proves (*pf1*, f-implies (a, b), *given*, *defns*, *symbols*)
$\rightarrow$ proves (contra-pos-proof1 (a, b, *pf1*),
f-implies (f-not (b), f-not (a)),
given,
defns,
symbols)

EVENT: Enable contra-pos-proof1; name this event 'g0293'.

DEFINITION:
contra-pos-proof2 (a, b, *pf*)
= tautconseq-proof (list (f-implies (f-not (b), f-not (a))),
f-implies (a, b),
list (*pf*))

```
;;;;;;;Derive "(A -> B)" from "((not B) -> (not A))".
```

THEOREM: contra-pos2
proves (*pf*, f-implies (f-not (b), f-not (a)), *given*, *defns*, *symbols*)
$\rightarrow$ proves (contra-pos-proof2 (a, b, *pf*), f-implies (a, b), *given*, *defns*, *symbols*)

EVENT: Enable contra-pos-proof2; name this event 'g0294'.

DEFINITION:
iff-commute-proof (a, b, *pf*)
= tautconseq-proof (list (f-iff (a, b)), f-iff (b, a), list (*pf*))

```
;;;;;;;Derive "(B <-> A)" from "(A <-> B)".
```

THEOREM: iff-commute-proof-proves
proves (*pf*, f-iff (a, b), *given*, *defns*, *symbols*)
$\rightarrow$ proves (iff-commute-proof (a, b, *pf*), f-iff (b, a), *given*, *defns*, *symbols*)

EVENT: Enable iff-commute-proof; name this event 'g0884'.

THEOREM: formula-f-iff
formula (f-iff (a, b), *symbols*) = (formula (a, *symbols*) $\wedge$ formula (b, *symbols*))

THEOREM: collect-free-f-iff
(*var* $\in$ collect-free (f-iff (a, b), 0))
= ((*var* $\in$ collect-free (a, 0)) $\vee$ (*var* $\in$ collect-free (b, 0)))

THEOREM: covering-f-iff
covering (f-iff (a, b), *var*, 0)
= append (append (covering (a, *var*, 0), covering (b, *var*, 0)),
append (covering (b, *var*, 0), covering (a, *var*, 0)))

THEOREM: eval-f-iff
eval (f-iff (a, b), *alist*) = (eval (a, *alist*) = eval (b, *alist*))

EVENT: Enable f-iff; name this event 'g0885'.

```
;;;;;;;Now we get into some rules about quantifiers.
```

DEFINITION:
forall-intro-proof(a, b, *var*, *pf*)
= tautconseq-proof (list (f-implies (forsome (*var*, f-not (b)), f-not (a))),
f-implies (a, f-not (forsome (*var*, f-not (b)))),
list (forsome-intro-proof(*var*,
f-not (b),
f-not (a),
contra-pos-proof1 (a, b, *pf*))))

THEOREM: memb-member-numberp
($x \in \mathbf{N}$) $\rightarrow$ (memb (x, y) = ($x \in y$))

```
;;;;;;;Derive "(A -> (forall x B))" from "(A -> B)", provided "x" does not occur free in "A".
```

THEOREM: forall-intro-proof-proves
(proves (*pf*, f-implies (a, b), *given*, *defns*, *symbols*)
$\wedge$ variable (*var*)
$\wedge$ (*var* $\notin$ collect-free (f-not (a), 0))
$\wedge$ (*concl* = f-implies (a, $\forall$ *var* b)))
$\rightarrow$ proves (forall-intro-proof(a, b, *var*, *pf*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable forall-intro-proof; name this event 'g0295'.

DEFINITION:
generalise-proof$(var,\ a,\ pf1)$
$=$ tautconseq-proof (list (f-implies (f-not ($\forall\ var\ a$), $\forall\ var\ a$)),
$\forall\ var\ a$,
list (forall-intro-proof (f-not ($\forall\ var\ a$),
a,
var,
tautconseq-proof (list (a),
f-implies (f-not ($\forall\ var\ a$),
a),
list ($pf1$))))))

THEOREM: collect-free-flg
$(flg \notin \mathbf{N}) \rightarrow$ (collect-free $(exp,\ flg)$ $=$ collect-free $(exp,\ 0)$)

THEOREM: covering-flg
$(flg \notin \mathbf{N}) \rightarrow$ (covering $(exp,\ var,\ flg)$ $=$ covering $(exp,\ var,\ 0)$)

THEOREM: subst-flg
$(flg \notin \mathbf{N}) \rightarrow$ (subst $(exp,\ var,\ term,\ flg)$ $=$ subst $(exp,\ var,\ term,\ 0)$)

THEOREM: variablep-collect-free
($\neg$ variable (var)) $\rightarrow$ ($var \notin$ collect-free $(exp,\ flg)$)

THEOREM: member-del
$x \notin$ del $(x,\ y)$

THEOREM: forsome-collect-free-reduc
$(var \in \mathbf{N}) \rightarrow$ ($var \notin$ collect-free (forsome $(var,\ exp)$, 0))

```
;;;;;;;Derive "(forall x A)" from "A".
```

THEOREM: generalise-proof-proves
(proves $(pf1,\ a,\ given,\ defns,\ symbols)$ $\wedge$ variable (var) $\wedge$ $(concl = \forall\ var\ a)$)
$\rightarrow$ proves (generalise-proof $(var,\ a,\ pf1)$, $concl,\ given,\ defns,\ symbols$)

EVENT: Enable generalise-proof; name this event 'g0300'.

DEFINITION:
subst-rule-proof$(a,\ var,\ term,\ pf1)$
$=$ tautconseq-proof (list (f-implies (f-not (subst $(a,\ var,\ term,\ 0)$),
forsome (var, f-not (a))),
f-not (forsome (var, f-not (a)))),
subst $(a,\ var,\ term,\ 0)$,
list (subst-axiom-proof (f-not (a), var, $term$),
generalise-proof $(var,\ a,\ pf1)$))

```
;;;;;;;Derive "[t/x]A" from "A", where "t" is free for "x" in "A".
```

THEOREM: subst-rule-proof-proves
(variable (var)
 $\wedge$ termp ($term$, 0, $symbols$)
 $\wedge$ proves ($pf1$, a, $given$, $defns$, $symbols$)
 $\wedge$ free-for (a, var, $term$, 0)
 $\wedge$ (a-$prime$ = subst (a, var, $term$, 0)))
$\rightarrow$ proves (subst-rule-proof (a, var, $term$, $pf1$), a-$prime$, $given$, $defns$, $symbols$)

EVENT: Enable subst-rule-proof; name this event ‘g0886’.

```
;;;;;;;"[x/x]A" is equal to "A".
```

THEOREM: subst-var-var
subst (exp, var, var, flg) = exp

DEFINITION: e-intro-proof (var, c) = subst-axiom-proof (c, var, var)

THEOREM: append-nil-intersect
nil-intersect (append (x, y), z)
= (nil-intersect (x, z) $\wedge$ nil-intersect (y, z))

THEOREM: var-covering-var
$var \notin$ covering (exp, var, flg)

THEOREM: nil-intersect-cons
(($\neg$ memb (z, x)) $\wedge$ nil-intersect (x, y)) $\rightarrow$ nil-intersect (x, cons (z, y))

THEOREM: nil-intersect-nil
nil-intersect (x, **nil**)

```
;;;;;;;"(C -> (forsome x C))" is a theorem.
```

THEOREM: e-intro-proof-proves
(formula (c, $symbols$) $\wedge$ variable (var))
$\rightarrow$ proves (e-intro-proof (var, c),
 f-implies (c, forsome (var, c)),
 $given$,
 $defns$,
 $symbols$)

EVENT: Enable e-intro-proof; name this event ‘g0306’.

DEFINITION:
all-elim-proof $(var,\ c)$
$=$ tautconseq-proof (list (f-implies (f-not (c), forsome $(var,$ f-not $(c))))$,
f-implies $(\forall\ var\ c,\ c)$,
list (e-intro-proof $(var,$ f-not $(c))))$

```
;;;;;;;"((forall x C) -> C)" is a theorem.
```

THEOREM: all-elim-proof-proves
(formula $(c,\ symbols) \wedge$ variable $(var))$
$\rightarrow$ proves (all-elim-proof $(var,\ c)$, f-implies $(\forall\ var\ c,\ c)$, $given,\ defns,\ symbols)$

EVENT: Enable all-elim-proof; name this event 'g0307'.

THEOREM: member-del1
$(\neg\ \text{eql}\,(x,\ y)) \rightarrow ((x \in \text{del}\,(y,\ z)) = (x \in z))$

```
;;;;;;;"[t/x]A" is equal to "A", if "x" does not occur free in "A".
```

THEOREM: subst-ident
$(var \notin$ collect-free $(exp,\ flg)) \rightarrow ($subst $(exp,\ var,\ term,\ flg) = exp)$

```
;;;;;;;"y" is free in "[t/x]A" iff either "y" is free in "t" and "x" is free in "A"
;;;;;;;                                   or "y" is not "x" and "y" is free in "A".
```

THEOREM: collect-free-subst
(variable $(var) \wedge$ variable $(var1) \wedge (var \notin$ covering $(exp,\ var1,\ flg)))$
$\rightarrow$ $((var \in$ collect-free (subst $(exp,\ var1,\ term,\ flg),\ flg))$
$=$ $(((var \neq var1) \wedge (var \in$ collect-free $(exp,\ flg)))$
$\vee$ $((var1 \in$ collect-free $(exp,\ flg))$
$\wedge$ $(var \in$ collect-free $(term,\ 0)))))$

```
;;;;;;;If "y" is a 'new' variable, the bound variables governing free occurrences
;;;;;;;of "y" in "[y/x]A" are the ones that govern free occurrences of "x" in "A".
```

THEOREM: free-for-subst
(variable (aux)
$\wedge$ variable (var)
$\wedge$ $(aux \notin$ collect-free $(a,\ flg))$
$\wedge$ $(aux \notin$ covering $(a,\ var,\ flg)))$
$\rightarrow$ (covering (subst $(a,\ var,\ aux,\ flg),\ aux,\ flg) =$ covering $(a,\ var,\ flg))$

THEOREM: not-free-covering
(variable $(var) \wedge (var \notin$ collect-free $(exp,\ flg)))$
$\rightarrow$ (covering $(exp,\ var,\ flg) =$ **nil**)

THEOREM: not-member-delete2
$(x \in \text{del}(y,\ z)) = ((\neg\ \text{eql}(x,\ y)) \land (x \in z))$

THEOREM: collect-free-subst1
(variable (*var1*)
$\land$ variable (*var*)
$\land$ (*var* $\notin$ collect-free (*exp*, *flg*))
$\land$ (*var* $\notin$ collect-free (*term*, 0)))
$\rightarrow$ (*var* $\notin$ collect-free (subst (*exp*, *var1*, *term*, *flg*), *flg*))

THEOREM: collect-free-subst2
(variable (*var*)
$\land$ variable (*var1*)
$\land$ (*var* $\neq$ *var1*)
$\land$ (*var* $\in$ collect-free (*exp*, *flg*)))
$\rightarrow$ (*var* $\in$ collect-free (subst (*exp*, *var1*, *term*, *flg*), *flg*))

THEOREM: subst-covering-unchanged
(variable (*var*)
$\land$ variable (*var1*)
$\land$ (*var* $\neq$ *var1*)
$\land$ (*var* $\notin$ collect-free (*term*, 0)))
$\rightarrow$ (covering (subst (*exp*, *var1*, *term*, *flg*), *var*, *flg*) = covering (*exp*, *var*, *flg*))

DEFINITION:
var-set (*list*, *n*)
= **if** $n \simeq 0$ **then** *list* = **nil**
else variable (car (*list*))
$\land$ (car (*list*) $\notin$ cdr (*list*))
$\land$ var-set (cdr (*list*), $n - 1$) **endif**

THEOREM: nil-intersect-append
nil-intersect (x, append (y, z))
= (nil-intersect (x, y) $\land$ nil-intersect (x, z))

THEOREM: var-set-length
var-set (x, *count*) $\rightarrow$ var-set (x, length (x))

THEOREM: nil-intersect-subst
(variable (*var*)
$\land$ var-set (x, length (x))
$\land$ nil-intersect (x, collect-free (*exp*, *flg*))
$\land$ nil-intersect (x, collect-free (*term*, 0)))
$\rightarrow$ nil-intersect (x, collect-free (subst (*exp*, *var*, *term*, *flg*), *flg*))

THEOREM: nil-intersect-cons2
$\text{nil-intersect}(x, \text{cons}(y, z)) = ((\neg \text{memb}(y, x)) \land \text{nil-intersect}(x, z))$

```
;;;;;;;PAR-SUBST is the first definition of simultaneous substitution.
;;;;;;;It returns "[t1/x1, t2/x2,...,tn/xn]A".  It employs a list of
;;;;;;;new variables "yi" and evaluates "[t1/y1]...[tn/yn][yn/xn]...[y1/x1]A".
;;;;;;;This is in order to avoid substituting for an occurrence of an "xi"
;;;;;;;in some "tj".  This definition of simultaneous substitution while
;;;;;;;clumsy, turns out to be convenient for the proof of instantiation rule,
;;;;;;;which by necessity must be derived from the substitution rule above.
```

DEFINITION:
par-subst (*exp*, *vars*, *terms*, *auxvars*)
= **if** listp (*vars*) ∧ listp (*terms*) ∧ listp (*auxvars*)
then subst (par-subst (subst (*exp*, car (*vars*), car (*auxvars*), 0),
cdr (*vars*),
cdr (*terms*),
cdr (*auxvars*)),
car (*auxvars*),
car (*terms*),
0)
else *exp* **endif**

```
;;;;;;;The unsubstituted free variables in EXP remain free after PAR-SUBST.
```

THEOREM: par-subst-collect-free
(variable (*aux*)
∧ (*aux* ∈ collect-free (*exp*, 0))
∧ var-set (*auxlist*, length (*varlist*))
∧ var-set (*varlist*, length (*varlist*))
∧ (*aux* ∉ *varlist*)
∧ (*aux* ∉ *auxlist*))
→ (*aux* ∈ collect-free (par-subst (*exp*, *varlist*, *termlist*, *auxlist*), 0))

```
;;;;;;;The COVERING of an unsubsubstituted free variables in EXP is unchanged
;;;;;;;after PAR-SUBST, provided it does not occur free in TERMLIST.
```

THEOREM: par-subst-covering
((length (*termlist*) = length (*varlist*))
∧ (*aux* ∉ collect-free (*termlist*, 1))
∧ (*aux* ∉ *auxlist*)
∧ (*aux* ∉ *varlist*)
∧ var-set (*varlist*, length (*varlist*))
∧ var-set (*auxlist*, length (*varlist*))
∧ variable (*aux*))
→ (covering (par-subst (*exp*, *varlist*, *termlist*, *auxlist*), *aux*, 0)
= covering (*exp*, *aux*, 0))

```
;;;;;;;FREE-FOR-LIST checks if each term in TERMLIST is free for the corresponding
;;;;;;;variable in VARLIST, in A.
```

Definition:
free-for-list (a, *varlist*, *termlist*)
$=$ **if** listp (*varlist*) $\land$ listp (*termlist*)
then free-for (a, car (*varlist*), car (*termlist*), 0)
$\land$ free-for-list (a, cdr (*varlist*), cdr (*termlist*))
else t endif

```
;;;;;;;The variables covering AUX in (SUBST EXP VAR AUX 0) are the ones covering VAR
;;;;;;;in EXP, and that is unaffected by PAR-SUBST, when the hypotheses below hold.
```

Theorem: covering-par-subst
(variable (*var*)
$\land$ variable (*aux*)
$\land$ (*aux* $\notin$ collect-free (*termlist*, 1))
$\land$ (*aux* $\notin$ collect-free (*exp*, 0))
$\land$ (*aux* $\notin$ covering (*exp*, *var*, 0))
$\land$ (*aux* $\notin$ *auxlist*)
$\land$ (*aux* $\notin$ *varlist*)
$\land$ var-set (*varlist*, length (*varlist*))
$\land$ var-set (*auxlist*, length (*varlist*))
$\land$ (length (*termlist*) $=$ length (*varlist*)))
$\rightarrow$ (covering (par-subst (subst (*exp*, *var*, *aux*, 0), *varlist*, *termlist*, *auxlist*),
aux,
0)
$=$ covering (*exp*, *var*, 0))

```
;;;;;;;INST-CONDITIONS lays down the various conditions for PAR-SUBST to be a proper
;;;;;;;simultaneous substitution: (1) VARS and AUXVARS must be sets of variables,
;;;;;;;(2) TERMS must be a list of terms, (3) AUXVARS must be a list of new variables,
;;;;;;;(4) the AUXVARS must be free the respective VARS in A, and (5) the TERMS must
;;;;;;;be free for the respective VARS in A.
```

Definition:
inst-conditions (a, *vars*, *terms*, *auxvars*, *symbols*)
$=$ (var-set (*vars*, length (*vars*))
$\land$ var-set (*auxvars*, length (*vars*))
$\land$ termp (*terms*, 1, *symbols*)
$\land$ (length (*terms*) $=$ length (*vars*))
$\land$ nil-intersect (*auxvars*, collect-free (*terms*, 1))
$\land$ nil-intersect (*auxvars*, collect-free (a, 0))
$\land$ nil-intersect (*auxvars*, *vars*)
$\land$ free-for-list (a, *vars*, *auxvars*)
$\land$ free-for-list (a, *vars*, *terms*))

```
;;;;;;;FREE-FOR-LIST is unaffected by an irrelevant substitution into A.
```

THEOREM: free-for-list-elim
 (variable (*var*)
 $\wedge$ variable (*aux*)
 $\wedge$ var-set (*vars*, length (*vars*))
 $\wedge$ ($var \notin vars$)
 $\wedge$ ($aux \notin vars$)
 $\wedge$ free-for-list (*a*, *vars*, *terms*))
 $\rightarrow$ free-for-list (subst (*a*, *var*, *aux*, 0), *vars*, *terms*)

```
;;;;;;;PAR-SUBST-PROOF constructs the proof of the instance of A, given the proof of A.
```

DEFINITION:
par-subst-proof (*a*, *vars*, *terms*, *auxvars*, *pf1*)
= **if** listp (*vars*) $\wedge$ listp (*terms*) $\wedge$ listp (*auxvars*)
 then subst-rule-proof (par-subst (subst (*a*, car (*vars*), car (*auxvars*), 0),
 cdr (*vars*),
 cdr (*terms*),
 cdr (*auxvars*)),
 car (*auxvars*),
 car (*terms*),
 par-subst-proof (subst (*a*,
 car (*vars*),
 car (*auxvars*),
 0),
 cdr (*vars*),
 cdr (*terms*),
 cdr (*auxvars*),
 subst-rule-proof (*a*,
 car (*vars*),
 car (*auxvars*),
 pf1)))
 else *pf1* **endif**

```
;;;;;;;The lemma below is the first form of the Instantiation Rule.
;;;;;;;It asserts that if A is provable, so is any instance of it as
;;;;;;;constructed by PAR-SUBST.
```

THEOREM: par-subst-proof-proves
 (inst-conditions (*a*, *vars*, *terms*, *auxvars*, *symbols*)
 $\wedge$ proves (*pf1*, *a*, *given*, *defns*, *symbols*))
 $\rightarrow$ proves (par-subst-proof (*a*, *vars*, *terms*, *auxvars*, *pf1*),
 par-subst (*a*, *vars*, *terms*, *auxvars*),
 given,

defns,
symbols)

```
;;;;;;;Since PAR-SUBST is not a very efficient way to carry out simultaneous
;;;;;;;substitution, we will prove it to be equivalent to a different
;;;;;;;form of simultaneous substitution.
```

DEFINITION:
del-pairs (*var*, *vars*, *terms*)
= **if** listp (*vars*) $\wedge$ listp (*terms*)
then if eql (car (*vars*), *var*) **then** del-pairs (*var*, cdr (*vars*), cdr (*terms*))
else cons (car (*terms*), del-pairs (*var*, cdr (*vars*), cdr (*terms*))) **endif**
else *terms* **endif**

DEFINITION:
getsub (*var*, *vars*, *terms*)
= **if** listp (*vars*) $\wedge$ listp (*terms*)
then if eql (car (*vars*), *var*) **then** car (*terms*)
else getsub (*var*, cdr (*vars*), cdr (*terms*)) **endif**
else *var* **endif**

THEOREM: del-pairs-getsub
variable (*var*)
$\rightarrow$ (getsub (*var*, del (*var*, *vars*), del-pairs (*var*, *vars*, *terms*)) = *var*)

```
;;;;;;;PARALLEL-SUBST below is the more efficient form of simultaneous substitution.
;;;;;;;It recurses down EXP and replaces any free occurrence of a variable in
;;;;;;;VARS by the corresponding term in TERMS.
```

DEFINITION:
parallel-subst (*exp*, *vars*, *terms*, *flg*)
= **if** *flg* $\simeq$ 0
then if variable (*exp*) **then** getsub (*exp*, *vars*, *terms*)
elseif f-notp (*exp*)
then f-not (parallel-subst (arg (*exp*), *vars*, *terms*, 0))
elseif f-orp (*exp*)
then f-or (parallel-subst (arg1 (*exp*), *vars*, *terms*, 0),
parallel-subst (arg2 (*exp*), *vars*, *terms*, 0))
elseif forsomep (*exp*)
then forsome (bind (*exp*),
parallel-subst (body (*exp*),
del (bind (*exp*), *vars*),
del-pairs (bind (*exp*), *vars*, *terms*),
0))
elseif listp (*exp*)

then cons (car (exp), parallel-subst (cdr (exp), $vars$, $terms$, `1`))
else exp **endif**
elseif listp (exp)
then cons (parallel-subst (car (exp), $vars$, $terms$, `0`),
parallel-subst (cdr (exp), $vars$, $terms$, `1`))
else exp **endif**

THEOREM: getsub-not-memb
$(\neg$ memb $(var,\ varlist)) \rightarrow ($getsub$\,(var,\ varlist,\ terms) = var)$

THEOREM: eql-trans
$($eql$\,(x,\ y) \wedge$ eql$\,(y,\ z)) \rightarrow$ eql$\,(x,\ z)$

THEOREM: getsub-del-pairs
$($getsub$\,(var,\ vars,\ terms) = var)$
$\rightarrow \quad ($getsub$\,(var,$ del$\,(var1,\ vars),$ del-pairs$\,(var1,\ vars,\ terms)) = var)$

EVENT: Enable eql-trans; name this event ‘g0636’.

```
;;;;;;;The lemma below asserts that the effect of a single substitution
;;;;;;;followed by a simultaneous substitution, can be got by extending the
;;;;;;;simultaneous substitution.
```

THEOREM: newvar-unchanged-by-parallel-subst
$(($getsub$\,(newvar,\ vars,\ terms) = newvar)$
$\wedge \quad$ variable$\,(var)$
$\wedge \quad$ variable$\,(newvar))$
$\rightarrow \quad ($parallel-subst$\,($subst$\,(exp,\ var,\ newvar,\ flg),\ vars,\ terms,\ flg)$
$= \quad$ parallel-subst$\,(exp,$ cons$\,(var,\ vars),$ cons$\,(newvar,\ terms),\ flg))$

THEOREM: not-free-in-terms-step
$($variable$\,(var)$
$\wedge \quad$ variable$\,(newvar)$
$\wedge \quad (newvar \neq var)$
$\wedge \quad (newvar \not\in$ collect-free$\,(terms,\ 1)))$
$\rightarrow \quad (newvar \not\in$ collect-free$\,($getsub$\,(var,\ vars,\ terms),\ 0))$

THEOREM: collect-terms-del-pairs
$(newvar \not\in$ collect-free$\,(terms,\ 1))$
$\rightarrow \quad (newvar \not\in$ collect-free$\,($del-pairs$\,(var,\ vars,\ terms),\ 1))$

THEOREM: collect-free-parallel-subst
$($variable$\,(newvar)$
$\wedge \quad (newvar \not\in$ collect-free$\,(exp,\ flg))$
$\wedge \quad (newvar \not\in$ collect-free$\,(terms,\ 1)))$
$\rightarrow \quad (newvar \not\in$ collect-free$\,($parallel-subst$\,(exp,\ vars,\ terms,\ flg),\ flg))$

THEOREM: parallel-subst-not-free-var
(variable (var) $\land$ ($var \notin$ collect-free (exp, flg)))
$\rightarrow$ (parallel-subst (exp, cons (var, $vars$), cons ($term$, $terms$), flg)
= parallel-subst (exp, $vars$, $terms$, flg))

THEOREM: subst-parallel-subst
(variable ($newvar$)
$\land$ ($newvar \notin$ collect-free (exp, flg))
$\land$ ($newvar \notin$ collect-free ($terms$, 1)))
$\rightarrow$ (subst (parallel-subst (exp, $vars$, $terms$, flg), $newvar$, $newterm$, flg)
= parallel-subst (exp, $vars$, $terms$, flg))

```
;;;;;;;A PARALLEL-SUBST where VAR is replaced by NEWVAR, followed by a single substitution
;;;;;;;where NEWVAR is replaced by NEWTERM, is equivalent to the same PARALLEL-SUBST
;;;;;;;with VAR replaced by NEWTERM instead.
```

THEOREM: parallel-subst-subst
(variable ($newvar$)
$\land$ variable (var)
$\land$ ($newvar \notin$ collect-free (exp, flg))
$\land$ ($newvar \notin$ collect-free ($terms$, 1))
$\land$ ($newvar \notin$ covering (exp, var, flg)))
$\rightarrow$ (subst (parallel-subst (exp, cons (var, $vars$), cons ($newvar$, $terms$), flg),
$newvar$,
$newterm$,
flg)
= parallel-subst (exp, cons (var, $vars$), cons ($newterm$, $terms$), flg))

```
;;;;;;;A null parallel-substitution leaved EXP unchanged.
```

THEOREM: parallel-subst-nil
(($vars \simeq$ **nil**) $\lor$ ($terms \simeq$ **nil**))
$\rightarrow$ (parallel-subst (exp, $vars$, $terms$, flg) = exp)

```
;;;;;;;Finally, the desired lemma that PAR-SUBST is equivalent to PARALLEL-SUBST,
;;;;;;;if the hypothesis conditions hold.
```

THEOREM: parallel-subst-same-as-par-subst
(var-set ($vars$, length ($vars$))
$\land$ var-set ($auxvars$, length ($auxvars$))
$\land$ (length ($auxvars$) = length ($vars$))
$\land$ nil-intersect ($auxvars$, $vars$)
$\land$ free-for-list (exp, $vars$, $auxvars$)
$\land$ nil-intersect ($auxvars$, collect-free ($terms$, 1))
$\land$ nil-intersect ($auxvars$, collect-free (exp, 0)))
$\rightarrow$ (par-subst (exp, $vars$, $terms$, $auxvars$)
= parallel-subst (exp, $vars$, $terms$, 0))

```
;;;;;;;We still have to show that list of new variables AUXVARS can be constructed.
;;;;;;;That is done in the sequence of events (definitions and lemmas) below.
```

DEFINITION:
make-newvars (*index*, *number*)
= **if** *number* $\simeq$ 0 **then nil**
else cons (1 + *index*, make-newvars (1 + *index*, *number* − 1)) **endif**

DEFINITION:
find-hi-index (*varlist*)
= **if** listp (*varlist*)
then if car (*varlist*) < find-hi-index (cdr (*varlist*))
then find-hi-index (cdr (*varlist*))
else fix (car (*varlist*)) **endif**
else 0 **endif**

DEFINITION:
make-auxvars (*varlist*, *number*)
= make-newvars (find-hi-index (*varlist*), *number*)

THEOREM: not-member-varlist
(find-hi-index (*varlist*) < *var*) → (*var* $\notin$ *varlist*)

THEOREM: newvars-make-newvars
(find-hi-index (*varlist*) ≤ *index*)
→ nil-intersect (make-newvars (*index*, *number*), *varlist*)

THEOREM: var-set-make-newvars-step
(*var* ≤ *index*) → (*var* $\notin$ make-newvars (*index*, *number*))

THEOREM: append-var-list
(var-list (x, length (x)) ∧ var-list (y, length (y)))
→ var-list (append (x, y), length (append (x, y)))

DEFINITION:
covering-list (*exp*, *vars*, *flg*)
= **if** listp (*vars*)
then append (covering (*exp*, car (*vars*), *flg*),
covering-list (*exp*, cdr (*vars*), *flg*))
else nil endif

THEOREM: free-for-list-newvars-covering-list
(var-set (*newvars*, length (*newvars*))
∧ nil-intersect (*newvars*, covering-list (*exp*, *vars*, 0)))
→ free-for-list (*exp*, *vars*, *newvars*)

THEOREM: var-list-var-set
var-set (*vars*, *length*) $\rightarrow$ var-list (*vars*, *length*)

THEOREM: find-hi-index-append
find-hi-index (append (*list1*, *list2*))
= **if** find-hi-index (*list1*) < find-hi-index (*list2*)
then find-hi-index (*list2*)
else find-hi-index (*list1*) **endif**

DEFINITION:
collect-vars (*exp*, *vars*, *terms*)
= append (*vars*,
append (collect-free (*terms*, 1),
append (collect-free (*exp*, 0), covering-list (*exp*, *vars*, 0))))

THEOREM: nil-intersect-newvars-vars
nil-intersect (make-newvars (find-hi-index (collect-vars (*exp*, *vars*, *terms*)),
length),
vars)

THEOREM: nil-intersect-newvars-terms
nil-intersect (make-newvars (find-hi-index (collect-vars (*exp*, *vars*, *terms*)),
length),
collect-free (*terms*, 1))

THEOREM: nil-intersect-newvars-exp
nil-intersect (make-newvars (find-hi-index (collect-vars (*exp*, *vars*, *terms*)),
length),
collect-free (*exp*, 0))

THEOREM: var-set-make-newvars-again
var-set (make-newvars (*index*, *number*), *number*)

THEOREM: length-make-newvars
length (make-newvars (x, n)) = fix (n)

THEOREM: free-for-list-newvars-exp
free-for-list (*exp*,
vars,
make-newvars (find-hi-index (collect-vars (*exp*, *vars*, *terms*)),
length))

EVENT: Enable make-newvars; name this event ‘g0347’.

DEFINITION:
instance-proof (*a*, *vars*, *terms*, *pf1*)
= par-subst-proof (*a*,
vars,
terms,
make-auxvars (collect-vars (*a*, *vars*, *terms*), length (*vars*)),
pf1)

THEOREM: par-subst-proof-proves1
((*concl* = par-subst (*a*, *vars*, *terms*, *auxvars*))
∧ inst-conditions (*a*, *vars*, *terms*, *auxvars*, *symbols*)
∧ proves (*pf1*, *a*, *given*, *defns*, *symbols*))
→ proves (par-subst-proof (*a*, *vars*, *terms*, *auxvars*, *pf1*),
concl,
given,
defns,
symbols)

EVENT: Enable par-subst-proof; name this event 'g0346'.

EVENT: Enable find-hi-index; name this event 'g0350'.

EVENT: Enable find-hi-index-append; name this event 'g0351'.

```
;;;;;;;The Instantiation Rule can then be phrased as shown below, in terms
;;;;;;;of PARALLEL-SUBST instead of PAR-SUBST.
```

THEOREM: instance-proof-proves
((*a-prime* = parallel-subst (*a*, *vars*, *terms*, 0))
∧ var-set (*vars*, length (*vars*))
∧ termp (*terms*, 1, *symbols*)
∧ (length (*terms*) = length (*vars*))
∧ free-for-list (*a*, *vars*, *terms*)
∧ proves (*pf1*, *a*, *given*, *defns*, *symbols*))
→ proves (instance-proof (*a*, *vars*, *terms*, *pf1*), *a-prime*, *given*, *defns*, *symbols*)

DEFINITION:
list-exis-quantify (*vars*, *a*)
= **if** listp (*vars*)
then forsome (car (*vars*), list-exis-quantify (cdr (*vars*), *a*))
else *a* **endif**

THEOREM: not-member-del-pairs
(¬ memb (*x*, *y*)) → (del-pairs (*x*, *y*, *z*) = *z*)

THEOREM: not-member-delete-equals
$(\neg\ \text{memb}\,(x,\ y)) \rightarrow (\text{del}\,(x,\ y) = y)$

DEFINITION:
$\text{pairlist}\,(\mathit{vars},\ \mathit{terms})$
$=$ **if** listp ($vars$) $\wedge$ listp ($terms$)
then cons (cons (car ($vars$), car ($terms$)), pairlist (cdr ($vars$), cdr ($terms$)))
else nil endif

THEOREM: parallel-subst-list-exis-quant
var-set ($vars$, length ($vars$))
$\rightarrow$ (parallel-subst (list-exis-quantify ($vars$, a), $vars$, $terms$, 0)
= list-exis-quantify ($vars$, a))

DEFINITION:
list-e-intro-proof ($varlist$, a)
$=$ **if** listp ($varlist$)
then if listp (cdr ($varlist$))
then trans-implies-proof (a,
list-exis-quantify (cdr ($varlist$), a),
list-exis-quantify ($varlist$, a),
list-e-intro-proof (cdr ($varlist$), a),
e-intro-proof (car ($varlist$),
list-exis-quantify (cdr ($varlist$),
a)))
else e-intro-proof (car ($varlist$), a) **endif**
else prop-axiom-proof (a) **endif**

THEOREM: formula-list-exis
(formula (a, $symbols$) $\wedge$ var-set ($vars$, length ($vars$)))
$\rightarrow$ formula (list-exis-quantify ($vars$, a), $symbols$)

EVENT: Enable instance-proof; name this event ‘g0353’.

EVENT: Disable f-implies; name this event ‘g0354’.

THEOREM: a-implies-a
formula (a, $symbols$)
$\rightarrow$ proves (prop-axiom-proof (a), f-implies (a, a), $given$, $defns$, $symbols$)

THEOREM: parallel-subst-f-implies
parallel-subst (f-implies (a, b), $vars$, $terms$, 0)
$=$ f-implies (parallel-subst (a, $vars$, $terms$, 0),
parallel-subst (b, $vars$, $terms$, 0))

THEOREM: subst-f-implies
subst (f-implies (x, y), *var*, *term*, 0)
= f-implies (subst (x, *var*, *term*, 0), subst (y, *var*, *term*, 0))

EVENT: Enable f-implies; name this event 'g0355'.

```
;;;;;;;The iterated forsome-introduction rule:
;;;;;;;   "A -> (forsome x1 (...(forsome xn A)))".
```

THEOREM: list-e-intro-proof-proves1
(var-set (*varlist*, length (*varlist*)) $\wedge$ formula (a, *symbols*))
$\rightarrow$ proves (list-e-intro-proof (*varlist*, a),
f-implies (a, list-exis-quantify (*varlist*, a)),
given,
defns,
symbols)

EVENT: Enable list-e-intro-proof; name this event 'g0348'.

THEOREM: list-e-intro-proof-proves
(var-set (*varlist*, length (*varlist*))
$\wedge$ formula (a, *symbols*)
$\wedge$ (*a-prime* = list-exis-quantify (*varlist*, a)))
$\rightarrow$ proves (list-e-intro-proof (*varlist*, a),
f-implies (a, *a-prime*),
given,
defns,
symbols)

DEFINITION:
subst-thm-a-proof (a, *vars*, *terms*)
= instance-proof (f-implies (a, list-exis-quantify (*vars*, a)),
vars,
terms,
list-e-intro-proof (*vars*, a))

THEOREM: not-free-list-exis
(var-set (*vars*, length (*vars*)) $\wedge$ (*var* $\in$ *vars*))
$\rightarrow$ (*var* $\notin$ collect-free (list-exis-quantify (*vars*, a), 0))

THEOREM: not-memb-del2
($\neg$ memb (x, y)) $\rightarrow$ ($\neg$ memb (x, del (z, y)))

THEOREM: covering-list-exis
(var-set (*vars*, length (*vars*)) $\wedge$ (*var* $\in$ *vars*) $\wedge$ (*var* $\in$ $\mathbf{N}$))
$\rightarrow$ (covering (list-exis-quantify (*vars*, a), *var*, 0) = **nil**)

THEOREM: free-for-list-step1
(var-set (*vars*, length (*vars*))
 ∧ var-set (*bvars*, length (*bvars*))
 ∧ subset (*vars*, *bvars*)
 ∧ free-for-list (*a*, *vars*, *terms*))
→ free-for-list (f-implies (*a*, list-exis-quantify (*bvars*, *a*)), *vars*, *terms*)

```
;;;;;;;The first substitution theorem:
;;;;;;; "[t1/x1, t2/x2,...,tn/xn]A -> (forsome x1 (...(forsome xn A)))".
```

THEOREM: subst-thm-a
(var-set (*vars*, length (*vars*))
 ∧ termp (*terms*, 1, *symbols*)
 ∧ (length (*terms*) = length (*vars*))
 ∧ free-for-list (*a*, *vars*, *terms*)
 ∧ formula (*a*, *symbols*)
 ∧ (*a-prime* = f-implies (parallel-subst (*a*, *vars*, *terms*, 0),
   list-exis-quantify (*vars*, *a*))))
→ proves (subst-thm-a-proof (*a*, *vars*, *terms*), *a-prime*, *given*, *defns*, *symbols*)

EVENT: Enable subst-thm-a-proof; name this event 'g0356'.

DEFINITION:
list-all-quantify (*vars*, *a*)
= **if** listp (*vars*) **then** ∀ *car*, *vars* list-all-quantify (cdr (*vars*), *a*)
  else *a* **endif**

THEOREM: parallel-subst-list-all-quant
var-set (*vars*, length (*vars*))
→ (parallel-subst (list-all-quantify (*vars*, *a*), *vars*, *terms*, 0)
  = list-all-quantify (*vars*, *a*))

DEFINITION:
list-all-elim-proof (*vars*, *a*)
= **if** listp (*vars*)
  then if cdr (*vars*) ≃ **nil** **then** all-elim-proof (car (*vars*), *a*)
    else trans-implies-proof (list-all-quantify (*vars*, *a*),
      list-all-quantify (cdr (*vars*), *a*),
      a,
      all-elim-proof (car (*vars*),
        list-all-quantify (cdr (*vars*),
          a)),
      list-all-elim-proof (cdr (*vars*), *a*)) **endif**
  else prop-axiom-proof (*a*) **endif**

THEOREM: formula-list-all-quant
(var-set ($vars$, length ($vars$)) $\wedge$ formula (a, $symbols$))
$\rightarrow$ formula (list-all-quantify ($vars$, a), $symbols$)

THEOREM: all-elim-proof-proves-again
(variable (var) $\wedge$ formula (a, $symbols$) $\wedge$ (a-$prime$ = f-implies ($\forall$ var a, a)))
$\rightarrow$ proves (all-elim-proof (var, a), a-$prime$, $given$, $defns$, $symbols$)

THEOREM: list-all-elim-proof-proves1
(var-set ($vars$, length ($vars$)) $\wedge$ formula (a, $symbols$))
$\rightarrow$ proves (list-all-elim-proof ($vars$, a),
f-implies (list-all-quantify ($vars$, a), a),
$given$,
$defns$,
$symbols$)

EVENT: Enable list-all-elim-proof; name this event 'g0357'.

THEOREM: not-free-list-all
(var-set ($vars$, length ($vars$)) $\wedge$ ($var \in vars$))
$\rightarrow$ ($var \notin$ collect-free (list-all-quantify ($vars$, a), 0))

THEOREM: covering-list-all
(var-set ($vars$, length ($vars$)) $\wedge$ ($var \in vars$) $\wedge$ ($var \in \mathbf{N}$))
$\rightarrow$ (covering (list-all-quantify ($vars$, a), var, 0) = **nil**)

THEOREM: free-for-list-step2
(var-set ($vars$, length ($vars$))
$\wedge$ var-set ($bvars$, length ($bvars$))
$\wedge$ subset ($vars$, $bvars$)
$\wedge$ free-for-list (a, $vars$, $terms$))
$\rightarrow$ free-for-list (f-implies (list-all-quantify ($bvars$, a), a), $vars$, $terms$)

DEFINITION:
subst-thm-b-proof (a, $vars$, $terms$)
= instance-proof (f-implies (list-all-quantify ($vars$, a), a),
$vars$,
$terms$,
list-all-elim-proof ($vars$, a))

THEOREM: list-all-elim-proof-proves
(var-set ($vars$, length ($vars$))
$\wedge$ formula (a, $symbols$)
$\wedge$ (a-$prime$ = f-implies (list-all-quantify ($vars$, a), a)))
$\rightarrow$ proves (list-all-elim-proof ($vars$, a), a-$prime$, $given$, $defns$, $symbols$)

```
;;;;;;;The second substitution theorem:
;;;;;;;    "(forall x1 (...(forall xn A))) -> [t1/x1, ..., tn/xn]A".
```

THEOREM: subst-thm-b
(var-set (*vars*, length (*vars*))
 ∧ termp (*terms*, `1`, *symbols*)
 ∧ (length (*terms*) = length (*vars*))
 ∧ formula (*a*, *symbols*)
 ∧ free-for-list (*a*, *vars*, *terms*)
 ∧ (*a-prime* = f-implies (list-all-quantify (*vars*, *a*),
 parallel-subst (*a*, *vars*, *terms*, `0`))))
→ proves (subst-thm-b-proof (*a*, *vars*, *terms*), *a-prime*, *given*, *defns*, *symbols*)

EVENT: Enable subst-thm-b-proof; name this event 'g0358'.

DEFINITION:
distrib-rule-proof-a (*var*, *a*, *b*, *pf1*)
= forsome-intro-proof (*var*,
 a,
 forsome (*var*, *b*),
 trans-implies-proof (*a*,
 b,
 forsome (*var*, *b*),
 pf1,
 e-intro-proof (*var*, *b*)))

```
;;;;;;;The first distribution rule:  Derive "(forsome x A) -> (forsome x B)" from "A -> B".
```

THEOREM: distrib-rule-proof-a-proves
(variable (*var*) ∧ proves (*pf1*, f-implies (*a*, *b*), *given*, *defns*, *symbols*))
→ proves (distrib-rule-proof-a (*var*, *a*, *b*, *pf1*),
 f-implies (forsome (*var*, *a*), forsome (*var*, *b*)),
 given,
 defns,
 symbols)

EVENT: Enable distrib-rule-proof-a; name this event 'g0360'.

DEFINITION:
distrib-rule-proof-b (*var*, *a*, *b*, *pf1*)
= forall-intro-proof (∀ *var* *a*,
 b,
 var,
 trans-implies-proof (∀ *var* *a*,

a,
b,
all-elim-proof$(var,\ a)$,
$pf1))$

```
;;;;;;The second distribution rule:  Derive "(forall x A) -> (forall x B)" from "A -> B".
```

THEOREM: distrib-rule-proof-b-proves
(variable(var)
$\wedge$ proves$(pf,$ f-implies$(a,\ b),\ given,\ defns,\ symbols)$
$\wedge$ $(a\text{-}prime =$ f-implies$(\forall\ var\ a,\ \forall\ var\ b)))$
$\rightarrow$ proves(distrib-rule-proof-b$(var,\ a,\ b,\ pf),\ a\text{-}prime,\ given,\ defns,\ symbols)$

EVENT: Enable distrib-rule-proof-b; name this event 'g0361'.

THEOREM: formula-forall
formula$(\forall\ var\ exp,\ symbols) =$ formula$(exp,\ symbols)$

THEOREM: collect-free-forsome
variable(var)
$\rightarrow$ (collect-free(forsome$(var,\ exp), 0) =$ del$(var,$ collect-free$(exp, 0)))$

THEOREM: collect-free-forall
variable(var)
$\rightarrow$ (collect-free$(\forall\ var\ exp, 0) =$ del$(var,$ collect-free$(exp, 0)))$

THEOREM: covering-forall
variable(var)
$\rightarrow$ (covering$(\forall\ var\ exp,\ x, 0)$
 $=$ **if** memb$(x,$ collect-free$(\forall\ var\ exp, 0))$
 then cons$(var,$ covering$(exp,\ x, 0))$
 else nil endif)

THEOREM: parallel-subst-forall
variable(var)
$\rightarrow$ (parallel-subst$(\forall\ var\ exp,\ vars,\ terms, 0)$
 $=$ $\forall\ var$ parallel-subst$(exp,$
 del$(var,\ vars)$,
 del-pairs$(var,\ vars,\ terms)$,
 $0))$

THEOREM: subst-forall
variable(var)
$\rightarrow$ (subst$(\forall\ var\ exp,\ x,\ term, 0)$
 $=$ **if** $var = x$ **then** $\forall\ var\ exp$
 else $\forall\ var$ subst$(exp,\ x,\ term, 0)$ **endif**)

EVENT: Enable forall; name this event 'g0363'.

DEFINITION:
list-f-iff (lhs, rhs)
= **if** listp (lhs) $\wedge$ listp (rhs)
then cons (f-iff (car (lhs), car (rhs)), list-f-iff (cdr (lhs), cdr (rhs)))
else nil endif

DEFINITION:
f-iff-distrib-rule-a-proof (var, a, $a\text{-}prime$, pf)
= iff-join-proof (forsome (var, a),
forsome (var, $a\text{-}prime$),
distrib-rule-proof-a (var,
a,
$a\text{-}prime$,
iff-split-proof1 (a, $a\text{-}prime$, pf)),
distrib-rule-proof-a (var,
$a\text{-}prime$,
a,
iff-split-proof2 (a, $a\text{-}prime$, pf)))

THEOREM: f-iff-distrib-rule-a-proof-proves
(proves (pf, f-iff (a, $a\text{-}prime$), $given$, $defns$, $symbols$) $\wedge$ variable (var))
$\rightarrow$ proves (f-iff-distrib-rule-a-proof (var, a, $a\text{-}prime$, pf),
f-iff (forsome (var, a), forsome (var, $a\text{-}prime$)),
$given$,
$defns$,
$symbols$)

DEFINITION:
not-iff-proof (a, b, pf)
= tautconseq-proof (list (f-iff (a, b)), f-iff (f-not (a), f-not (b)), list (pf))

THEOREM: not-iff-proof-proves
proves (pf, f-iff (a, b), $given$, $defns$, $symbols$)
$\rightarrow$ proves (not-iff-proof (a, b, pf),
f-iff (f-not (a), f-not (b)),
$given$,
$defns$,
$symbols$)

EVENT: Enable not-iff-proof; name this event 'g0317'.

EVENT: Enable f-iff-distrib-rule-a-proof; name this event 'g0318'.

Definition:
or-iff-proof ($a1$, $a2$, $b1$, $b2$, $pf1$, $pf2$)
= tautconseq-proof (list (f-iff ($a1$, $b1$), f-iff ($a2$, $b2$)),
f-iff (f-or ($a1$, $a2$), f-or ($b1$, $b2$)),
list ($pf1$, $pf2$))

Theorem: or-iff-proof-proves
(proves ($pf1$, f-iff ($a1$, $b1$), *given*, *defns*, *symbols*)
$\wedge$ proves ($pf2$, f-iff ($a2$, $b2$), *given*, *defns*, *symbols*))
$\rightarrow$ proves (or-iff-proof ($a1$, $a2$, $b1$, $b2$, $pf1$, $pf2$),
f-iff (f-or ($a1$, $a2$), f-or ($b1$, $b2$)),
given,
defns,
symbols)

Event: Enable or-iff-proof; name this event 'g0319'.

```
;;;;;;;EQN-FORM checks if A and A-PRIME are identical modulo the equivalences
;;;;;;;given in the list GIVEN.  It is used to prove the equivalence theorem.
```

Definition:
eqn-form (*a-prime*, *a*, *given*)
= **if** *a-prime* = *a* **then t**
elseif f-iff (*a*, *a-prime*) $\in$ *given* **then t**
elseif f-notp (*a*) $\wedge$ f-notp (*a-prime*)
then eqn-form (arg (*a-prime*), arg (*a*), *given*)
elseif f-orp (*a*) $\wedge$ f-orp (*a-prime*)
then eqn-form (arg1 (*a-prime*), arg1 (*a*), *given*)
$\wedge$ eqn-form (arg2 (*a-prime*), arg2 (*a*), *given*)
elseif forsomep (*a*) $\wedge$ forsomep (*a-prime*)
then (bind (*a*) = bind (*a-prime*))
$\wedge$ eqn-form (body (*a-prime*), body (*a*), *given*)
else f endif

Definition:
getpf (x, y, z)
= **if** listp (y) $\wedge$ listp (z)
then if x = car (y) **then** car (z)
else getpf (x, cdr (y), cdr (z)) **endif**
else x **endif**

Theorem: getpf-proves
(proves-list (*pflist*, *list*, *given*, *defns*, *symbols*) $\wedge$ (*concl* $\in$ *list*))
$\rightarrow$ proves (getpf (*concl*, *list*, *pflist*), *concl*, *given*, *defns*, *symbols*)

EVENT: Disable proves; name this event 'g0596'.

THEOREM: eql-trans2
(eql $(x,\ z)$ $\wedge$ eql $(y,\ z)$) $\rightarrow$ eql $(x,\ y)$

THEOREM: eql-trans3
(eql $(x,\ y)$ $\wedge$ eql $(x,\ z)$) $\rightarrow$ eql $(y,\ z)$

THEOREM: proves-eql
(proves (*pf*, *conc1*, *given*, *defns*, *symbols*)
 $\wedge$ eql (*conc1*, *conc2*)
 $\wedge$ formula (*conc2*, *symbols*))
$\rightarrow$ proves (*pf*, *conc2*, *given*, *defns*, *symbols*)

EVENT: Enable proves; name this event 'g0598'.

THEOREM: getsub-proves
(proves-list (*pflist*, *list*, *given*, *defns*, *symbols*)
 $\wedge$ memb (*concl*, *list*)
 $\wedge$ formula (*concl*, *symbols*))
$\rightarrow$ proves (getsub (*concl*, *list*, *pflist*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable proves-eql; name this event 'g0600'.

```
;;;;;;;EQN-FORM-PROOF is the proof-constructor which constructs the proof of the
;;;;;;;equivalence of A and A-PRIME from the proofs of the equivalences in GIVEN.
```

DEFINITION:
eqn-form-proof (*a-prime*, *a*, *given*, *pflist*)
= **if** *a-prime* = *a* **then** tautconseq-proof (**nil**, f-iff (*a*, *a*), **nil**)
elseif f-iff (*a*, *a-prime*) $\in$ *given*
then getpf (f-iff (*a*, *a-prime*), *given*, *pflist*)
elseif f-notp (*a*) $\wedge$ f-notp (*a-prime*)
then not-iff-proof (arg (*a*),
arg (*a-prime*),
eqn-form-proof (arg (*a-prime*), arg (*a*), *given*, *pflist*))
elseif f-orp (*a*) $\wedge$ f-orp (*a-prime*)
then or-iff-proof (arg1 (*a*),
arg2 (*a*),
arg1 (*a-prime*),
arg2 (*a-prime*),
eqn-form-proof (arg1 (*a-prime*), arg1 (*a*), *given*, *pflist*),
eqn-form-proof (arg2 (*a-prime*), arg2 (*a*), *given*, *pflist*))

elseif forsomep (a) $\wedge$ forsomep (*a-prime*)
then f-iff-distrib-rule-a-proof (bind (a),
body (a),
body (*a-prime*),
eqn-form-proof (body (*a-prime*),
body (a),
given,
pflist))
else nil endif

```
;;;;;;;The equivalence theorem:  If A and A-PRIME are identical modulo the
;;;;;;;proven equivalences in GIVEN (whose proofs are in PFLIST), then
;;;;;;; "A <-> A-PRIME" is provable.
```

THEOREM: eqn-form-proof-proves1
(formula (f-iff (a, *a-prime*), *symbols*)
$\wedge$ eqn-form (*a-prime*, a, *list*)
$\wedge$ proves-list (*pflist*, *list*, *given*, *defns*, *symbols*))
$\rightarrow$ proves (eqn-form-proof (*a-prime*, a, *list*, *pflist*),
f-iff (a, *a-prime*),
given,
defns,
symbols)

EVENT: Enable eqn-form-proof; name this event 'g0338'.

DEFINITION:
elim-link-proof (x, y, z, *pf1*, *pf2*)
= tautconseq-proof (list (f-implies (x, f-implies (y, z)), f-implies (x, y)),
f-implies (x, z),
list (*pf1*, *pf2*))

THEOREM: elim-link-proof-proves
(proves (*pf1*, f-implies (x, f-implies (y, z)), *given*, *defns*, *symbols*)
$\wedge$ proves (*pf2*, f-implies (x, y), *given*, *defns*, *symbols*))
$\rightarrow$ proves (elim-link-proof (x, y, z, *pf1*, *pf2*),
f-implies (x, z),
given,
defns,
symbols)

EVENT: Enable elim-link-proof; name this event 'g0499'.

DEFINITION:

```
elim-args-proof (hyp, args, exp, pflist, pf)
=   if listp (args)
    then elim-args-proof (hyp,
                          cdr (args),
                          exp,
                          cdr (pflist),
                          elim-link-proof (hyp,
                                           car (args),
                                           list-implies (cdr (args), exp),
                                           pf,
                                           car (pflist)))
    else pf endif
```

DEFINITION:

```
dist-f-implies (hyp, list)
=   if listp (list)
    then cons (f-implies (hyp, car (list)), dist-f-implies (hyp, cdr (list)))
    else nil endif
```

EVENT: Enable list-implies-reduc; name this event 'g0506'.

THEOREM: elim-args-proof-proves

```
 (formula (exp, symbols)
  ∧   proves-list (pflist, dist-f-implies (hyp, args), given, defns, symbols)
  ∧   proves (pf, f-implies (hyp, list-implies (args, exp)), given, defns, symbols))
 →   proves (elim-args-proof (hyp, args, exp, pflist, pf),
             f-implies (hyp, exp),
             given,
             defns,
             symbols)
```

EVENT: Enable elim-args-proof; name this event 'g0507'.

DEFINITION:

```
symmetry-proof1 (x, y)
=   tautconseq-proof (list (f-equal (x, x),
                          f-implies (f-equal (x, y),
                                     f-implies (f-equal (x, x),
                                                f-implies (f-equal (x, x),
                                                           f-equal (y, x))))),
                     f-implies (f-equal (x, y), f-equal (y, x)),
                     list (ident-axiom-proof (x),
                          equal-axiom2-proof (p (0, 2), list (x, x), list (y, x))))
```

EVENT: Disable f-equal; name this event 'g0365'.

THEOREM: equal-equal-axiom2
formula (f-implies (f-equal (x, y),
 f-implies (f-equal (u, v),
 f-implies (f-equal (x, u), f-equal (y, v)))),
 symbols)
$\rightarrow$ proves (equal-axiom2-proof (p (0, 2), list (x, u), list (y, v)),
 f-implies (f-equal (x, y),
 f-implies (f-equal (u, v),
 f-implies (f-equal (x, u), f-equal (y, v)))),
 given,
 defns,
 symbols)

EVENT: Enable f-equal; name this event 'g0368'.

```
;;;;;;;The symmetry theorem: "(x = y) -> (y = x)"
;;;;;;;The next few events lead up to the proof of the equality theorem.
```

THEOREM: symmetry-theorem1
formula (f-equal (x, y), *symbols*)
$\rightarrow$ proves (symmetry-proof1 (x, y),
 f-implies (f-equal (x, y), f-equal (y, x)),
 given,
 defns,
 symbols)

EVENT: Enable symmetry-proof1; name this event 'g0372'.

DEFINITION:
list-f-and (*list*)
= **if** listp (*list*)
 then if listp (cdr (*list*)) **then** f-and (car (*list*), list-f-and (cdr (*list*)))
 else car (*list*) **endif**
 else nil endif

THEOREM: formula-list-f-and
(form-list (*list*, *symbols*) $\wedge$ listp (*list*))
$\rightarrow$ formula (list-f-and (*list*), *symbols*)

DEFINITION:
list-f-and-member-proof (x, *list*)

$=$ **if** listp (*list*)
then if listp (cdr (*list*))
then if $x =$ car (*list*)
then tautconseq-proof (**nil**,
f-implies (f-and (x,
list-f-and (cdr (*list*))),
x),
nil)
else tautconseq-proof (list (f-implies (list-f-and (cdr (*list*)),
x)),
f-implies (list-f-and (*list*), x),
list (list-f-and-member-proof (x,
cdr (*list*)))) **endif**
else tautconseq-proof (**nil**, f-implies (x, x), **nil**) **endif**
else nil endif

THEOREM: list-f-and-member-proof-proves
(form-list (*list*, *symbols*) $\land$ ($x \in$ *list*) $\land$ formula (x, *symbols*))
$\rightarrow$ proves (list-f-and-member-proof (x, *list*),
f-implies (list-f-and (*list*), x),
given,
defns,
symbols)

EVENT: Enable list-f-and; name this event ‘g0508’.

DEFINITION:
add-hyp-proof (*hyp*, *exp*, *pf*)
$=$ tautconseq-proof (list (*exp*), f-implies (*hyp*, *exp*), list (*pf*))

THEOREM: add-hyp-proof-proves
(formula (*hyp*, *symbols*) $\land$ proves (*pf*, *exp*, *given*, *defns*, *symbols*))
$\rightarrow$ proves (add-hyp-proof (*hyp*, *exp*, *pf*),
f-implies (*hyp*, *exp*),
given,
defns,
symbols)

EVENT: Enable add-hyp-proof; name this event ‘g0509’.

DEFINITION:
list-f-equal (*lista*, *listb*)
$=$ **if** listp (*lista*) $\land$ listp (*listb*)
then cons (f-equal (car (*lista*), car (*listb*)),

list-f-equal (cdr ($lista$), cdr ($listb$)))
else nil endif

Definition:
flg-f-implies-equal (hyp, $terma$, $termb$, flg)
= **if** $flg \simeq$ `0` **then** f-implies (hyp, f-equal ($terma$, $termb$))
else dist-f-implies (hyp, list-f-equal ($terma$, $termb$)) **endif**

Definition:
flg-proves (pf, exp, $given$, $defns$, $symbols$, flg)
= **if** $flg \simeq$ `0` **then** proves (pf, exp, $given$, $defns$, $symbols$)
else proves-list (pf, exp, $given$, $defns$, $symbols$) **endif**

Definition:
term-list ($terms$, $symbols$)
= **if** listp ($terms$)
then termp (car ($terms$), `0`, $symbols$) $\wedge$ term-list (cdr ($terms$), $symbols$)
else t endif

Definition:
flg-termp ($term$, flg, $symbols$)
= **if** $flg \simeq$ `0` **then** termp ($term$, `0`, $symbols$)
else term-list ($term$, $symbols$) **endif**

Theorem: termp-term-list
termp (exp, `1`, $symbols$) $\rightarrow$ term-list (exp, $symbols$)

Theorem: pairequals-reduc
pairequals ($vars1$, $vars2$, exp) = list-implies (list-f-equal ($vars1$, $vars2$), exp)

Theorem: formula-list-implies1
formula (list-implies ($list$, exp), $symbols$)
= (form-list ($list$, $symbols$) $\wedge$ formula (exp, $symbols$))

Theorem: form-list-f-equal
(term-list ($terms1$, $symbols$) $\wedge$ term-list ($terms2$, $symbols$))
$\rightarrow$ form-list (list-f-equal ($terms1$, $terms2$), $symbols$)

Definition:
symmetry-proof (a, b)
= iff-join-proof (f-equal (a, b),
f-equal (b, a),
symmetry-proof1 (a, b),
symmetry-proof1 (b, a))

THEOREM: symmetry-theorem
formula (f-equal (a, b), *symbols*)
$\rightarrow$ proves (symmetry-proof (a, b),
f-iff (f-equal (a, b), f-equal (b, a)),
given,
defns,
symbols)

EVENT: Enable symmetry-proof; name this event ‘g0373’.

DEFINITION:
commute-equal-proof (x, y, pf)
= detach-proof (f-equal (x, y), f-equal (y, x), pf, symmetry-proof1 (x, y))

THEOREM: commute-equal-proof-proves
(proves (pf, f-equal (x, y), *given*, *defns*, *symbols*) $\land$ (*concl* = f-equal (y, x)))
$\rightarrow$ proves (commute-equal-proof (x, y, pf), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable commute-equal-proof; name this event ‘g0370’.

DEFINITION:
list-commute-equal-proof (*lista*, *listb*, *pflist*)
= **if** listp (*lista*) $\land$ listp (*listb*)
then cons (commute-equal-proof (car (*lista*), car (*listb*), car (*pflist*)),
list-commute-equal-proof (cdr (*lista*), cdr (*listb*), cdr (*pflist*)))
else nil endif

THEOREM: list-commute-equal-proof-proves
proves-list (*pflist*, list-f-equal (*lista*, *listb*), *given*, *defns*, *symbols*)
$\rightarrow$ proves-list (list-commute-equal-proof (*lista*, *listb*, *pflist*),
list-f-equal (*listb*, *lista*),
given,
defns,
symbols)

EVENT: Enable list-commute-equal-proof; name this event ‘g0371’.

DEFINITION:
forall-over-implies-proof (x, a, b)
= tautconseq-proof (list (f-implies (forsome (x, a),
f-implies ($\forall$ x f-implies (a, b),
forsome (x, b)))),
f-implies ($\forall$ x f-implies (a, b),

```
f-implies (forsome (x, a), forsome (x, b))),
list (forsome-intro-proof (x,
                          a,
                          f-implies (∀ x f-implies (a, b),
                                     forsome (x, b)),
                          tautconseq-proof (list (f-implies (∀ x f-implies (a,
                                                                            b),
                                                             f-implies (a,
                                                                        b)),
                                                  f-implies (b,
                                                             forsome (x,
                                                                      b))),
                                            f-implies (a,
                                                       f-implies (∀ x f-implies (a,
                                                                                 b),
                                                                  forsome (x,
                                                                           b))),
                                            list (all-elim-proof (x,
                                                                  f-implies (a,
                                                                             b)),
                                                  e-intro-proof (x,
                                                                 b))))))
```

THEOREM: forall-over-implies-proof-proves

```
 (variable (x) ∧ formula (a, symbols) ∧ formula (b, symbols))
 →    proves (forall-over-implies-proof (x, a, b),
              f-implies (∀ x f-implies (a, b),
                         f-implies (forsome (x, a), forsome (x, b))),
              given,
              defns,
              symbols)
```

EVENT: Enable forall-over-implies-proof; name this event ‘g0910’.

DEFINITION:

```
forall-over-iff-proof (x, a, b)
=    tautconseq-proof (list (f-implies (forsome (x, a),
                                       f-implies (∀ x f-iff (a, b), forsome (x, b))),
                            f-implies (forsome (x, b),
                                       f-implies (∀ x f-iff (a, b), forsome (x, a)))),
                      f-implies (∀ x f-iff (a, b),
                                 f-iff (forsome (x, a), forsome (x, b))),
                      list (forsome-intro-proof (x,
                                                 a,
```

```
                    f-implies (∀ x f-iff (a, b),
                               forsome (x, b)),
                    tautconseq-proof (list (f-implies (∀ x f-iff (a,
                                                                 b),
                                                      f-iff (a,
                                                             b)),
                                           f-implies (b,
                                                      forsome (x,
                                                               b))),
                                       f-implies (a,
                                                  f-implies (∀ x f-iff (a,
                                                                       b),
                                                             forsome (x,
                                                                      b))),
                                       list (all-elim-proof (x,
                                                             f-iff (a,
                                                                    b)),
                                             e-intro-proof (x,
                                                            b)))),
forsome-intro-proof (x,
                    b,
                    f-implies (∀ x f-iff (a, b),
                               forsome (x, a)),
                    tautconseq-proof (list (f-implies (∀ x f-iff (a,
                                                                 b),
                                                      f-iff (a,
                                                             b)),
                                           f-implies (a,
                                                      forsome (x,
                                                               a))),
                                       f-implies (b,
                                                  f-implies (∀ x f-iff (a,
                                                                       b),
                                                             forsome (x,
                                                                      a))),
                                       list (all-elim-proof (x,
                                                             f-iff (a,
                                                                    b)),
                                             e-intro-proof (x,
                                                            a))))))
```

THEOREM: forall-over-iff-proof-proves
(variable (x) $\wedge$ formula $(a,\ symbols)$ $\wedge$ formula $(b,\ symbols)$)
$\rightarrow$ proves (forall-over-iff-proof $(x,\ a,\ b)$,

f-implies ($\forall\, x$ f-iff (a, b), f-iff (forsome (x, a), forsome (x, b))),
given,
defns,
symbols)

Event: Enable forall-over-iff-proof; name this event 'g0911'.

Definition:
elim-free-terms (x, *list*)
= **if** listp (*list*)
then if $x \in$ collect-free (car (*list*), 0)
then elim-free-terms (x, cdr (*list*))
else cons (car (*list*), elim-free-terms (x, cdr (*list*))) **endif**
else nil endif

Theorem: subset-elim-free-terms
subset (elim-free-terms (x, *list*), *list*)

Theorem: not-free-elim-free-terms
$x \notin$ collect-free (elim-free-terms (x, *list*), 1)

Event: Disable list-f-and; name this event 'g0912'.

Theorem: collect-free-list-f-and
($x \in$ collect-free (list-f-and (*list*), 0)) = ($x \in$ collect-free (*list*, 1))

Theorem: formula-f-and
formula (f-and (x, y), *symbols*) = (formula (x, *symbols*) $\wedge$ formula (y, *symbols*))

Theorem: covering-f-and
covering (f-and (x, y), *var*, 0) = append (covering (x, *var*, 0), covering (y, *var*, 0))

Theorem: collect-free-f-and
collect-free (f-and (y, z), 0) = append (collect-free (y, 0), collect-free (z, 0))

Theorem: subst-f-and
subst (f-and (x, y), *var*, *term*, 0)
= f-and (subst (x, *var*, *term*, 0), subst (y, *var*, *term*, 0))

Theorem: parallel-subst-f-and
parallel-subst (f-and (x, y), *vars*, *terms*, 0)
= f-and (parallel-subst (x, *vars*, *terms*, 0), parallel-subst (y, *vars*, *terms*, 0))

Theorem: eval-f-and
eval (f-and (x, y), *alist*) = (eval (x, *alist*) $\wedge$ eval (y, *alist*))

EVENT: Enable f-and; name this event 'g0384'.

DEFINITION:
subset-list-f-and-proof (*list1*, *list2*)
= **if** listp (*list2*)
  then if listp (cdr (*list2*))
    then tautconseq-proof (list (f-implies (list-f-and (*list1*),
                                                      car (*list2*)),
                                          f-implies (list-f-and (*list1*),
                                                      list-f-and (cdr (*list2*)))),
                                     f-implies (list-f-and (*list1*),
                                                 list-f-and (*list2*)),
                                     list (list-f-and-member-proof (car (*list2*),
                                                                      list1),
                                          subset-list-f-and-proof (*list1*,
                                                                     cdr (*list2*))))
    else list-f-and-member-proof (car (*list2*), *list1*) **endif**
  else nil endif

THEOREM: subset-list-f-and-proof-proves
(form-list (*list1*, *symbols*)
$\land$ form-list (*list2*, *symbols*)
$\land$ listp (*list1*)
$\land$ listp (*list2*)
$\land$ subset (*list2*, *list1*))
$\rightarrow$ proves (subset-list-f-and-proof (*list1*, *list2*),
         f-implies (list-f-and (*list1*), list-f-and (*list2*)),
         given,
         defns,
         symbols)

DEFINITION:
list-f-implies (x, y)
= **if** listp (x) $\land$ listp (y)
  then cons (f-implies (car (x), car (y)), list-f-implies (cdr (x), cdr (y)))
  else nil endif

DEFINITION:
implies-implies-proof (x, y, u, v, *pf1*, *pf2*)
= tautconseq-proof (list (f-implies (u, x), f-implies (y, v)),
                    f-implies (f-implies (x, y), f-implies (u, v)),
                    list (*pf1*, *pf2*))

THEOREM: implies-implies-proof-proves

(proves ($pf1$, f-implies (u, x), $given$, $defns$, $symbols$)
 $\wedge$ proves ($pf2$, f-implies (y, v), $given$, $defns$, $symbols$))
$\rightarrow$ proves (implies-implies-proof (x, y, u, v, $pf1$, $pf2$),
f-implies (f-implies (x, y), f-implies (u, v)),
$given$,
$defns$,
$symbols$)

EVENT: Enable implies-implies-proof; name this event 'g0876'.

DEFINITION:
list-implies-implies-proof (x, y, z, $pflist$)
= **if** listp (x) $\wedge$ listp (y)
then implies-implies-proof (car (y),
list-implies (cdr (y), z),
car (x),
list-implies (cdr (x), z),
car ($pflist$),
list-implies-implies-proof (cdr (x),
cdr (y),
z,
cdr ($pflist$)))
else tautconseq-proof (**nil**, f-implies (z, z), **nil**) **endif**

THEOREM: list-implies-implies-proof-proves
((length (x) = length (y))
 $\wedge$ formula (z, $symbols$)
 $\wedge$ proves-list ($pflist$, list-f-implies (x, y), $given$, $defns$, $symbols$))
$\rightarrow$ proves (list-implies-implies-proof (x, y, z, $pflist$),
f-implies (list-implies (y, z), list-implies (x, z)),
$given$,
$defns$,
$symbols$)

EVENT: Enable list-implies-implies-proof; name this event 'g0877'.

DEFINITION:
list-symmetry-proof1 (x, y)
= **if** listp (x) $\wedge$ listp (y)
then cons (symmetry-proof1 (car (x), car (y)),
list-symmetry-proof1 (cdr (x), cdr (y)))
else nil endif

THEOREM: list-symmetry-proof1-proves

(term-list (x, *symbols*) $\land$ term-list (y, *symbols*))
$\rightarrow$ proves-list (list-symmetry-proof1 (x, y),
list-f-implies (list-f-equal (x, y), list-f-equal (y, x)),
given,
defns,
symbols)

EVENT: Enable list-symmetry-proof1; name this event ‘g0878’.

DEFINITION:
imp-iff-not-proof (x, y, z, *pf*)
= tautconseq-proof (list (f-implies (x, f-iff (y, z))),
f-implies (x, f-iff (f-not (y), f-not (z))),
list (*pf*))

THEOREM: imp-iff-not-proof-proves
proves (*pf*, f-implies (x, f-iff (y, z)), *given*, *defns*, *symbols*)
$\rightarrow$ proves (imp-iff-not-proof (x, y, z, *pf*),
f-implies (x, f-iff (f-not (y), f-not (z))),
given,
defns,
symbols)

EVENT: Enable imp-iff-not-proof; name this event ‘g0879’.

DEFINITION:
imp-iff-or-proof (x, *y1*, *z1*, *y2*, *z2*, *pf1*, *pf2*)
= tautconseq-proof (list (f-implies (x, f-iff (*y1*, *z1*)),
f-implies (x, f-iff (*y2*, *z2*))),
f-implies (x, f-iff (f-or (*y1*, *y2*), f-or (*z1*, *z2*))),
list (*pf1*, *pf2*))

THEOREM: imp-iff-or-proof-proves
(proves (*pf1*, f-implies (x, f-iff (*y1*, *z1*)), *given*, *defns*, *symbols*)
$\land$ proves (*pf2*, f-implies (x, f-iff (*y2*, *z2*)), *given*, *defns*, *symbols*))
$\rightarrow$ proves (imp-iff-or-proof (x, *y1*, *z1*, *y2*, *z2*, *pf1*, *pf2*),
f-implies (x, f-iff (f-or (*y1*, *y2*), f-or (*z1*, *z2*))),
given,
defns,
symbols)

EVENT: Enable imp-iff-or-proof; name this event ‘g0880’.

DEFINITION:

make-chain (a, b, *list*)
= **if** listp (*list*)
then cons (f-implies (a, car (*list*)), make-chain (car (*list*), b, cdr (*list*)))
else list (f-implies (a, b)) **endif**

DEFINITION:
chain-proof (a, b, *list*, *pflist*)
= **if** listp (*list*)
then trans-implies-proof (a,
car (*list*),
b,
car (*pflist*),
chain-proof (car (*list*),
b,
cdr (*list*),
cdr (*pflist*)))
else car (*pflist*) **endif**

```
;;;;;;;The chain rule below is a frequently used rule of inference.
;;;;;;;It asserts that "A -> B" is provable if there is a chain of provable
;;;;;;;implications leading from "A" to "B".
```

THEOREM: chain-rule
proves-list (*pflist*, make-chain (a, b, *list*), *given*, *defns*, *symbols*)
$\rightarrow$ proves (chain-proof (a, b, *list*, *pflist*),
f-implies (a, b),
given,
defns,
symbols)

EVENT: Enable chain-proof; name this event 'g0396'.

DEFINITION:
make-iff-chain (a, b, *list*)
= **if** listp (*list*)
then cons (f-iff (a, car (*list*)), make-iff-chain (car (*list*), b, cdr (*list*)))
else list (f-iff (a, b)) **endif**

DEFINITION:
chain-iff-proof (a, b, *list*, *pflist*)
= **if** listp (*list*)
then trans-f-iff-proof (a,
car (*list*),
b,

```
                                    car (pflist),
                                    chain-iff-proof (car (list),
                                                     b,
                                                     cdr (list),
                                                     cdr (pflist)))
     else car (pflist) endif
```

```
;;;;;;;The chain rule form "<->".
```

THEOREM: chain-iff-proof-proves

```
 proves-list (pflist, make-iff-chain (a, b, list), given, defns, symbols)
 →    proves (chain-iff-proof (a, b, list, pflist),
              f-iff (a, b),
              given,
              defns,
              symbols)
```

EVENT: Enable chain-iff-proof; name this event ‘g0386’.

DEFINITION:

```
imp-iff-forsome-proof (x, y, z, var, pf)
=    chain-proof (x,
                  f-iff (forsome (var, y), forsome (var, z)),
                  list (∀ var f-iff (y, z)),
                  list (forall-intro-proof (x, f-iff (y, z), var, pf),
                        forall-over-iff-proof (var, y, z)))
```

THEOREM: imp-iff-forsome-proof-proves

```
 (variable (var)
  ∧    (var ∉ collect-free (x, 0))
  ∧    proves (pf, f-implies (x, f-iff (y, z)), given, defns, symbols))
 →    proves (imp-iff-forsome-proof (x, y, z, var, pf),
              f-implies (x, f-iff (forsome (var, y), forsome (var, z))),
              given,
              defns,
              symbols)
```

EVENT: Enable imp-iff-forsome-proof; name this event ‘g0881’.

DEFINITION:

```
dist-f-implies-symmetry-proof (x, list1, list2, pflist)
=    if listp (list1) ∧ listp (list2)
     then cons (chain-proof (x,
                             f-equal (car (list2), car (list1)),
```

list (f-equal (car (*list1*), car (*list2*))),
list (car (*pflist*),
symmetry-proof1 (car (*list1*), car (*list2*)))),
dist-f-implies-symmetry-proof(x,
cdr (*list1*),
cdr (*list2*),
cdr (*pflist*)))
else nil endif

THEOREM: dist-f-implies-symmetry-proof-proves
proves-list (*pflist*,
dist-f-implies (x, list-f-equal (*list1*, *list2*)),
given,
defns,
symbols)
$\rightarrow$ proves-list (dist-f-implies-symmetry-proof(x, *list1*, *list2*, *pflist*),
dist-f-implies (x, list-f-equal (*list2*, *list1*)),
given,
defns,
symbols)

EVENT: Enable dist-f-implies-symmetry-proof; name this event 'g0882'.

DEFINITION:
imp-iff-implies-proof(x, y, z, *pf1*, *pf2*)
= tautconseq-proof (list (f-implies (x, f-implies (y, z)),
f-implies (x, f-implies (z, y))),
f-implies (x, f-iff (y, z)),
list (*pf1*, *pf2*))

THEOREM: imp-iff-implies-proof-proves
(proves (*pf1*, f-implies (x, f-implies (y, z)), *given*, *defns*, *symbols*)
$\wedge$ proves (*pf2*, f-implies (x, f-implies (z, y)), *given*, *defns*, *symbols*))
$\rightarrow$ proves (imp-iff-implies-proof(x, y, z, *pf1*, *pf2*),
f-implies (x, f-iff (y, z)),
given,
defns,
symbols)

EVENT: Enable imp-iff-implies-proof; name this event 'g0883'.

```
;;;;;;;TERM-EQUAL-PROOF is the proof-constructor for the equality theorem
;;;;;;;for terms.
```

DEFINITION:

```
term-equal-proof (terma, termb, list, flg)
=   if flg ≃ 0
    then if terma = termb
         then add-hyp-proof (list-f-and (list),
                             f-equal (terma, terma),
                             ident-axiom-proof (terma))
         elseif f-equal (terma, termb) ∈ list
         then list-f-and-member-proof (f-equal (terma, termb), list)
         elseif f-equal (termb, terma) ∈ list
         then chain-proof (list-f-and (list),
                           f-equal (terma, termb),
                           list (f-equal (termb, terma)),
                           list (list-f-and-member-proof (f-equal (termb,
                                                                   terma),
                                                          list),
                                 symmetry-proof1 (termb, terma)))
         elseif listp (terma) ∧ listp (termb)
         then elim-args-proof (list-f-and (list),
                               list-f-equal (cdr (terma), cdr (termb)),
                               f-equal (terma, termb),
                               term-equal-proof (cdr (terma),
                                                 cdr (termb),
                                                 list,
                                                 1),
                               add-hyp-proof (list-f-and (list),
                                              list-implies (list-f-equal (cdr (terma),
                                                                          cdr (termb)),
                                                            f-equal (terma,
                                                                     termb)),
                                              equal-axiom1-proof (car (terma),
                                                                  cdr (terma),
                                                                  cdr (termb))))
         else nil endif
    elseif listp (terma) ∧ listp (termb)
    then cons (term-equal-proof (car (terma), car (termb), list, 0),
               term-equal-proof (cdr (terma), cdr (termb), list, 1))
    else nil endif
```

```
;;;;;;;TERM-EQUAL checks if TERMA and TERMB are identical modulo the list of
;;;;;;;equalities in LIST.
```

DEFINITION:

```
term-equal (terma, termb, list, flg)
```

$=$ **if** *flg* $\simeq$ `0`
then if *terma* $=$ *termb* **then t**
elseif f-equal (*terma*, *termb*) $\in$ *list* **then t**
elseif f-equal (*termb*, *terma*) $\in$ *list* **then t**
elseif listp (*terma*) $\wedge$ listp (*termb*)
then (car (*terma*) $=$ car (*termb*))
$\wedge$ term-equal (cdr (*terma*), cdr (*termb*), *list*, `1`)
else f endif
elseif listp (*terma*) $\wedge$ listp (*termb*)
then term-equal (car (*terma*), car (*termb*), *list*, `0`)
$\wedge$ term-equal (cdr (*terma*), cdr (*termb*), *list*, `1`)
else *terma* $=$ *termb* **endif**

```
;;;;;;;The equality theorem for terms:  If "a" and "b" are identical
;;;;;;;modulo the equalities "x1 = y1",..., "xn = yn", etc., then
;;;;;;;it is provable that, "((x1=y1) ^ ... ^ (xn=yn))->(a=b)".
```

THEOREM: term-equal-proof-proves
(flg-termp (*terma*, *flg*, *symbols*)
$\wedge$ flg-termp (*termb*, *flg*, *symbols*)
$\wedge$ listp (*list*)
$\wedge$ form-list (*list*, *symbols*)
$\wedge$ term-equal (*terma*, *termb*, *list*, *flg*))
$\rightarrow$ flg-proves (term-equal-proof (*terma*, *termb*, *list*, *flg*),
flg-f-implies-equal (list-f-and (*list*), *terma*, *termb*, *flg*),
given,
defns,
symbols,
flg)

EVENT: Enable term-equal-proof; name this event 'g0364'.

THEOREM: term-equal-proof-proves1
(termp (*terma*, `0`, *symbols*)
$\wedge$ termp (*termb*, `0`, *symbols*)
$\wedge$ listp (*list*)
$\wedge$ form-list (*list*, *symbols*)
$\wedge$ term-equal (*terma*, *termb*, *list*, `0`))
$\rightarrow$ proves (term-equal-proof (*terma*, *termb*, *list*, `0`),
f-implies (list-f-and (*list*), f-equal (*terma*, *termb*)),
given,
defns,
symbols)

THEOREM: term-equal-proof-proves2
(term-list (*terma*, *symbols*)
∧ term-list (*termb*, *symbols*)
∧ listp (*list*)
∧ form-list (*list*, *symbols*)
∧ term-equal (*terma*, *termb*, *list*, 1)
∧ (*concl* = dist-f-implies (list-f-and (*list*), list-f-equal (*terma*, *termb*))))
→ proves-list (term-equal-proof (*terma*, *termb*, *list*, 1),
concl,
given,
defns,
symbols)

EVENT: Enable term-equal-proof-proves; name this event 'g0887'.

```
;;;;;;;FORM-EQUAL is used in the equality theorem for formulas.
;;;;;;;It checks that FORMA and FORMB are identical modulo the equalities in LIST.
```

DEFINITION:
form-equal (*forma*, *formb*, *list*)
= **if** *forma* = *formb* **then t**
elseif f-notp (*forma*) ∧ f-notp (*formb*)
then form-equal (arg (*forma*), arg (*formb*), *list*)
elseif f-orp (*forma*) ∧ f-orp (*formb*)
then form-equal (arg1 (*forma*), arg1 (*formb*), *list*)
∧ form-equal (arg2 (*forma*), arg2 (*formb*), *list*)
elseif forsomep (*forma*) ∧ forsomep (*formb*)
then (bind (*forma*) = bind (*formb*))
∧ listp (elim-free-terms (bind (*forma*), *list*))
∧ form-equal (body (*forma*),
body (*formb*),
elim-free-terms (bind (*forma*), *list*))
elseif listp (*forma*) ∧ listp (*formb*)
then (car (*forma*) = car (*formb*))
∧ term-equal (cdr (*forma*), cdr (*formb*), *list*, 1)
else f endif

```
;;;;;;;FORM-EQUAL-PROOF is the corresponding proof-constructor.
```

DEFINITION:
form-equal-proof (*forma*, *formb*, *list*)
= **if** *forma* = *formb*
then add-hyp-proof (list-f-and (*list*),
f-iff (*forma*, *formb*),

```
                          eqn-form-proof (forma, formb, nil, nil))
elseif f-notp (forma) ∧ f-notp (formb)
then imp-iff-not-proof (list-f-and (list),
                             arg (forma),
                             arg (formb),
                             form-equal-proof (arg (forma), arg (formb), list))
elseif f-orp (forma) ∧ f-orp (formb)
then imp-iff-or-proof (list-f-and (list),
                           arg1 (forma),
                           arg1 (formb),
                           arg2 (forma),
                           arg2 (formb),
                           form-equal-proof (arg1 (forma), arg1 (formb), list),
                           form-equal-proof (arg2 (forma), arg2 (formb), list))
elseif forsomep (forma) ∧ forsomep (formb)
then chain-proof (list-f-and (list),
                      f-iff (forma, formb),
                      list (list-f-and (elim-free-terms (bind (forma), list))),
                      list (subset-list-f-and-proof (list,
                                                      elim-free-terms (bind (forma),
                                                                       list)),
                           imp-iff-forsome-proof (list-f-and (elim-free-terms (bind (forma),
                                                                                   list)),
                                                  body (forma),
                                                  body (formb),
                                                  bind (forma),
                                                  form-equal-proof (body (forma),
                                                                    body (formb),
                                                                    elim-free-terms (bind (forma),
                                                                                     list)))))
else imp-iff-implies-proof (list-f-and (list),
                                forma,
                                formb,
                                elim-args-proof (list-f-and (list),
                                                 list-f-equal (cdr (forma),
                                                               cdr (formb)),
                                                 f-implies (forma,
                                                            formb),
                                                 term-equal-proof (cdr (forma),
                                                                   cdr (formb),
                                                                   list,
                                                                   1),
                                                 add-hyp-proof (list-f-and (list),
                                                                list-implies (list-f-equal (cdr (forma),
```

```
                                                            cdr (formb)),
                                                 f-implies (forma,
                                                            formb)),
                                      equal-axiom2-proof (car (forma),
                                                          cdr (forma),
                                                          cdr (formb)))),
                   elim-args-proof (list-f-and (list),
                                    list-f-equal (cdr (formb),
                                                  cdr (forma)),
                                    f-implies (formb,
                                               forma),
                                    dist-f-implies-symmetry-proof (list-f-and (list),
                                                                   cdr (forma),
                                                                   cdr (formb),
                                                                   term-equal-proof (cdr (forma),
                                                                                     cdr (formb),
                                                                                     list,
                                                                                     1)),
                                    add-hyp-proof (list-f-and (list),
                                                   list-implies (list-f-equal (cdr (formb),
                                                                               cdr (forma)),
                                                                 f-implies (formb,
                                                                            forma)),
                                                   equal-axiom2-proof (car (forma),
                                                                       cdr (formb),
                                                                       cdr (forma))))) endif
```

THEOREM: form-list-elim-free-terms
form-list (*list*, *symbols*) → form-list (elim-free-terms (x, *list*), *symbols*)

```
;;;;;;;The equality theorem for formulas:  If "A" is identical to "B" modulo
;;;;;;;the equalities "x1=y1", ..., "xn=yn", etc., then it is provable that
;;;;;;; "((x1=y1) ^ ... ^ (xn=yn)) -> (A <-> B)".
```

THEOREM: form-equal-proof-proves
```
 (formula (forma, symbols)
  ∧  formula (formb, symbols)
  ∧  form-equal (forma, formb, list)
  ∧  listp (list)
  ∧  form-list (list, symbols))
 →  proves (form-equal-proof (forma, formb, list),
            f-implies (list-f-and (list), f-iff (forma, formb)),
            given,
            defns,
            symbols)
```

EVENT: Enable form-equal-proof; name this event 'g0893'.

THEOREM: get-list-f-equal

$(exp \neq \text{getpf}(exp,\ vars,\ terms))$

$\rightarrow \quad (\text{f-equal}(exp,\ \text{getpf}(exp,\ vars,\ terms)) \in \text{list-f-equal}(vars,\ terms))$

THEOREM: getsub-getpf

$(x \in \mathbf{N}) \rightarrow (\text{getsub}(x,\ y,\ z) = \text{getpf}(x,\ y,\ z))$

THEOREM: term-equal-parallel-subst

$\text{termp}(exp,\ flg,\ symbols)$

$\rightarrow \quad \text{term-equal}(exp,$
$\qquad \text{parallel-subst}(exp,\ vars,\ terms,\ flg),$
$\qquad \text{list-f-equal}(vars,\ terms),$
$\qquad flg)$

THEOREM: delete-delete

$\text{del}(x,\ \text{del}(y,\ z)) = \text{del}(y,\ \text{del}(x,\ z))$

THEOREM: del-pairs-del-pairs

$\text{del-pairs}(x,\ \text{del}(y,\ vars),\ \text{del-pairs}(y,\ vars,\ terms))$

$= \quad \text{del-pairs}(y,\ \text{del}(x,\ vars),\ \text{del-pairs}(x,\ vars,\ terms))$

THEOREM: not-free-del-pairs-parallel-subst

$(x \notin \text{collect-free}(exp,\ flg))$

$\rightarrow \quad (\text{parallel-subst}(exp,\ \text{del}(x,\ vars),\ \text{del-pairs}(x,\ vars,\ terms),\ flg)$
$\qquad = \quad \text{parallel-subst}(exp,\ vars,\ terms,\ flg))$

EVENT: Disable f-equal; name this event 'g0894'.

THEOREM: collect-free-f-equal

$(x \in \text{collect-free}(\text{f-equal}(y,\ z),\ \mathtt{0}))$

$= \quad ((x \in \text{collect-free}(y,\ \mathtt{0})) \lor (x \in \text{collect-free}(z,\ \mathtt{0})))$

EVENT: Enable f-equal; name this event 'g0895'.

THEOREM: del-pairs-elim-free

$(\text{variable}(x)$

$\land \quad \text{var-list}(vars,\ \text{length}(vars))$

$\land \quad (x \notin \text{collect-free}(terms,\ \mathtt{1})))$

$\rightarrow \quad (\text{list-f-equal}(\text{del}(x,\ vars),\ \text{del-pairs}(x,\ vars,\ terms))$
$\qquad = \quad \text{elim-free-terms}(x,\ \text{list-f-equal}(vars,\ terms)))$

THEOREM: var-list-del

$\text{var-list}(vars,\ \text{length}(vars)) \rightarrow \text{var-list}(\text{del}(x,\ vars),\ \text{length}(\text{del}(x,\ vars)))$

THEOREM: append-nil-one
$(y \neq \textbf{nil}) \rightarrow (\text{append}\,(x,\ y) \neq \textbf{nil})$

THEOREM: append-nil-two
$(\text{append}\,(x,\ \textbf{nil}) \neq \textbf{nil}) \rightarrow (\text{append}\,(x,\ y) \neq \textbf{nil})$

THEOREM: del-pairs-collect-free
$(\text{collect-free}\,(\mathit{terms},\ \mathtt{1}) = \textbf{nil})$
$\rightarrow \quad (\text{collect-free}\,(\text{del-pairs}\,(x,\ \mathit{vars},\ \mathit{terms}),\ \mathtt{1}) = \textbf{nil})$

THEOREM: nlistp-term-equal
$((x \neq y) \wedge (\mathit{list} \simeq \textbf{nil})) \rightarrow (\neg\ \text{term-equal}\,(x,\ y,\ \mathit{list},\ \mathit{flg}))$

THEOREM: nlistp-form-equal
$((x \neq y) \wedge (\mathit{list} \simeq \textbf{nil})) \rightarrow (\neg\ \text{form-equal}\,(x,\ y,\ \mathit{list}))$

THEOREM: form-equal-parallel-subst
$(\text{var-list}\,(\mathit{vars},\ \text{length}\,(\mathit{vars}))$
$\wedge \quad \text{formula}\,(\mathit{exp},\ \mathit{symbols})$
$\wedge \quad (\text{collect-free}\,(\mathit{terms},\ \mathtt{1}) = \textbf{nil}))$
$\rightarrow \quad \text{form-equal}\,(\mathit{exp},$
$\qquad\qquad \text{parallel-subst}\,(\mathit{exp},\ \mathit{vars},\ \mathit{terms},\ \mathtt{0}),$
$\qquad\qquad \text{list-f-equal}\,(\mathit{vars},\ \mathit{terms}))$

THEOREM: term-equal-subst
$(\text{variable}\,(\mathit{var}) \wedge \text{termp}\,(\mathit{exp},\ \mathit{flg},\ \mathit{symbols}))$
$\rightarrow \quad \text{term-equal}\,(\mathit{exp},\ \text{subst}\,(\mathit{exp},\ \mathit{var},\ \mathit{term},\ \mathit{flg}),\ \text{list}\,(\text{f-equal}\,(\mathit{var},\ \mathit{term})),\ \mathit{flg})$

THEOREM: form-equal-subst
$(\text{variable}\,(\mathit{var}) \wedge \text{formula}\,(\mathit{exp},\ \mathit{symbols}) \wedge \text{free-for}\,(\mathit{exp},\ \mathit{var},\ \mathit{term},\ \mathtt{0}))$
$\rightarrow \quad \text{form-equal}\,(\mathit{exp},\ \text{subst}\,(\mathit{exp},\ \mathit{var},\ \mathit{term},\ \mathtt{0}),\ \text{list}\,(\text{f-equal}\,(\mathit{var},\ \mathit{term})))$

THEOREM: symb-equals
$\text{symb}\,(\mathit{sym},\ \mathit{symbols}) = (\text{memb}\,(\mathit{sym},\ \mathit{symbols}) \vee (\mathit{sym} = \text{p}\,(\mathtt{0},\ \mathtt{2})))$

THEOREM: index-not-equal
$((x \in \mathbf{N}) \wedge (x \neq \mathtt{0})) \rightarrow (\text{p}\,(x,\ y) \neq \text{p}\,(\mathtt{0},\ \mathtt{2}))$

THEOREM: fn-p-not-equal
$\text{fn}\,(x,\ y) \neq \text{p}\,(\mathtt{0},\ \mathtt{2})$

THEOREM: memb-fn
$\text{memb}\,(\text{fn}\,(x,\ y),\ \mathit{symbols}) = (\text{fn}\,(x,\ y) \in \mathit{symbols})$

THEOREM: memb-p
$\text{memb}\,(\text{p}\,(x,\ y),\ \mathit{symbols}) = (\text{p}\,(x,\ y) \in \mathit{symbols})$

`;;;;;;;The next section contains results that follow from the Z2 set axioms.`

THEOREM: formula-isin
formula (isin (x, y), *symbols*)
$=$ (memb (p (1, 2), *symbols*) $\land$ termp (x, 0, *symbols*) $\land$ termp (y, 0, *symbols*))

THEOREM: equal-axiom2-isin
(termp (list (x, y, u, v), 1, *symbols*) $\land$ (p (1, 2) $\in$ *symbols*))
$\rightarrow$ proves (equal-axiom2-proof (p (1, 2), list (x, y), list (u, v)),
f-implies (f-equal (x, u),
f-implies (f-equal (y, v),
f-implies (isin (x, y), isin (u, v)))),
given,
defns,
symbols)

THEOREM: collect-free-isin
collect-free (isin (y, z), 0) $=$ collect-free (list (y, z), 1)

THEOREM: covering-isin
covering (isin (x, y), z, 0) $=$ covering (list (x, y), z, 1)

THEOREM: subst-isin
subst (isin (x, y), *var*, *term*, 0)
$=$ isin (subst (x, *var*, *term*, 0), subst (y, *var*, *term*, 0))

THEOREM: parallel-subst-isin
parallel-subst (isin (x, y), *vars*, *terms*, 0)
$=$ isin (parallel-subst (x, *vars*, *terms*, 0), parallel-subst (y, *vars*, *terms*, 0))

EVENT: Enable isin; name this event ‘g0381’.

DEFINITION:
mk-bnd (*list*, n) $=$ ((1 + n) + find-hi-index (collect-free (*list*, 1)))

THEOREM: lessp-mk-bnd
find-hi-index (collect-free (*list*, 1)) $<$ mk-bnd (*list*, n)

THEOREM: mk-bnd-nequal
(fix (m) $\neq$ fix (n)) $\rightarrow$ (mk-bnd (*list*, m) $\neq$ mk-bnd (*list*, n))

EVENT: Enable mk-bnd; name this event ‘g0612’.

THEOREM: mk-bnd-not-free1
mk-bnd (list (x), n) $\notin$ collect-free (x, 0)

THEOREM: mk-bnd-not-free2

$(\text{mk-bnd}(\text{list}(x, y), n) \notin \text{collect-free}(x, \mathtt{0}))$
$\land \quad (\text{mk-bnd}(\text{list}(x, y), n) \notin \text{collect-free}(y, \mathtt{0}))$

THEOREM: mk-bnd-not-free3

$(\text{mk-bnd}(\text{list}(x, y, z), n) \notin \text{collect-free}(x, \mathtt{0}))$
$\land \quad (\text{mk-bnd}(\text{list}(x, y, z), n) \notin \text{collect-free}(y, \mathtt{0}))$
$\land \quad (\text{mk-bnd}(\text{list}(x, y, z), n) \notin \text{collect-free}(z, \mathtt{0}))$

```
;;;;;;;The next few rules are usable versions of the set-theory axioms.
```

DEFINITION:

```
ext-axiom-eq-elim1-proof (terma, termb, termc, pf)
=   tautconseq-proof (list (ext-axiom (mk-bnd (list (terma, termb), 0),
                                       terma,
                                       termb),
                            f-implies (∀ mk-bnd, (list terma termb), 0 f-iff (isin (mk-bnd (list (terma,
                                                                                                  termb),
                                                                                            0),
                                                                                     terma),
                                                                               isin (mk-bnd (list (terma,
                                                                                                  termb),
                                                                                            0),
                                                                                     termb)),
                                       f-iff (isin (termc, terma),
                                              isin (termc, termb))),
                            isin (termc, terma)),
                      f-implies (f-equal (terma, termb), isin (termc, termb)),
                      list (ext-axiom-proof (mk-bnd (list (terma, termb), 0),
                                             terma,
                                             termb),
                            subst-thm-b-proof (f-iff (isin (mk-bnd (list (terma,
                                                                          termb),
                                                                    0),
                                                             terma),
                                                      isin (mk-bnd (list (terma,
                                                                          termb),
                                                                    0),
                                                             termb)),
                                               list (mk-bnd (list (terma, termb),
                                                             0)),
                                               list (termc)),
                            pf))
```

THEOREM: nil-intersect-del

$\text{nil-intersect}(x, \text{del}(y, z)) \to \text{nil-intersect}(\text{del}(y, x), z)$

THEOREM: not-free-parallel-subst
nil-intersect (*vars*, collect-free (*exp*, *flg*))
→ (parallel-subst (*exp*, *vars*, *terms*, *flg*) = *exp*)

THEOREM: parallel-subst-f-iff
parallel-subst (f-iff (*x*, *y*), *vars*, *terms*, 0)
= f-iff (parallel-subst (*x*, *vars*, *terms*, 0), parallel-subst (*y*, *vars*, *terms*, 0))

```
;;;;;;;One case of the extensionality axiom: "(c in a)"|- "(a=b)->(c in b)".
```

THEOREM: ext-axiom-eq-elim1
(termp (*termb*, 0, *symbols*)
∧ (p (1, 2) ∈ *symbols*)
∧ proves (*pf*, isin (*termc*, *terma*), *given*, *defns*, *symbols*))
→ proves (ext-axiom-eq-elim1-proof (*terma*, *termb*, *termc*, *pf*),
f-implies (f-equal (*terma*, *termb*), isin (*termc*, *termb*)),
given,
defns,
symbols)

DEFINITION:
ext-axiom-eq-elim2-proof (*terma*, *termb*, *termc*, *pf*)
= tautconseq-proof (list (ext-axiom (mk-bnd (list (*terma*, *termb*), 0),
terma,
termb),
f-implies (∀ *mk-bnd*, (list terma termb), 0 f-iff (isin (mk-bnd (list (*terma*,
termb),
0),
terma),
isin (mk-bnd (list (*terma*,
termb),
0),
termb)),
f-iff (isin (*termc*, *terma*),
isin (*termc*, *termb*))),
isin (*termc*, *termb*)),
f-implies (f-equal (*terma*, *termb*), isin (*termc*, *terma*)),
list (ext-axiom-proof (mk-bnd (list (*terma*, *termb*), 0),
terma,
termb),
subst-thm-b-proof (f-iff (isin (mk-bnd (list (*terma*,
termb),
0),
terma),
isin (mk-bnd (list (*terma*,

```
                                                        termb),
                                                   0),
                                          termb)),
                                     list (mk-bnd (list (terma, termb),
                                                  0)),
                                     list (termc)),
                        pf))
```

;;;;;;;Second case of the extensionality axiom: "(c in b)" |- "(a=b)->(c in a)".

THEOREM: ext-axiom-eq-elim2
(termp (*terma*, 0, *symbols*)
 ∧ (p (1, 2) ∈ *symbols*)
 ∧ proves (*pf*, isin (*termc*, *termb*), *given*, *defns*, *symbols*))
→ proves (ext-axiom-eq-elim2-proof (*terma*, *termb*, *termc*, *pf*),
  f-implies (f-equal (*terma*, *termb*), isin (*termc*, *terma*)),
  given,
  defns,
  symbols)

EVENT: Enable ext-axiom-eq-elim1-proof; name this event 'g0631'.

EVENT: Enable ext-axiom-eq-elim2-proof; name this event 'g0632'.

EVENT: Enable f-iff-mp-proof2; name this event 'g0633'.

DEFINITION:
ext-axiom-eq-intro-proof (*x*, *terma*, *termb*, *pf*)
= f-iff-mp-proof2 (f-equal (*terma*, *termb*),
  ∀ *x* f-iff (isin (*x*, *terma*), isin (*x*, *termb*)),
  ext-axiom-proof (*x*, *terma*, *termb*),
  generalise-proof (*x*,
    f-iff (isin (*x*, *terma*), isin (*x*, *termb*)),
    pf))

;;;;;;;Third case of ext. axiom: "(x in a) <-> (x in b)" |- "(a=b)".

THEOREM: ext-axiom-eql-intro
(variable (*x*)
 ∧ (*x* ∉ collect-free (list (*terma*, *termb*), 1))
 ∧ proves (*pf*, f-iff (isin (*x*, *terma*), isin (*x*, *termb*)), *given*, *defns*, *symbols*))
→ proves (ext-axiom-eq-intro-proof (*x*, *terma*, *termb*, *pf*),
  f-equal (*terma*, *termb*),

$\quad\quad$ *given*,
$\quad\quad$ *defns*,
$\quad\quad$ *symbols*)

Event: Enable ext-axiom-eq-intro-proof; name this event ‘g0634’.

Theorem: symb-reduc
$(sym \in symbols) \rightarrow \text{symb}\,(sym,\ symbols)$

Event: Enable symb; name this event ‘g0635’.

Theorem: equal-axiom1-z-pair
$$
\begin{aligned}
&(\text{termp}\,(\text{list}\,(x,\ y,\ u,\ v),\ \mathtt{1},\ symbols) \land (\text{fn}\,(\mathtt{1},\ \mathtt{2}) \in symbols))\\
\rightarrow\quad &\text{proves}\,(\text{equal-axiom1-proof}\,(\text{fn}\,(\mathtt{1},\ \mathtt{2}),\ \text{list}\,(x,\ y),\ \text{list}\,(u,\ v)),\\
&\qquad\quad \text{f-implies}\,(\text{f-equal}\,(x,\ u),\\
&\qquad\qquad\qquad \text{f-implies}\,(\text{f-equal}\,(y,\ v),\\
&\qquad\qquad\qquad\qquad\quad \text{f-equal}\,(\text{z-pair}\,(x,\ y),\ \text{z-pair}\,(u,\ v)))),\\
&\qquad\quad given,\\
&\qquad\quad defns,\\
&\qquad\quad symbols)
\end{aligned}
$$

Theorem: equal-axiom1-z-union
$$
\begin{aligned}
&(\text{termp}\,(\text{list}\,(x,\ y,\ u,\ v),\ \mathtt{1},\ symbols) \land (\text{fn}\,(\mathtt{2},\ \mathtt{2}) \in symbols))\\
\rightarrow\quad &\text{proves}\,(\text{equal-axiom1-proof}\,(\text{fn}\,(\mathtt{2},\ \mathtt{2}),\ \text{list}\,(x,\ y),\ \text{list}\,(u,\ v)),\\
&\qquad\quad \text{f-implies}\,(\text{f-equal}\,(x,\ u),\\
&\qquad\qquad\qquad \text{f-implies}\,(\text{f-equal}\,(y,\ v),\\
&\qquad\qquad\qquad\qquad\quad \text{f-equal}\,(\text{z-union}\,(x,\ y),\ \text{z-union}\,(u,\ v)))),\\
&\qquad\quad given,\\
&\qquad\quad defns,\\
&\qquad\quad symbols)
\end{aligned}
$$

Theorem: collect-free-z-pair
$\text{collect-free}\,(\text{z-pair}\,(y,\ z),\ \mathtt{0}) = \text{collect-free}\,(\text{list}\,(y,\ z),\ \mathtt{1})$

Theorem: covering-z-pair
$\text{covering}\,(\text{z-pair}\,(x,\ y),\ var,\ \mathtt{0}) = \text{covering}\,(\text{list}\,(x,\ y),\ var,\ \mathtt{1})$

Theorem: subst-z-pair
$$
\begin{aligned}
&\text{subst}\,(\text{z-pair}\,(x,\ y),\ var,\ term,\ \mathtt{0})\\
=\quad &\text{z-pair}\,(\text{subst}\,(x,\ var,\ term,\ \mathtt{0}),\ \text{subst}\,(y,\ var,\ term,\ \mathtt{0}))
\end{aligned}
$$

Theorem: parallel-subst-z-pair
$$
\begin{aligned}
&\text{parallel-subst}\,(\text{z-pair}\,(x,\ y),\ vars,\ terms,\ \mathtt{0})\\
=\quad &\text{z-pair}\,(\text{parallel-subst}\,(x,\ vars,\ terms,\ \mathtt{0}),\ \text{parallel-subst}\,(y,\ vars,\ terms,\ \mathtt{0}))
\end{aligned}
$$

THEOREM: termp-z-pair
termp (z-pair (x, y), 0, *symbols*)
$=$ ((fn (1, 2) $\in$ *symbols*)
$\land$ termp (x, 0, *symbols*)
$\land$ termp (y, 0, *symbols*))

THEOREM: term-equal-z-pair
(term-equal (x, u, *list*, 0) $\land$ term-equal (y, v, *list*, 0))
$\rightarrow$ term-equal (z-pair (x, y), z-pair (u, v), *list*, 0)

THEOREM: term-equal-z-union
(term-equal (x, u, *list*, 0) $\land$ term-equal (y, v, *list*, 0))
$\rightarrow$ term-equal (z-union (x, y), z-union (u, v), *list*, 0)

EVENT: Enable z-pair; name this event 'g0387'.

THEOREM: termp-z-union
termp (z-union (x, y), 0, *symbols*)
$=$ ((fn (2, 2) $\in$ *symbols*)
$\land$ termp (x, 0, *symbols*)
$\land$ termp (y, 0, *symbols*))

THEOREM: collect-free-z-union
collect-free (z-union (y, z), 0) $=$ collect-free (list (y, z), 1)

THEOREM: covering-z-union
covering (z-union (x, y), *var*, 0) $=$ covering (list (x, y), *var*, 1)

THEOREM: subst-z-union
subst (z-union (x, y), *var*, *term*, 0)
$=$ z-union (subst (x, *var*, *term*, 0), subst (y, *var*, *term*, 0))

THEOREM: parallel-subst-z-union
parallel-subst (z-union (x, y), *vars*, *terms*, 0)
$=$ z-union (parallel-subst (x, *vars*, *terms*, 0),
parallel-subst (y, *vars*, *terms*, 0))

EVENT: Enable z-union; name this event 'g0388'.

DEFINITION:
z-pair-elim-proof (x, *terma*, *termb*, *exp*, *pf1*, *pf2*)
$=$ tautconseq-proof (list (f-iff (isin (x, z-pair (*terma*, *termb*)),
f-or (f-equal (x, *terma*), f-equal (x, *termb*))),
f-implies (f-equal (x, *terma*), *exp*),
f-implies (f-equal (x, *termb*), *exp*)),
f-implies (isin (x, z-pair (*terma*, *termb*)), *exp*),
list (pair-axiom-proof (x, *terma*, *termb*), *pf1*, *pf2*))

```
;;;;;;;Pair-elimination:  if |-"(x = a)->exp" and |-"(x=b)->exp",
;;;;;;;then |-"(x in {a,b}) -> exp".
```

THEOREM: z-pair-elim
(subset (list (p (1, 2), fn (1, 2)), *symbols*)
∧ proves (*pf1*, f-implies (f-equal (x, *terma*), *exp*), *given*, *defns*, *symbols*)
∧ proves (*pf2*, f-implies (f-equal (x, *termb*), *exp*), *given*, *defns*, *symbols*))
→ proves (z-pair-elim-proof (x, *terma*, *termb*, *exp*, *pf1*, *pf2*),
f-implies (isin (x, z-pair (*terma*, *termb*)), *exp*),
given,
defns,
symbols)

EVENT: Enable z-pair-elim-proof; name this event 'g0643'.

DEFINITION:
isin-pair1-proof (*terma*, *termb*)
= f-iff-mp-proof2 (isin (*terma*, z-pair (*terma*, *termb*)),
f-or (f-equal (*terma*, *terma*), f-equal (*terma*, *termb*)),
pair-axiom-proof (*terma*, *terma*, *termb*),
rt-expan-proof (f-equal (*terma*, *terma*),
f-equal (*terma*, *termb*),
ident-axiom-proof (*terma*)))

```
;;;;;;; |-"(x in {x, y})".
```

THEOREM: isin-pair1-proof-proves
formula (isin (*terma*, z-pair (*terma*, *termb*)), *symbols*)
→ proves (isin-pair1-proof (*terma*, *termb*),
isin (*terma*, z-pair (*terma*, *termb*)),
given,
defns,
symbols)

EVENT: Enable isin-pair1-proof; name this event 'g0391'.

DEFINITION:
isin-pair2-proof (*terma*, *termb*)
= f-iff-mp-proof2 (isin (*termb*, z-pair (*terma*, *termb*)),
f-or (f-equal (*termb*, *terma*), f-equal (*termb*, *termb*)),
pair-axiom-proof (*termb*, *terma*, *termb*),
expan-proof (f-equal (*termb*, *terma*),
f-equal (*termb*, *termb*),
ident-axiom-proof (*termb*)))

```
;;;;;;; |- "(y in {x,y})".
```

THEOREM: isin-pair2-proof-proves
formula (isin ($termb$, z-pair ($terma$, $termb$)), $symbols$)
$\rightarrow$ proves (isin-pair2-proof ($terma$, $termb$),
isin ($termb$, z-pair ($terma$, $termb$)),
$given$,
$defns$,
$symbols$)

EVENT: Enable isin-pair2-proof; name this event 'g0392'.

DEFINITION: z-sing (x) = z-pair (x, x)

DEFINITION:
isin-sing-proof (x, y)
= chain-iff-proof (isin (y, z-sing (x)),
f-equal (y, x),
list (f-or (f-equal (y, x), f-equal (y, x))),
list (pair-axiom-proof (y, x, x),
tautconseq-proof (**nil**,
f-iff (f-or (f-equal (y, x),
f-equal (y, x)),
f-equal (y, x)),
nil)))

```
;;;;;;; |- "(y in {x}) <-> (y=x)".
```

THEOREM: isin-sing
formula (f-iff (isin (y, z-sing (x)), f-equal (y, x)), $symbols$)
$\rightarrow$ proves (isin-sing-proof (x, y),
f-iff (isin (y, z-sing (x)), f-equal (y, x)),
$given$,
$defns$,
$symbols$)

EVENT: Enable isin-sing-proof; name this event 'g0913'.

THEOREM: termp-z-sing
termp (z-sing (x), 0, $symbols$)
= ((fn (1, 2) $\in$ $symbols$) $\wedge$ termp (x, 0, $symbols$))

THEOREM: collect-free-z-sing
collect-free (z-sing (y), 0) = collect-free (list (y, y), 1)

THEOREM: covering-z-sing
covering (z-sing (x), *var*, 0) = covering (list (x, x), *var*, 1)

THEOREM: subst-z-sing
subst (z-sing (x), *var*, *term*, 0) = z-sing (subst (x, *var*, *term*, 0))

THEOREM: parallel-subst-z-sing
parallel-subst (z-sing (x), *vars*, *terms*, 0)
= z-sing (parallel-subst (x, *vars*, *terms*, 0))

THEOREM: term-equal-z-sing
term-equal (x, y, *list*, 0) $\rightarrow$ term-equal (z-sing (x), z-sing (y), *list*, 0)

EVENT: Enable z-sing; name this event 'g0389'.

DEFINITION:
isin-sing1-proof (x)
= f-iff-mp-proof2 (isin (x, z-sing (x)),
f-equal (x, x),
isin-sing-proof (x, x),
ident-axiom-proof (x))

```
;;;;;;; |- "(x in {x})".
```

THEOREM: isin-sing1-proof-proves
formula (isin (*term*, z-sing (*term*)), *symbols*)
$\rightarrow$ proves (isin-sing1-proof (*term*),
isin (*term*, z-sing (*term*)),
given,
defns,
symbols)

EVENT: Enable isin-sing1-proof; name this event 'g0393'.

DEFINITION:
del-forsome-proof (x, *exp*)
= iff-join-proof (forsome (x, *exp*),
exp,
forsome-intro-proof (x, *exp*, *exp*, prop-axiom-proof (*exp*)),
subst-thm-a-proof (*exp*, list (x), list (x)))

```
;;;;;;; If "x" not free in "exp", then |- "(forsome x exp) <-> exp".
```

THEOREM: del-forsome-proof-proves
(variable (x) ∧ formula $(exp, symbols)$ ∧ $(x \notin$ collect-free $(exp, 0)))$
→ proves (del-forsome-proof (x, exp),
f-iff (forsome (x, exp), exp),
given,
defns,
symbols)

EVENT: Enable del-forsome-proof; name this event 'g0597'.

DEFINITION:
del-forall-proof (x, exp)
= iff-join-proof ($\forall\ x\ exp$,
exp,
subst-thm-b-proof (exp, list (x), list (x)),
forall-intro-proof (exp, exp, x, prop-axiom-proof (exp)))

```
;;;;;;; If "x" not free in "exp", then |- "(forall x exp) <-> exp".
```

THEOREM: del-forall-proof-proves
(variable (x) ∧ formula $(exp, symbols)$ ∧ $(x \notin$ collect-free $(exp, 0)))$
→ proves (del-forall-proof (x, exp), f-iff ($\forall\ x\ exp$, exp), *given*, *defns*, *symbols*)

EVENT: Enable del-forall-proof; name this event 'g0599'.

DEFINITION:
sing-sing-equal-proof (x, y)
= chain-proof (f-equal (z-sing (x), z-sing (y)),
f-equal (x, y),
list (isin (x, z-sing (y))),
list (ext-axiom-eq-elim1-proof (z-sing (x),
z-sing (y),
x,
isin-sing1-proof (x)),
iff-split-proof1 (isin (x, z-sing (y)),
f-equal (x, y),
isin-sing-proof (y, x))))

```
;;;;;;; |- "({x}={y}) -> (x=y)".
```

THEOREM: sing-sing-equal-proof-proves
(termp (list (x, y), 1, *symbols*) ∧ subset (list (p (1, 2), fn (1, 2)), *symbols*))
→ proves (sing-sing-equal-proof (x, y),
f-implies (f-equal (z-sing (x), z-sing (y)), f-equal (x, y)),

given,
defns,
symbols)

EVENT: Enable sing-sing-equal-proof; name this event 'g0652'.

DEFINITION:
sing-pair-equal-proof1 (x, u, v)
= chain-proof (f-equal (z-sing (x), z-pair (u, v)),
f-equal (x, u),
list (isin (u, z-sing (x)), f-equal (u, x)),
list (ext-axiom-eq-elim2-proof (z-sing (x),
z-pair (u, v),
u,
isin-pair1-proof (u, v)),
iff-split-proof1 (isin (u, z-sing (x)),
f-equal (u, x),
isin-sing-proof (x, u)),
symmetry-proof1 (u, x)))

```
;;;;;;; |- "({x}={u,v}) -> (x=u)".
```

THEOREM: sing-pair-equal-proof1-proves
(termp (list (x, u, v), 1, *symbols*) $\wedge$ subset (list (p (1, 2), fn (1, 2)), *symbols*))
$\rightarrow$ proves (sing-pair-equal-proof1 (x, u, v),
f-implies (f-equal (z-sing (x), z-pair (u, v)), f-equal (x, u)),
given,
defns,
symbols)

EVENT: Enable sing-pair-equal-proof1; name this event 'g0653'.

DEFINITION:
sing-pair-equal-proof2 (x, u, v)
= chain-proof (f-equal (z-sing (x), z-pair (u, v)),
f-equal (x, v),
list (isin (v, z-sing (x)), f-equal (v, x)),
list (ext-axiom-eq-elim2-proof (z-sing (x),
z-pair (u, v),
v,
isin-pair2-proof (u, v)),
iff-split-proof1 (isin (v, z-sing (x)),
f-equal (v, x),
isin-sing-proof (x, v)),
symmetry-proof1 (v, x)))

```
;;;;;;; |-"({x}={u,v}) -> (x=v)".
```

THEOREM: sing-pair-equal-proof-proves
(termp (list (x, u, v), `1`, *symbols*) $\land$ subset (list (p (`1`, `2`), fn (`1`, `2`)), *symbols*))
$\rightarrow$ proves (sing-pair-equal-proof2 (x, u, v),
f-implies (f-equal (z-sing (x), z-pair (u, v)), f-equal (x, v)),
given,
defns,
symbols)

EVENT: Enable sing-pair-equal-proof2; name this event ‘g0654’.

DEFINITION: z-opair (x, y) = z-pair (z-sing (x), z-pair (x, y))

THEOREM: termp-z-opair
termp (z-opair (x, y), `0`, *symbols*)
= ((fn (`1`, `2`) $\in$ *symbols*)
$\land$ termp (x, `0`, *symbols*)
$\land$ termp (y, `0`, *symbols*))

THEOREM: term-equal-z-opair
((f-equal (x, u) $\in$ *list*) $\land$ (f-equal (y, v) $\in$ *list*))
$\rightarrow$ term-equal (z-opair (x, y), z-opair (u, v), *list*, `0`)

DEFINITION:
z-car-unique-proof (x, y, u, v)
= chain-proof (f-equal (z-opair (x, y), z-opair (u, v)),
f-equal (x, u),
list (isin (z-sing (x), z-opair (u, v))),
list (ext-axiom-eq-elim1-proof (z-opair (x, y),
z-opair (u, v),
z-sing (x),
isin-pair1-proof (z-sing (x),
z-pair (x, y))),
z-pair-elim-proof (z-sing (x),
z-sing (u),
z-pair (u, v),
f-equal (x, u),
sing-sing-equal-proof (x, u),
sing-pair-equal-proof1 (x, u, v))))

```
;;;;;;; |- "(<x y>=<u v>)-> (x=u)".
```

THEOREM: z-car-unique-proof-proves
(termp (list (x, y, u, v), `1`, *symbols*) $\land$ subset (list (p (`1`, `2`), fn (`1`, `2`)), *symbols*))

```
→    proves (z-car-unique-proof (x, y, u, v),
             f-implies (f-equal (z-opair (x, y), z-opair (u, v)), f-equal (x, u)),
             given,
             defns,
             symbols)
```

EVENT: Enable z-car-unique-proof; name this event ‘g0656’.

DEFINITION:

```
z-cdr-unique-step-proof (x, y, u, v)
=    chain-proof (f-equal (z-opair (x, y), z-opair (u, v)),
                  f-or (f-equal (y, u), f-equal (y, v)),
                  list (isin (z-pair (x, y), z-opair (u, v))),
                  list (ext-axiom-eq-elim1-proof (z-opair (x, y),
                                                  z-opair (u, v),
                                                  z-pair (x, y),
                                                  isin-pair2-proof (z-sing (x),
                                                                    z-pair (x, y))),
                        z-pair-elim-proof (z-pair (x, y),
                                           z-sing (u),
                                           z-pair (u, v),
                                           f-or (f-equal (y, u), f-equal (y, v)),
                                           tautconseq-proof (list (f-implies (f-equal (z-pair (x,
                                                                                               y),
                                                                                       z-sing (u)),
                                                                              f-equal (y,
                                                                                       u))),
                                                                   f-implies (f-equal (z-pair (x,
                                                                                               y),
                                                                                       z-sing (u)),
                                                                              f-or (f-equal (y,
                                                                                             u),
                                                                                    f-equal (y,
                                                                                             v))),
                                                                   list (chain-proof (f-equal (z-pair (x,
                                                                                                       y),
                                                                                               z-sing (u)),
                                                                                      f-equal (y,
                                                                                               u),
                                                                                      list (f-equal (z-sing (u),
                                                                                                     z-pair (x,
                                                                                                             y)),
                                                                                            f-equal (u,
                                                                                                     y)),
```

```
                                  list (symmetry-proof1 (z-pair (x,
                                                                 y),
                                                          z-sing (u)),
                                        sing-pair-equal-proof2 (u,
                                                                x,
                                                                y),
                                        symmetry-proof1 (u,
                                                         y))))),
chain-proof (f-equal (z-pair (x, y),
                      z-pair (u, v)),
             f-or (f-equal (y, u),
                   f-equal (y, v)),
             list (isin (y,
                         z-pair (u, v))),
             list (ext-axiom-eq-elim1-proof (z-pair (x,
                                                     y),
                                             z-pair (u,
                                                     v),
                                             y,
                                             isin-pair2-proof (x,
                                                               y)),
                   iff-split-proof1 (isin (y,
                                           z-pair (u,
                                                   v)),
                                     f-or (f-equal (y,
                                                    u),
                                           f-equal (y,
                                                    v)),
                                     pair-axiom-proof (y,
                                                       u,
                                                       v)))))))
```

```
;;;;;;; |- "(<x y>=<u v>) -> ((y=u) v (y=v))".
```

THEOREM: z-cdr-unique-step
(termp (list (x, y, u, v), 1, *symbols*) $\land$ subset (list (p (1, 2), fn (1, 2)), *symbols*))
$\rightarrow$ proves (z-cdr-unique-step-proof (x, y, u, v),
f-implies (f-equal (z-opair (x, y), z-opair (u, v)),
f-or (f-equal (y, u), f-equal (y, v))),
given,
defns,
symbols)

EVENT: Enable z-cdr-unique-step-proof; name this event ‘g0657’.

DEFINITION:
trans-equal-proof1 (x, y, z)
$=$ tautconseq-proof (list (list-implies (list (f-equal (y, x),
f-equal (z, z),
f-equal (y, z)),
f-equal (x, z)),
f-equal (z, z)),
list-implies (list (f-equal (y, x), f-equal (y, z)),
f-equal (x, z))),
list (equal-axiom2-proof (p (0, 2), list (y, z), list (x, z)),
ident-axiom-proof (z)))

```
;;;;;;; |-"(y = x) -> ((y = z) -> (x = z))".
```

THEOREM: trans-equal-proof-proves
(termp $(x$, 0, *symbols*$)$ $\wedge$ termp $(y$, 0, *symbols*$)$ $\wedge$ termp $(z$, 0, *symbols*$)$)
$\rightarrow$ proves (trans-equal-proof1 (x, y, z),
f-implies (f-equal (y, x), f-implies (f-equal (y, z), f-equal (x, z))),
given,
defns,
symbols)

EVENT: Enable trans-equal-proof1; name this event 'g0362'.

THEOREM: termp-covering
termp (*exp*, *flg*, *symbols*) $\rightarrow$ (covering (*exp*, *var*, *flg*) $=$ **nil**)

THEOREM: atomp-covering
atomp (*exp*, *symbols*) $\rightarrow$ (covering (*exp*, *var*, 0) $=$ **nil**)

THEOREM: collect-free-z-opair
collect-free (z-opair (y, z), 0)
$=$ collect-free (list (z-sing (y), z-pair (y, z)), 1)

EVENT: Enable z-opair; name this event 'g0366'.

DEFINITION:
trans-equal-proof2 (x, y, z)
$=$ tautconseq-proof (list (f-implies (f-equal (x, y), f-equal (y, x)),
f-implies (f-equal (z, y), f-equal (y, z)),
f-implies (f-equal (y, x),
f-implies (f-equal (y, z), f-equal (x, z)))),
f-implies (f-equal (x, y),
f-implies (f-equal (z, y), f-equal (x, z))),

list (symmetry-proof1 (x, y),
symmetry-proof1 (z, y),
trans-equal-proof1 (x, y, z)))

`;;;;;;; |- "(x = y) -> ((z = y) -> (x = z))".`

THEOREM: trans-equal-proof2-proves
(termp (x, 0, *symbols*) $\wedge$ termp (y, 0, *symbols*) $\wedge$ termp (z, 0, *symbols*))
$\rightarrow$ proves (trans-equal-proof2 (x, y, z),
f-implies (f-equal (x, y), f-implies (f-equal (z, y), f-equal (x, z))),
given,
defns,
symbols)

EVENT: Enable trans-equal-proof2; name this event 'g0367'.

DEFINITION:
trans-equal-proof3 (x, y, z)
= tautconseq-proof (list (f-implies (f-equal (y, z),
f-implies (f-equal (x, y), f-equal (x, z)))),
f-implies (f-equal (x, y),
f-implies (f-equal (y, z), f-equal (x, z))),
list (detach-proof (f-equal (x, x),
f-implies (f-equal (y, z),
f-implies (f-equal (x, y),
f-equal (x, z))),
ident-axiom-proof (x),
equal-axiom2-proof (p (0, 2),
list (x, y),
list (x, z)))))

`;;;;;;; |- "(x = y) -> ((y = z) -> (x = z))".`

THEOREM: trans-equal-proof3-proves
(termp (x, 0, *symbols*) $\wedge$ termp (y, 0, *symbols*) $\wedge$ termp (z, 0, *symbols*))
$\rightarrow$ proves (trans-equal-proof3 (x, y, z),
f-implies (f-equal (x, y), f-implies (f-equal (y, z), f-equal (x, z))),
given,
defns,
symbols)

EVENT: Enable trans-equal-proof3; name this event 'g0736'.

DEFINITION:

```
z-cdr-unique-proof (x, u, y, v)
=   tautconseq-proof (list (f-implies (f-equal (z-opair (x, u), z-opair (y, v)),
                                       f-or (f-equal (u, y), f-equal (u, v))),
                            f-implies (f-equal (z-opair (x, u), z-opair (y, v)),
                                       f-or (f-equal (v, x), f-equal (v, u))),
                            f-implies (f-equal (z-opair (x, u), z-opair (y, v)),
                                       f-equal (x, y)),
                            f-implies (f-equal (v, u), f-equal (u, v)),
                            f-implies (f-equal (u, y),
                                       f-implies (f-equal (x, y), f-equal (u, x))),
                            f-implies (f-equal (u, x),
                                       f-implies (f-equal (v, x), f-equal (u, v)))),
                      f-implies (f-equal (z-opair (x, u), z-opair (y, v)),
                                 f-equal (u, v)),
                      list (z-cdr-unique-step-proof (x, u, y, v),
                            chain-proof (f-equal (z-opair (x, u), z-opair (y, v)),
                                         f-or (f-equal (v, x), f-equal (v, u)),
                                         list (f-equal (z-opair (y, v),
                                                        z-opair (x, u))),
                                         list (symmetry-proof1 (z-opair (x, u),
                                                                z-opair (y, v)),
                                               z-cdr-unique-step-proof (y,
                                                                        v,
                                                                        x,
                                                                        u))),
                            z-car-unique-proof (x, u, y, v),
                            symmetry-proof1 (v, u),
                            trans-equal-proof2 (u, y, x),
                            trans-equal-proof2 (u, x, v)))
```

```
;;;;;;; |- "(<x u>=<u v>) -> (u = v)".
```

THEOREM: z-cdr-unique-proof-proves
 (termp $(x,$ 0, *symbols*)
 $\wedge$ termp $(u,$ 0, *symbols*)
 $\wedge$ termp $(y,$ 0, *symbols*)
 $\wedge$ termp $(v,$ 0, *symbols*)
 $\wedge$ (p (1, 2) $\in$ *symbols*)
 $\wedge$ (fn (1, 2) $\in$ *symbols*))
 $\rightarrow$ proves (z-cdr-unique-proof (x, u, y, v),
     f-implies (f-equal (z-opair (x, u), z-opair (y, v)), f-equal (u, v)),
     given,
     defns,
     symbols)

EVENT: Enable z-cdr-unique-proof; name this event 'g0369'.

```
;;;;;;;The next definition introduces the "if" connective which
;;;;;;;plays an important role in the proofs to follow.
;;;;;;;Def.: "(if A B C)" =  "((A ^ B) v ((not A) ^ C))".
```

DEFINITION: f-if (x, y, z) = f-or (f-and (x, y), f-and (f-not (x), z))

DEFINITION:
f-if-intro-step-proof1 (x, y, z)
= tautconseq-proof (**nil**, f-implies (x, f-implies (y, f-if (x, y, z))), **nil**)

```
;;;;;;; |- "(A -> (B -> (if A B C)))".
```

THEOREM: f-if-intro-step1
(formula (x, *symbols*) $\land$ formula (y, *symbols*) $\land$ formula (z, *symbols*))
$\rightarrow$ proves (f-if-intro-step-proof1 (x, y, z),
f-implies (x, f-implies (y, f-if (x, y, z))),
given,
defns,
symbols)

DEFINITION:
f-if-intro-step-proof2 (x, y, z)
= tautconseq-proof (**nil**, f-implies (f-not (x), f-implies (z, f-if (x, y, z))), **nil**)

```
;;;;;;; |- "((not A) -> (C -> (if A B C)))".
```

THEOREM: f-if-intro-step2
(formula (x, *symbols*) $\land$ formula (y, *symbols*) $\land$ formula (z, *symbols*))
$\rightarrow$ proves (f-if-intro-step-proof2 (x, y, z),
f-implies (f-not (x), f-implies (z, f-if (x, y, z))),
given,
defns,
symbols)

EVENT: Enable f-if-intro-step-proof1; name this event 'g0663'.

EVENT: Enable f-if-intro-step-proof2; name this event 'g0664'.

DEFINITION:
f-if-intro-proof1 (x, y, z, pfx, pfy)
= tautconseq-proof (list (x, y), f-if (x, y, z), list (pfx, pfy))

```
;;;;;;; If |- "A" and |- "B", then |- "(if A B C)".
```

THEOREM: f-if-intro-proof1-proves
(formula (x, *symbols*)
$\land$ formula (y, *symbols*)
$\land$ formula (z, *symbols*)
$\land$ proves (*pfx*, x, *given*, *defns*, *symbols*)
$\land$ proves (*pfy*, y, *given*, *defns*, *symbols*))
$\rightarrow$ proves (f-if-intro-proof1 (x, y, z, *pfx*, *pfy*),
f-if (x, y, z),
given,
defns,
symbols)

EVENT: Enable f-if-intro-proof1; name this event 'g0902'.

DEFINITION:
f-if-intro-proof2 (x, y, z, *pfnx*, *pfz*)
= tautconseq-proof (list (f-not (x), z), f-if (x, y, z), list (*pfnx*, *pfz*))

```
;;;;;;; If |- "(not A)" and |- "C", then |- "(if A B C)".
```

THEOREM: f-if-intro-proof2-proves
(formula (x, *symbols*)
$\land$ formula (y, *symbols*)
$\land$ formula (z, *symbols*)
$\land$ proves (*pfnx*, f-not (x), *given*, *defns*, *symbols*)
$\land$ proves (*pfz*, z, *given*, *defns*, *symbols*))
$\rightarrow$ proves (f-if-intro-proof2 (x, y, z, *pfnx*, *pfz*),
f-if (x, y, z),
given,
defns,
symbols)

EVENT: Enable f-if-intro-proof2; name this event 'g0903'.

DEFINITION:
f-if-elim-proof1 (x, y, z, *pfx*)
= tautconseq-proof (list (x), f-implies (f-if (x, y, z), y), list (*pfx*))

```
;;;;;;; If |- "A", then |- "((if A B C) -> C)".
```

THEOREM: f-if-elim-proof1-proves
(formula (x, *symbols*)

$\wedge$ formula (y, *symbols*)
$\wedge$ formula (z, *symbols*)
$\wedge$ proves (*pfx*, x, *given*, *defns*, *symbols*))
$\rightarrow$ proves (f-if-elim-proof1 (x, y, z, *pfx*),
f-implies (f-if (x, y, z), y),
given,
defns,
symbols)

EVENT: Enable f-if-elim-proof1; name this event 'g0904'.

DEFINITION:
f-if-elim-proof2 (x, y, z, *pfnx*)
= tautconseq-proof (list (f-not (x)), f-implies (f-if (x, y, z), z), list (*pfnx*))

```
;;;;;;; If |-"(not A)", then |-"((if A B C) -> C)".
```

THEOREM: f-if-elim-proof2-proves
(formula (x, *symbols*)
$\wedge$ formula (y, *symbols*)
$\wedge$ formula (z, *symbols*)
$\wedge$ proves (*pfnx*, f-not (x), *given*, *defns*, *symbols*))
$\rightarrow$ proves (f-if-elim-proof2 (x, y, z, *pfnx*),
f-implies (f-if (x, y, z), z),
given,
defns,
symbols)

EVENT: Enable f-if-elim-proof2; name this event 'g0905'.

THEOREM: formula-f-if
formula (f-if (x, y, z), *symbols*)
= (formula (x, *symbols*) $\wedge$ formula (y, *symbols*) $\wedge$ formula (z, *symbols*))

THEOREM: collect-free-f-if
collect-free (f-if (x, y, z), 0)
= collect-free (f-or (f-and (x, y), f-and (f-not (x), z)), 0)

THEOREM: subst-f-if
subst (f-if (x, y, z), *var*, *term*, 0)
= f-if (subst (x, *var*, *term*, 0), subst (y, *var*, *term*, 0), subst (z, *var*, *term*, 0))

THEOREM: parallel-subst-f-if
parallel-subst (f-if (x, y, z), *vars*, *terms*, 0)

$=$ f-if (parallel-subst (x, *vars*, *terms*, 0),
parallel-subst (y, *vars*, *terms*, 0),
parallel-subst (z, *vars*, *terms*, 0))

THEOREM: covering-f-if
covering (f-if (x, y, z), *var*, 0)
$=$ append (append (covering (x, *var*, 0), covering (y, *var*, 0)),
append (covering (x, *var*, 0), covering (z, *var*, 0)))

THEOREM: eval-f-if
eval (f-if (x, y, z), *alist*)
$=$ **if** eval (x, *alist*) **then** eval (y, *alist*)
else eval (z, *alist*) **endif**

EVENT: Enable f-if; name this event 'g0909'.

DEFINITION:
case-split-proof (x, y, *pf1*, *pf2*)
$=$ tautconseq-proof (list (f-implies (x, y), f-implies (f-not (x), y)),
y,
list (*pf1*, *pf2*))

```
;;;;;;;If |- "(A -> B)" and |- "((not A) -> B)", then |- "B".
```

THEOREM: case-split-proof-proves
(proves (*pf1*, f-implies (x, y), *given*, *defns*, *symbols*)
$\wedge$ proves (*pf2*, f-implies (f-not (x), y), *given*, *defns*, *symbols*))
$\rightarrow$ proves (case-split-proof (x, y, *pf1*, *pf2*), y, *given*, *defns*, *symbols*)

EVENT: Enable case-split-proof; name this event 'g0601'.

THEOREM: termp-phi
termp (PHI, 0, *symbols*) $=$ (fn (0, 0) $\in$ *symbols*)

THEOREM: collect-free-phi
collect-free (PHI, 0) $=$ **nil**

THEOREM: covering-phi
covering (PHI, *var*, 0) $=$ **nil**

THEOREM: subst-phi
subst (PHI, *var*, *term*, 0) $=$ PHI

THEOREM: parallel-subst-phi
parallel-subst (PHI, *vars*, *terms*, 0) $=$ PHI

EVENT: Enable phi; name this event 'g0668'.

THEOREM: proves-list-nil
proves-list (**nil**, **nil**, *given*, *defns*, *symbols*)

DEFINITION:
forsome-distrib-or-proof1 (x, a, b)
= forsome-intro-proof (x,
    f-or (a, b),
    f-or (forsome (x, a), forsome (x, b)),
    tautconseq-proof (list (f-implies (a, forsome (x, a)),
                            f-implies (b, forsome (x, b))),
                      f-implies (f-or (a, b),
                                 f-or (forsome (x, a),
                                       forsome (x, b))),
                      list (e-intro-proof (x, a),
                            e-intro-proof (x, b))))

```
;;;;;;; |- "((forsome x (A v B)) -> ((forsome x A) v (forsome x B)))".
```

THEOREM: forsome-distrib-or-proof-proves1
(variable (x) $\land$ formula (a, *symbols*) $\land$ formula (b, *symbols*))
$\rightarrow$ proves (forsome-distrib-or-proof1 (x, a, b),
    f-implies (forsome (x, f-or (a, b)), f-or (forsome (x, a), forsome (x, b))),
    given,
    defns,
    symbols)

DEFINITION:
forsome-distrib-step-proof (x, a, b)
= forsome-intro-proof (x,
    b,
    forsome (x, f-or (a, b)),
    chain-proof (b,
        forsome (x, f-or (a, b)),
        list (f-or (a, b)),
        list (tautconseq-proof (**nil**,
                                f-implies (b,
                                           f-or (a,
                                                 b)),
                                nil),
              e-intro-proof (x, f-or (a, b)))))

```
;;;;;;; |- "(forsome x B) -> (forsome x (A v B))".
```

THEOREM: forsome-distrib-step-proof-proves
(variable (x) $\wedge$ formula (a, *symbols*) $\wedge$ formula (b, *symbols*))
$\rightarrow$ proves (forsome-distrib-step-proof (x, a, b),
f-implies (forsome (x, b), forsome (x, f-or (a, b))),
given,
defns,
symbols)

EVENT: Enable forsome-distrib-or-proof1; name this event 'g0669'.

EVENT: Enable forsome-distrib-step-proof; name this event 'g0670'.

DEFINITION:
forsome-distrib-or-proof2 (x, a, b)
= tautconseq-proof (list (f-implies (forsome (x, a), forsome (x, f-or (a, b))),
f-implies (forsome (x, b), forsome (x, f-or (a, b)))),
f-implies (f-or (forsome (x, a), forsome (x, b)),
forsome (x, f-or (a, b))),
list (forsome-intro-proof (x,
a,
forsome (x, f-or (a, b)),
chain-proof (a,
forsome (x,
f-or (a,
b)),
list (f-or (a, b)),
list (tautconseq-proof (**nil**,
f-implies (a,
f-or (a,
b)),
nil),
e-intro-proof (x,
f-or (a,
b))))),
forsome-distrib-step-proof (x, a, b)))

```
;;;;;;; |- "(((forsome x A) v (forsome x B)) -> (forsome x (A v B)))".
```

THEOREM: forsome-distrib-or-proof-proves2
(variable (x) $\wedge$ formula (a, *symbols*) $\wedge$ formula (b, *symbols*))
$\rightarrow$ proves (forsome-distrib-or-proof2 (x, a, b),
f-implies (f-or (forsome (x, a), forsome (x, b)), forsome (x, f-or (a, b))),
given,
defns,

symbols)

DEFINITION:
forall-distrib-or-proof (x, a, b)
= tautconseq-proof (list (f-implies ($\forall$ x a, $\forall$ x f-or (a, b)),
f-implies ($\forall$ x b, $\forall$ x f-or (a, b))),
f-implies (f-or ($\forall$ x a, $\forall$ x b), $\forall$ x f-or (a, b)),
list (forall-intro-proof ($\forall$ x a,
f-or (a, b),
x,
tautconseq-proof (list (f-implies ($\forall$ x a,
a)),
f-implies ($\forall$ x a,
f-or (a,
b)),
list (all-elim-proof (x,
a)))),
forall-intro-proof ($\forall$ x b,
f-or (a, b),
x,
tautconseq-proof (list (f-implies ($\forall$ x b,
b)),
f-implies ($\forall$ x b,
f-or (a,
b)),
list (all-elim-proof (x,
b))))))

THEOREM: proves-list-reduc
proves-list (cons (*pf1*, *pf2*), cons (*exp1*, *exp2*), *given*, *defns*, *symbols*)
= (proves (*pf1*, *exp1*, *given*, *defns*, *symbols*)
$\land$ proves-list (*pf2*, *exp2*, *given*, *defns*, *symbols*))

```
;;;;;;; |-"((forall x A) v (forall x B)) -> (forall x (A v B))".
```

THEOREM: forall-distrib-or
(variable (x) $\land$ formula (a, *symbols*) $\land$ formula (b, *symbols*))
$\rightarrow$ proves (forall-distrib-or-proof (x, a, b),
f-implies (f-or ($\forall$ x a, $\forall$ x b), $\forall$ x f-or (a, b)),
given,
defns,
symbols)

EVENT: Enable forall-distrib-or-proof; name this event 'g0672'.

EVENT: Disable forall; name this event 'g0673'.

DEFINITION:
forsome-forall-proof$(x,\ a)$
= tautconseq-proof (list (f-iff (forsome $(x,$ f-not (f-not $(a))),$ forsome $(x,\ a))),$
f-iff (f-not (forsome $(x,\ a)),\ \forall\ x$ f-not $(a)),$
list (eqn-form-proof (forsome $(x,\ a),$
forsome $(x,$ f-not (f-not $(a))),$
list (f-iff (f-not (f-not $(a)),\ a)),$
list (tautconseq-proof (**nil**,
f-iff (f-not (f-not $(a)),$
$a),$
nil)))))

```
;;;;;;; |- "(not (forsome x A)) <-> (forall x (not A))".
```

THEOREM: forsome-forall
(variable $(x) \wedge$ formula $(a,\ \mathit{symbols}))$
$\rightarrow$ proves (forsome-forall-proof$(x,\ a),$
f-iff (f-not (forsome $(x,\ a)),\ \forall\ x$ f-not $(a)),$
given,
defns,
symbols)

DEFINITION:
forall-forsome-proof$(x,\ a)$
= tautconseq-proof (**nil**, f-iff (f-not $(\forall\ x\ a),$ forsome $(x,$ f-not $(a))),$ **nil**)

```
;;;;;;; |- "(not (forall x A)) <-> (forsome x (not A))".
```

THEOREM: forall-forsome
(variable $(x) \wedge$ formula $(a,\ \mathit{symbols}))$
$\rightarrow$ proves (forall-forsome-proof$(x,\ a),$
f-iff (f-not $(\forall\ x\ a),$ forsome $(x,$ f-not $(a))),$
given,
defns,
symbols)

EVENT: Enable forsome-forall-proof; name this event 'g0674'.

EVENT: Enable forall-forsome-proof; name this event 'g0675'.

EVENT: Enable forsome-distrib-or-proof2; name this event 'g0593'.

EVENT: Enable forall; name this event ‘g0676’.

DEFINITION:

```
forsome-distrib-and-proof (x, a, b)
=   chain-proof (f-and (a, forsome (x, b)),
                 forsome (x, f-and (a, b)),
                 list (f-not (f-or (f-not (a), f-not (forsome (x, b)))),
                       f-not (f-or (forsome (x, f-not (a)), f-not (forsome (x, b)))),
                       f-not (f-implies (forsome (x, b), forsome (x, f-not (a)))),
                       f-not (∀ x f-implies (b, f-not (a))),
                       forsome (x, f-not (f-implies (b, f-not (a))))),
                 list (tautconseq-proof (nil,
                                         f-implies (f-and (a, forsome (x, b)),
                                                    f-not (f-or (f-not (a),
                                                                 f-not (forsome (x,
                                                                                 b))))),
                                         nil),
                       iff-split-proof1 (f-not (f-or (f-not (a),
                                                      f-not (forsome (x, b)))),
                                         f-not (f-or (forsome (x, f-not (a)),
                                                      f-not (forsome (x, b)))),
                                         eqn-form-proof (f-not (f-or (forsome (x,
                                                                               f-not (a)),
                                                                      f-not (forsome (x,
                                                                                      b)))),
                                                         f-not (f-or (f-not (a),
                                                                      f-not (forsome (x,
                                                                                      b)))),
                                                         list (f-iff (f-not (a),
                                                                      forsome (x,
                                                                               f-not (a)))),
                                                         list (iff-commute-proof (forsome (x,
                                                                                           f-not (a)),
                                                                                  f-not (a),
                                                                                  del-forsome-proof (x,
                                                                                                     f-not (a)))))),
                       tautconseq-proof (nil,
                                         f-implies (f-not (f-or (forsome (x,
                                                                          f-not (a)),
                                                                 f-not (forsome (x,
                                                                                 b)))),
                                                    f-not (f-implies (forsome (x,
                                                                               b),
```

```
                                                     forsome (x,
                                                              f-not (a))))),
                                   nil),
                  contra-pos-proof1 (∀ x f-implies (b, f-not (a)),
                                     f-implies (forsome (x, b),
                                                forsome (x, f-not (a))),
                                     forall-over-implies-proof (x,
                                                                b,
                                                                f-not (a))),
                  iff-split-proof1 (f-not (∀ x f-implies (b, f-not (a))),
                                    forsome (x,
                                             f-not (f-implies (b, f-not (a)))),
                                    forall-forsome-proof (x,
                                                          f-implies (b,
                                                                     f-not (a)))),
                  iff-split-proof1 (forsome (x,
                                             f-not (f-implies (b, f-not (a)))),
                                    forsome (x, f-and (a, b)),
                                    eqn-form-proof (forsome (x, f-and (a, b)),
                                                    forsome (x,
                                                             f-not (f-implies (b,
                                                                               f-not (a)))),
                                                    list (f-iff (f-not (f-implies (b,
                                                                                   f-not (a))),
                                                                 f-and (a, b))),
                                                    list (tautconseq-proof (nil,
                                                                            f-iff (f-not (f-implies (b,
                                                                                                     f-not (a))),
                                                                                   f-and (a,
                                                                                          b)),
                                                                            nil))))))
```

```
;;;;;;; If "x" not free in "A", then |- "(A ^ (forsome x B)) -> (forsome x (A ^ B))".
```

THEOREM: forsome-distrib-and-proof-proves
(variable (x)
 ∧ formula (a, *symbols*)
 ∧ formula (b, *symbols*)
 ∧ ($x \notin$ collect-free (a, 0)))
→ proves (forsome-distrib-and-proof(x, a, b),
  f-implies (f-and (a, forsome (x, b)), forsome (x, f-and (a, b))),
  given,
  defns,
  symbols)

EVENT: Enable forsome-distrib-and-proof; name this event 'g0681'.

EVENT: Enable forsome-distrib-or-proof2; name this event 'g0683'.

DEFINITION:

```
forsome-f-if-proof (var, x, y, z)
=   chain-proof (f-if (x, forsome (var, y), forsome (var, z)),
                 forsome (var, f-if (x, y, z)),
                 list (f-or (forsome (var, f-and (x, y)),
                             forsome (var, f-and (f-not (x), z)))),
                 list (tautconseq-proof (list (f-implies (f-and (x,
                                                                 forsome (var, y)),
                                                          forsome (var,
                                                                   f-and (x, y))),
                                               f-implies (f-and (f-not (x),
                                                                 forsome (var, z)),
                                                          forsome (var,
                                                                   f-and (f-not (x),
                                                                          z)))),
                                         f-implies (f-if (x,
                                                          forsome (var, y),
                                                          forsome (var, z)),
                                                    f-or (forsome (var,
                                                                   f-and (x, y)),
                                                          forsome (var,
                                                                   f-and (f-not (x),
                                                                          z)))),
                                         list (forsome-distrib-and-proof (var,
                                                                          x,
                                                                          y),
                                               forsome-distrib-and-proof (var,
                                                                          f-not (x),
                                                                          z))),
                       forsome-distrib-or-proof2 (var,
                                                  f-and (x, y),
                                                  f-and (f-not (x), z))))
```

```
;;;;;;; If "v" not free in "A", then
;;;;;;; |-"(if A (forsome v B) (forsome v C)) -> (forsome v (if A B C))".
;;;;;;;This important rule justifies definition by cases using the "if" connective.
```

THEOREM: forsome-f-if-proof-proves
(variable (var)

$\wedge$ formula (x, *symbols*)
$\wedge$ formula (y, *symbols*)
$\wedge$ formula (z, *symbols*)
$\wedge$ (*var* $\notin$ collect-free (x, 0)))
$\rightarrow$ proves (forsome-f-if-proof (*var*, x, y, z),
f-implies (f-if (x, forsome (*var*, y), forsome (*var*, z)),
forsome (*var*, f-if (x, y, z))),
given,
defns,
symbols)

EVENT: Enable forsome-f-if-proof; name this event 'g0684'.

DEFINITION:
unique-f-if-proof (*test*, *left1*, *left2*, *right1*, *right2*, *exp*, *pf1*, *pf2*)
= tautconseq-proof (list (f-implies (f-and (*left1*, *left2*), *exp*),
f-implies (f-and (*right1*, *right2*), *exp*)),
f-implies (f-and (f-if (*test*, *left1*, *right1*),
f-if (*test*, *left2*, *right2*)),
exp),
list (*pf1*, *pf2*))

```
;;;;;;;If |-"(B1 ^ B2) -> P" and |-"(C1 ^ C2) -> P", then
;;;;;;; |- "(((if A B1 C1) ^ (if A B2 C2)) -> P)".
;;;;;;;This lemma justifies the uniqueness part of the admissibility of a
;;;;;;;definition by cases using the "if" connective.
```

THEOREM: unique-f-if-proof-proves
(formula (*test*, *symbols*)
$\wedge$ proves (*pf1*, f-implies (f-and (*left1*, *left2*), *exp*), *given*, *defns*, *symbols*)
$\wedge$ proves (*pf2*, f-implies (f-and (*right1*, *right2*), *exp*), *given*, *defns*, *symbols*))
$\rightarrow$ proves (unique-f-if-proof (*test*, *left1*, *left2*, *right1*, *right2*, *exp*, *pf1*, *pf2*),
f-implies (f-and (f-if (*test*, *left1*, *right1*),
f-if (*test*, *left2*, *right2*)),
exp),
given,
defns,
symbols)

EVENT: Enable unique-f-if-proof; name this event 'g0603'.

DEFINITION:
forsome-f-if-case-proof (*var*, x, y, z, *pf1*, *pf2*)
= tautconseq-proof (list (f-implies (x, forsome (*var*, y)),

$$\text{f-implies}\,(\text{f-not}\,(x),\ \text{forsome}\,(var,\ z)),$$
$$\text{f-implies}\,(\text{f-if}\,(x,\ \text{forsome}\,(var,\ y),\ \text{forsome}\,(var,\ z)),$$
$$\text{forsome}\,(var,\ \text{f-if}\,(x,\ y,\ z)))),$$
$$\text{forsome}\,(var,\ \text{f-if}\,(x,\ y,\ z)),$$
$$\text{list}\,(pf1,\ pf2,\ \text{forsome-f-if-proof}\,(var,\ x,\ y,\ z)))$$

```
;;;;;;;If |- "(A -> (forsome v B))" and |-"((not A) -> (forsome v C))",
;;;;;;;then |- "(forsome v (if A B C))".
```

THEOREM: forsome-f-if-case-proof-proves
(variable (var)
 ∧ formula (x, $symbols$)
 ∧ formula (y, $symbols$)
 ∧ formula (z, $symbols$)
 ∧ ($var \notin$ collect-free (x, 0))
 ∧ proves ($pf1$, f-implies (x, forsome (var, y)), $given$, $defns$, $symbols$)
 ∧ proves ($pf2$, f-implies (f-not (x), forsome (var, z)), $given$, $defns$, $symbols$))
→ proves (forsome-f-if-case-proof (var, x, y, z, $pf1$, $pf2$),
forsome (var, f-if (x, y, z)),
$given$,
$defns$,
$symbols$)

EVENT: Enable forsome-f-if-case-proof; name this event ‘g0687’.

DEFINITION:
exis-fun-intro-proof (var, $term$)
= detach-proof (f-equal ($term$, $term$),
forsome (var, f-equal (var, $term$)),
ident-axiom-proof ($term$),
subst-thm-a-proof (f-equal (var, $term$), list (var), list ($term$)))

THEOREM: parallel-subst-f-equal
parallel-subst (f-equal (a, b), $vars$, $terms$, 0)
= f-equal (parallel-subst (a, $vars$, $terms$, 0),
parallel-subst (b, $vars$, $terms$, 0))

THEOREM: f-equal-col-free
collect-free (f-equal (x, y), 0) = collect-free (list (x, y), 1)

THEOREM: covering-f-equal
covering (f-equal (x, y), var, 0) = covering (list (x, y), var, 1)

```
;;;;;;; |- "(forsome v (v = t))", "v" not free in "t".
```

THEOREM: exis-fun-intro-proof-proves
(variable (var) $\wedge$ termp ($term$, 0, $symbols$) $\wedge$ ($var \notin$ collect-free ($term$, 0)))
$\rightarrow$ proves (exis-fun-intro-proof (var, $term$),
forsome (var, f-equal (var, $term$)),
$given$,
$defns$,
$symbols$)

EVENT: Enable exis-fun-intro-proof; name this event 'g0690'.

THEOREM: subset-append-reduc
subset (append (x, y), z) = (subset (x, z) $\wedge$ subset (y, z))

DEFINITION:
z-succ-hyps ($defns$, $symbols$)
= ((assoc (fn (3, 1), $defns$) = f-equal (z-succ (0), z-union (z-sing (0), 0)))
$\wedge$ subset (list (fn (2, 2), fn (1, 2)), rest-of (fn (3, 1), $symbols$)))

DEFINITION:
unique-fun-intro-proof ($var1$, $var2$, $term$)
= tautconseq-proof (list (f-implies (f-equal ($var1$, $term$),
f-implies (f-equal ($var2$, $term$),
f-equal ($var1$, $var2$)))),
f-implies (f-and (f-equal ($var1$, $term$),
f-equal ($var2$, $term$)),
f-equal ($var1$, $var2$)),
list (trans-equal-proof2 ($var1$, $term$, $var2$)))

```
;;;;;;; |- "((v1 = t) ^ (v2 = t)) -> (v1 = v2)".
```

THEOREM: unique-fun-intro-proof-proves
termp (list ($var1$, $var2$, $term$), 1, $symbols$)
$\rightarrow$ proves (unique-fun-intro-proof ($var1$, $var2$, $term$),
f-implies (f-and (f-equal ($var1$, $term$), f-equal ($var2$, $term$)),
f-equal ($var1$, $var2$)),
$given$,
$defns$,
$symbols$)

EVENT: Enable unique-fun-intro-proof; name this event 'g0604'.

EVENT: Enable rest-of; name this event 'g0702'.

DEFINITION:

z-succ-defn-proof (*term*)
= subst-rule-proof (f-equal (z-succ (0), z-union (z-sing (0), 0)),
    0,
    term,
    fun-defn-proof (fn (3, 1),
        list (0),
        1,
        2,
        f-equal (1, z-union (z-sing (0), 0)),
        exis-fun-intro-proof (1,
            z-union (z-sing (0),
                0)),
        unique-fun-intro-proof (1,
            2,
            z-union (z-sing (0),
                0))))

```
;;;;;;;The definition of the successor operation: |- "(S t) = ({t} U t)".
```

THEOREM: z-succ-defn
(z-succ-hyps (*defns*, *symbols*)
$\land$ termp (*term*, 0, *symbols*)
$\land$ subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), *symbols*))
$\rightarrow$ proves (z-succ-defn-proof (*term*),
    f-equal (z-succ (*term*), z-union (z-sing (*term*), *term*)),
    given,
    defns,
    symbols)

EVENT: Enable z-succ-defn-proof; name this event 'g0407'.

EVENT: Enable z-succ-hyps; name this event 'g0677'.

THEOREM: termp-z-succ
termp (z-succ (x), 0, *symbols*)
= ((fn (3, 1) $\in$ *symbols*) $\land$ termp (x, 0, *symbols*))

THEOREM: subst-z-succ
subst (z-succ (x), *var*, *term*, 0) = z-succ (subst (x, *var*, *term*, 0))

THEOREM: parallel-subst-z-succ
parallel-subst (z-succ (x), *vars*, *terms*, 0)
= z-succ (parallel-subst (x, *vars*, *terms*, 0))

THEOREM: covering-z-succ
covering (z-succ (x), *var*, 0) = covering (list (x), *var*, 1)

THEOREM: z-succ-col-free
collect-free (z-succ (x), 0) = collect-free (list (x), 1)

THEOREM: equal-axiom1-z-succ
((fn (3, 1) $\in$ *symbols*) $\wedge$ termp (list (x, y), 1, *symbols*))
$\rightarrow$ proves (equal-axiom1-proof (fn (3, 1), list (x), list (y)),
f-implies (f-equal (x, y), f-equal (z-succ (x), z-succ (y))),
given,
defns,
symbols)

EVENT: Enable z-succ; name this event 'g0408'.

DEFINITION:
ext-axiom-eq-detach-proof (*terma*, *termb*, *termc*, *pf*)
= detach-proof (f-equal (*terma*, *termb*),
f-iff (isin (*termc*, *terma*), isin (*termc*, *termb*)),
pf,
chain-proof (f-equal (*terma*, *termb*),
f-iff (isin (*termc*, *terma*), isin (*termc*, *termb*)),
list ($\forall$ *mk-bnd*, (list terma termb), 0 f-iff (isin (mk-bnd (list (*terma*,
termb),
0),
terma),
isin (mk-bnd (list (*terma*,
termb),
0),
termb))),
list (iff-split-proof1 (f-equal (*terma*, *termb*),
$\forall$ *mk-bnd*, (list terma termb), 0 f-iff (isin (mk-bnd (list (*terma*,
termb),
0),
terma),
isin (mk-bnd (list (*terma*,
termb),
0),
termb)),
ext-axiom-proof (mk-bnd (list (*terma*,
termb),
0),
terma,

```
                                              termb)),
                        subst-thm-b-proof (f-iff (isin (mk-bnd (list (terma,
                                                                     termb),
                                                                0),
                                                          terma),
                                                     isin (mk-bnd (list (terma,
                                                                     termb),
                                                                0),
                                                          termb)),
                                                   list (mk-bnd (list (terma,
                                                                     termb),
                                                                0)),
                                                   list (termc)))))
```

```
;;;;;;; If |-"(a = b)", then |-"(c in a) <-> (c in b)".
```

THEOREM: ext-axiom-eq-detach

```
 (termp (list (terma, termb, termc), 1, symbols)
  ∧    proves (pf, f-equal (terma, termb), given, defns, symbols)
  ∧    (p (1, 2) ∈ symbols))
 →    proves (ext-axiom-eq-detach-proof (terma, termb, termc, pf),
               f-iff (isin (termc, terma), isin (termc, termb)),
               given,
               defns,
               symbols)
```

EVENT: Enable ext-axiom-eq-detach-proof; name this event 'g0704'.

DEFINITION:

```
in-z-succ-proof (x, y)
=    chain-iff-proof (isin (y, z-succ (x)),
                      f-or (f-equal (y, x), isin (y, x)),
                      list (isin (y, z-union (z-sing (x), x)),
                            f-or (isin (y, z-sing (x)), isin (y, x))),
                      list (ext-axiom-eq-detach-proof (z-succ (x),
                                                       z-union (z-sing (x), x),
                                                       y,
                                                       z-succ-defn-proof (x)),
                            union-axiom-proof (y, z-sing (x), x),
                            eqn-form-proof (f-or (f-equal (y, x), isin (y, x)),
                                            f-or (isin (y, z-sing (x)), isin (y, x)),
                                            list (f-iff (isin (y, z-sing (x)),
                                                         f-equal (y, x))),
                                            list (isin-sing-proof (x, y)))))
```

```
;;;;;;; |-"(y in (S x)) <-> ((y = x) v (y in x))".
```

THEOREM: in-z-succ-proof-proves

```
 (z-succ-hyps (defns, symbols)
  ∧   subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), symbols)
  ∧   termp (list (x, y), 1, symbols))
 →   proves (in-z-succ-proof (x, y),
              f-iff (isin (y, z-succ (x)), f-or (f-equal (y, x), isin (y, x))),
              given,
              defns,
              symbols)
```

EVENT: Enable in-z-succ-proof; name this event 'g0705'.

DEFINITION:

```
x-in-z-succ-x-proof (x)
=   f-iff-mp-proof2 (isin (x, z-succ (x)),
                     f-or (f-equal (x, x), isin (x, x)),
                     in-z-succ-proof (x, x),
                     rt-expan-proof (f-equal (x, x),
                                     isin (x, x),
                                     ident-axiom-proof (x)))
```

```
;;;;;;; |-"(x in (S x))".
```

THEOREM: x-in-z-succ-x

```
 (z-succ-hyps (defns, symbols)
  ∧   subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), symbols)
  ∧   termp (x, 0, symbols))
 →   proves (x-in-z-succ-x-proof (x), isin (x, z-succ (x)), given, defns, symbols)
```

EVENT: Enable x-in-z-succ-x-proof; name this event 'g0707'.

DEFINITION:

```
z-succ-equal-step-proof (x, y)
=   chain-proof (f-equal (z-succ (x), z-succ (y)),
                 f-or (f-equal (x, y), isin (x, y)),
                 list (isin (x, z-succ (y))),
                 list (ext-axiom-eq-elim1-proof (z-succ (x),
                                                 z-succ (y),
                                                 x,
                                                 x-in-z-succ-x-proof (x)),
                   iff-split-proof1 (isin (x, z-succ (y)),
                                     f-or (f-equal (x, y), isin (x, y)),
                                     in-z-succ-proof (y, x))))
```

```
;;;;;;; |- "((S x)=(S y)) -> ((x = y) v (x in y))".
```

THEOREM: z-succ-equal-step
(z-succ-hyps (*defns*, *symbols*)
∧ subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), *symbols*)
∧ termp (list (x, y), 1, *symbols*))
→ proves (z-succ-equal-step-proof (x, y),
f-implies (f-equal (z-succ (x), z-succ (y)),
f-or (f-equal (x, y), isin (x, y))),
given,
defns,
symbols)

EVENT: Enable z-succ-equal-step-proof; name this event ‘g0708’.

DEFINITION:
equal-z-succ-proof (x, y)
= tautconseq-proof (list (f-implies (f-equal (z-succ (x), z-succ (y)),
f-or (f-equal (x, y), isin (x, y))),
f-implies (f-equal (z-succ (x), z-succ (y)),
f-equal (z-succ (y), z-succ (x))),
f-implies (f-equal (z-succ (y), z-succ (x)),
f-or (f-equal (y, x), isin (y, x))),
f-implies (f-equal (y, x), f-equal (x, y))),
f-implies (f-equal (z-succ (x), z-succ (y)),
f-or (f-equal (x, y),
f-and (isin (x, y), isin (y, x)))),
list (z-succ-equal-step-proof (x, y),
symmetry-proof1 (z-succ (x), z-succ (y)),
z-succ-equal-step-proof (y, x),
symmetry-proof1 (y, x)))

```
;;;;;;; |- "((S x)=(S y)) -> ((x = y) v ((x in y) ^ (y in x)))".
```

THEOREM: equal-z-succ-proof-proves
(z-succ-hyps (*defns*, *symbols*)
∧ subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), *symbols*)
∧ termp (x, 0, *symbols*)
∧ termp (y, 0, *symbols*))
→ proves (equal-z-succ-proof (x, y),
f-implies (f-equal (z-succ (x), z-succ (y)),
f-or (f-equal (x, y), f-and (isin (x, y), isin (y, x)))),
given,
defns,
symbols)

EVENT: Enable equal-z-succ-proof; name this event ‘g0709’.

```
;;;;;;;The next series of defns and lemmas deal with the definition of "(I x)".
;;;;;;;"(I x)" holds of "x" iff "x" is transitive and linearly ordered by "in".
;;;;;;;"(Trans x)" iff "(forall y (forall z ((y in x) -> ((z in y) -> (z in x)))))".
;;;;;;;"x" is linearly ordered by "in" (i.e., "(Trich x)") iff
;;;;;;;"(forall y (forall z (((y in x) ^ (z in x)) -> ((y = z) v ((y in z) v (z in y))))))"
;;;;;;;Z-TRANS generates the transitivity assertion.
```

DEFINITION:
z-trans1 $(x,\ u,\ v)$ = f-implies (f-and (isin $(u,\ x)$, isin $(v,\ u)$), isin $(v,\ x)$)

DEFINITION: z-trans $(x,\ y,\ z) = \forall\ y\ \forall\ z$ z-trans1 $(x,\ y,\ z)$

DEFINITION:
z-trans-instance-proof $(x,\ y,\ z,\ y1,\ z1)$
= subst-thm-b-proof (z-trans1 $(x,\ y,\ z)$, list $(y,\ z)$, list $(y1,\ z1)$)

THEOREM: z-trans-instance-proof-proves
(termp (list $(x,\ y1,\ z1)$, 1, *symbols*)
$\wedge$ var-set (list $(y,\ z)$, 2)
$\wedge$ nil-intersect (list $(y,\ z)$, collect-free $(x,\ 0)$)
$\wedge$ (p (1, 2) $\in$ *symbols*)
$\wedge$ (*concl* = f-implies (z-trans $(x,\ y,\ z)$, z-trans1 $(x,\ y1,\ z1)$)))
$\rightarrow$ proves (z-trans-instance-proof $(x,\ y,\ z,\ y1,\ z1)$, *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-trans-instance-proof; name this event ‘g0710’.

THEOREM: formula-z-trans
formula (z-trans $(x,\ y,\ z)$, *symbols*)
= ((p (1, 2) $\in$ *symbols*) $\wedge$ termp (list $(x,\ y,\ z)$, 1, *symbols*))

THEOREM: collect-free-z-trans
(var-list (list $(y,\ z)$, 2) $\wedge$ nil-intersect (list $(y,\ z)$, collect-free $(x,\ 0)$))
$\rightarrow$ ((*var* $\in$ collect-free (z-trans $(x,\ y,\ z)$, 0))
= (*var* $\in$ collect-free $(x,\ 0)$))

EVENT: Enable z-trans; name this event ‘g0715’.

DEFINITION:
in-z-trans-proof $(x,\ v1,\ v2,\ y,\ z)$
= tautconseq-proof (list (f-implies (z-trans $(x,\ v1,\ v2)$, z-trans1 $(x,\ y,\ z)$),
f-implies (z-trans $(y,\ v1,\ v2)$, z-trans1 $(y,\ x,\ z)$)),
f-implies (f-and (f-and (z-trans $(x,\ v1,\ v2)$,

z-trans (*y*, *v1*, *v2*)),
f-and (isin (*x*, *y*), isin (*y*, *x*))),
f-iff (isin (*z*, *x*), isin (*z*, *y*))),
list (z-trans-instance-proof (*x*, *v1*, *v2*, *y*, *z*),
z-trans-instance-proof (*y*, *v1*, *v2*, *x*, *z*)))

`;;;;;;; |- "((Trans x) ^ (Trans y) ^ (x in y) ^ (y in x)) -> ((z in x) <-> (z in y))".`

THEOREM: in-z-trans-proof-proves
(termp (list (*x*, *y*, *z*), 1, *symbols*)
∧ var-set (list (*v1*, *v2*), 2)
∧ nil-intersect (list (*v1*, *v2*), collect-free (list (*x*, *y*), 1))
∧ (p (1, 2) ∈ *symbols*))
→ proves (in-z-trans-proof (*x*, *v1*, *v2*, *y*, *z*),
f-implies (f-and (f-and (z-trans (*x*, *v1*, *v2*), z-trans (*y*, *v1*, *v2*)),
f-and (isin (*x*, *y*), isin (*y*, *x*))),
f-iff (isin (*z*, *x*), isin (*z*, *y*))),
given,
defns,
symbols)

EVENT: Enable in-z-trans-proof; name this event 'g0718'.

DEFINITION:
z-trans-equal-proof1 (*x*, *y*, *v1*, *v2*)
= chain-proof (f-and (f-and (z-trans (*x*, *v1*, *v2*), z-trans (*y*, *v1*, *v2*)),
f-and (isin (*x*, *y*), isin (*y*, *x*))),
f-equal (*x*, *y*),
list (∀ *v1* f-iff (isin (*v1*, *x*), isin (*v1*, *y*))),
list (forall-intro-proof (f-and (f-and (z-trans (*x*, *v1*, *v2*),
z-trans (*y*, *v1*, *v2*)),
f-and (isin (*x*, *y*), isin (*y*, *x*))),
f-iff (isin (*v1*, *x*), isin (*v1*, *y*)),
v1,
in-z-trans-proof (*x*, *v1*, *v2*, *y*, *v1*)),
iff-split-proof2 (f-equal (*x*, *y*),
∀ *v1* f-iff (isin (*v1*, *x*), isin (*v1*, *y*)),
ext-axiom-proof (*v1*, *x*, *y*))))

`;;;;;;; |- "((Trans x) ^ (Trans y) ^ (x in y) ^ (y in x)) -> (x = y)".`

THEOREM: z-trans-equal-proof1-proves
(termp (list (*x*, *y*), 1, *symbols*)
∧ var-set (list (*v1*, *v2*), 2)

```
 ∧   nil-intersect (list (v1, v2), collect-free (list (x, y), 1))
 ∧   (p (1, 2) ∈ symbols))
→    proves (z-trans-equal-proof1 (x, y, v1, v2),
             f-implies (f-and (f-and (z-trans (x, v1, v2), z-trans (y, v1, v2)),
                                f-and (isin (x, y), isin (y, x))),
                        f-equal (x, y)),
             given,
             defns,
             symbols)
```

Event: Enable z-trans-equal-proof1; name this event 'g0719'.

Definition:

```
z-succ-equal-proof (x, y, v1, v2)
=    tautconseq-proof (list (f-implies (f-equal (z-succ (x), z-succ (y)),
                                        f-or (f-equal (x, y),
                                              f-and (isin (x, y), isin (y, x)))),
                              f-implies (f-and (f-and (z-trans (x, v1, v2),
                                                       z-trans (y, v1, v2)),
                                                f-and (isin (x, y), isin (y, x))),
                                         f-equal (x, y))),
                        f-implies (z-trans (x, v1, v2),
                                   f-implies (z-trans (y, v1, v2),
                                              f-implies (f-equal (z-succ (x),
                                                                  z-succ (y)),
                                                         f-equal (x, y)))),
                        list (equal-z-succ-proof (x, y),
                              z-trans-equal-proof1 (x, y, v1, v2)))
```

```
;;;;;;; |- "((Trans x) -> ((Trans y) -> (((S x)=(S y)) -> (x = y))))".
```

Theorem: z-succ-equal-proof-proves

```
 (termp (list (x, y), 1, symbols)
 ∧   var-set (list (v1, v2), 2)
 ∧   nil-intersect (list (v1, v2), collect-free (list (x, y), 1))
 ∧   subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), symbols)
 ∧   z-succ-hyps (defns, symbols))
→    proves (z-succ-equal-proof (x, y, v1, v2),
             f-implies (z-trans (x, v1, v2),
                        f-implies (z-trans (y, v1, v2),
                                   f-implies (f-equal (z-succ (x), z-succ (y)),
                                              f-equal (x, y)))),
             given,
             defns,
             symbols)
```

EVENT: Enable z-succ-equal-proof; name this event ‘g0720’.

EVENT: Enable z-trans-equal-proof1-proves; name this event ‘g0721’.

EVENT: Enable in-z-trans-proof-proves; name this event ‘g0722’.

```
;;;;;;;NUMERAL returns the set-theoretic ordinal corresponding to a Lisp natural number.
;;;;;;;The numeral for X will be represented as "|X|".
```

DEFINITION:
numeral (x)
= **if** $x \simeq$ `0` **then** PHI
else z-succ (numeral ($x - 1$)) **endif**

THEOREM: termp-numeral
((fn (`0`, `0`) $\in$ *symbols*) $\wedge$ (fn (`3`, `1`) $\in$ *symbols*))
$\rightarrow$ termp (numeral (x), `0`, *symbols*)

THEOREM: collect-free-numeral
collect-free (numeral (x), `0`) = **nil**

DEFINITION:
chain-or-proof (a, b, c, d, *pf1*, *pf2*, *pf3*)
= tautconseq-proof (list (f-implies (a, f-or (b, c)),
f-implies (b, d),
f-implies (c, d)),
f-implies (a, d),
list (*pf1*, *pf2*, *pf3*))

```
;;;;;;; If |- "(A -> (B v C))",  |- "(B -> D)", and |- "(C -> D)", then |- "(A -> D)".
```

THEOREM: chain-or-proof-proves
(proves (*pf1*, f-implies (a, f-or (b, c)), *given*, *defns*, *symbols*)
$\wedge$ proves (*pf2*, f-implies (b, d), *given*, *defns*, *symbols*)
$\wedge$ proves (*pf3*, f-implies (c, d), *given*, *defns*, *symbols*))
$\rightarrow$ proves (chain-or-proof (a, b, c, d, *pf1*, *pf2*, *pf3*),
f-implies (a, d),
given,
defns,
symbols)

EVENT: Enable chain-or-proof; name this event ‘g0605’.

DEFINITION:

```
z-trans-numeral-proof(x, y, z)
=   if x ≃ 0
    then tautconseq-proof (list (f-not (isin (y, PHI))),
                                 f-implies (f-and (isin (y, PHI), isin (z, y)),
                                            isin (z, PHI)),
                           list (null-axiom-proof (y)))
    else chain-proof (f-and (isin (y, numeral (x)), isin (z, y)),
                      isin (z, numeral (x)),
                      list (f-and (f-or (f-equal (y, numeral (x - 1)),
                                         isin (y, numeral (x - 1))),
                                   isin (z, y)),
                            isin (z, numeral (x - 1))),
                      list (tautconseq-proof (list (f-implies (isin (y,
                                                                     z-succ (numeral (x - 1))),
                                                               f-or (f-equal (y,
                                                                              numeral (x - 1)),
                                                                     isin (y,
                                                                           numeral (x - 1))))),
                                                    f-implies (f-and (isin (y,
                                                                            z-succ (numeral (x - 1))),
                                                                      isin (z, y)),
                                                               f-and (f-or (f-equal (y,
                                                                                     numeral (x - 1)),
                                                                            isin (y,
                                                                                  numeral (x - 1))),
                                                                      isin (z, y)))),
                                              list (iff-split-proof1 (isin (y,
                                                                            z-succ (numeral (x - 1))),
                                                                      f-or (f-equal (y,
                                                                                     numeral (x - 1)),
                                                                            isin (y,
                                                                                  numeral (x - 1))),
                                                                      in-z-succ-proof (numeral (x - 1),
                                                                                       y)))),
                            tautconseq-proof (list (f-implies (f-equal (y,
                                                                        numeral (x - 1)),
                                                               f-implies (isin (z,
                                                                                y),
                                                                          isin (z,
                                                                                numeral (x - 1)))),
                                                    f-implies (f-and (isin (y,
                                                                            numeral (x - 1)),
                                                                      isin (z,
```

```
                                          y)),
                               isin (z,
                                     numeral (x − 1)))),
                 f-implies (f-and (f-or (f-equal (y,
                                                  numeral (x − 1)),
                                         isin (y,
                                               numeral (x − 1))),
                                   isin (z, y)),
                            isin (z,
                                  numeral (x − 1))),
                 list (detach-proof (f-equal (z,
                                              z),
                                     f-implies (f-equal (y,
                                                         numeral (x − 1)),
                                                f-implies (isin (z,
                                                                 y),
                                                           isin (z,
                                                                 numeral (x − 1)))),
                                     ident-axiom-proof (z),
                                     equal-axiom2-proof (p (1,
                                                            2),
                                                         list (z,
                                                               y),
                                                         list (z,
                                                               numeral (x − 1)))),
                      z-trans-numeral-proof (x − 1,
                                             y,
                                             z))),
tautconseq-proof (list (f-implies (f-or (f-equal (z,
                                                  numeral (x − 1)),
                                         isin (z,
                                               numeral (x − 1))),
                                   isin (z,
                                         z-succ (numeral (x − 1))))),
                        f-implies (isin (z,
                                         numeral (x − 1)),
                                   isin (z,
                                         z-succ (numeral (x − 1)))),
                        list (iff-split-proof2 (isin (z,
                                                      z-succ (numeral (x − 1))),
                                                f-or (f-equal (z,
                                                               numeral (x − 1)),
                                                      isin (z,
                                                            numeral (x − 1))),
```

in-z-succ-proof (numeral ($x - 1$),
z)))))) **endif**

```
;;;;;;; |- "((y in |X|) ^ (z in y)) -> (z in |X|)".
```

THEOREM: z-trans-numeral-proof-proves
(z-succ-hyps (*defns*, *symbols*)
∧ subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), *symbols*)
∧ termp (list (y, z), 1, *symbols*))
→ proves (z-trans-numeral-proof (x, y, z),
f-implies (f-and (isin (y, numeral (x)), isin (z, y)),
isin (z, numeral (x))),
given,
defns,
symbols)

EVENT: Enable z-trans-numeral-proof; name this event 'g0723'.

THEOREM: form-equal-proof-proves1
(formula (*forma*, *symbols*)
∧ formula (*formb*, *symbols*)
∧ form-equal (*forma*, *formb*, *list*)
∧ listp (*list*)
∧ form-list (*list*, *symbols*)
∧ (*concl* = f-implies (list-f-and (*list*), f-iff (*forma*, *formb*))))
→ proves (form-equal-proof (*forma*, *formb*, *list*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable form-equal-proof-proves; name this event 'g0724'.

THEOREM: termp-del-pairs
termp (x, 1, *symbols*) → termp (del-pairs (y, *vars*, x), 1, *symbols*)

THEOREM: length-parallel-subst
length (parallel-subst (*exp*, *vars*, *terms*, 1)) = length (*exp*)

THEOREM: termp-parallel-subst
(termp (*term*, *flg*, *symbols*) ∧ termp (*terms*, 1, *symbols*))
→ termp (parallel-subst (*term*, *vars*, *terms*, *flg*), *flg*, *symbols*)

THEOREM: formula-parallel-subst
(formula (*exp*, *symbols*) ∧ termp (*terms*, 1, *symbols*))
→ formula (parallel-subst (*exp*, *vars*, *terms*, 0), *symbols*)

THEOREM: form-equal-parallel-subst-again
(var-list (*vars*, length (*vars*))
 ∧ formula (*exp*, *symbols*)
 ∧ (collect-free (*terms*, `1`) = **nil**)
 ∧ (*list* = list-f-equal (*vars*, *terms*)))
→ form-equal (*exp*, parallel-subst (*exp*, *vars*, *terms*, `0`), *list*)

EVENT: Enable form-equal-parallel-subst; name this event 'g0729'.

EVENT: Enable form-equal; name this event 'g0730'.

DEFINITION:
in-z-succ-proof1 (*x*, *y*)
= iff-split-proof1 (isin (*y*, z-succ (*x*)),
 f-or (f-equal (*y*, *x*), isin (*y*, *x*)),
 in-z-succ-proof (*x*, *y*))

DEFINITION:
z-trich1 (*x*, *y*, *z*)
= f-implies (f-and (isin (*y*, *x*), isin (*z*, *x*)),
 f-or (f-equal (*y*, *z*), f-or (isin (*y*, *z*), isin (*z*, *y*))))

DEFINITION:
ext-case1-proof (*terma*, *termb*, *termc*, *x*)
= chain-proof (f-equal (*terma*, *termb*),
 f-iff (isin (*termc*, *terma*), isin (*termc*, *termb*)),
 list (∀ *x* f-iff (isin (*x*, *terma*), isin (*x*, *termb*))),
 list (iff-split-proof1 (f-equal (*terma*, *termb*),
 ∀ *x* f-iff (isin (*x*, *terma*),
 isin (*x*, *termb*)),
 ext-axiom-proof (*x*, *terma*, *termb*)),
 subst-thm-b-proof (f-iff (isin (*x*, *terma*), isin (*x*, *termb*)),
 list (*x*),
 list (*termc*))))

```
;;;;;;; |- "(a=b) -> ((c in a) <-> (c in b))".
```

THEOREM: ext-case1
(variable (*x*)
 ∧ ($x \notin$ collect-free (*terma*, `0`))
 ∧ ($x \notin$ collect-free (*termb*, `0`))
 ∧ formula (f-implies (f-equal (*terma*, *termb*),
 f-iff (isin (*termc*, *terma*), isin (*termc*, *termb*))),
 symbols))

$\rightarrow$ proves (ext-case1-proof (*terma*, *termb*, *termc*, x),
f-implies (f-equal (*terma*, *termb*),
f-iff (isin (*termc*, *terma*), isin (*termc*, *termb*))),
given,
defns,
symbols)

EVENT: Enable ext-case1-proof; name this event 'g0914'.

DEFINITION:
ext-case-proof (*terma*, *termb*, *termc*)
= ext-case1-proof (*terma*, *termb*, *termc*, mk-bnd (list (*terma*, *termb*), `0`))

THEOREM: ext-axiom-case
formula (f-implies (f-equal (*terma*, *termb*),
f-iff (isin (*termc*, *terma*), isin (*termc*, *termb*))),
symbols)
$\rightarrow$ proves (ext-case-proof (*terma*, *termb*, *termc*),
f-implies (f-equal (*terma*, *termb*),
f-iff (isin (*termc*, *terma*), isin (*termc*, *termb*))),
given,
defns,
symbols)

EVENT: Enable ext-case-proof; name this event 'g0915'.

DEFINITION:
ext-subcase-proof2 (x, y, z)
= tautconseq-proof (list (f-implies (f-equal (x, y),
f-iff (isin (z, x), isin (z, y)))),
f-implies (f-equal (x, y),
f-implies (isin (z, y), isin (z, x))),
list (ext-case-proof (x, y, z)))

```
;;;;;;; |-"(a=b) -> ((c in a) -> (c in b))".
```

THEOREM: ext-subcase-proof2-proves
((p (`1`, `2`) $\in$ *symbols*) $\wedge$ termp (list (x, y, z), `1`, *symbols*))
$\rightarrow$ proves (ext-subcase-proof2 (x, y, z),
f-implies (f-equal (x, y), f-implies (isin (z, y), isin (z, x))),
given,
defns,
symbols)

EVENT: Enable ext-subcase-proof2; name this event ‘g0737’.

DEFINITION:

```
z-trich-numeral-proof (x, y, z)
=   if x ≃ 0
    then tautconseq-proof (list (f-not (isin (y, PHI))),
                               z-trich1 (PHI, y, z),
                               list (null-axiom-proof (y)))
    else chain-or-proof (f-and (isin (y, z-succ (numeral (x − 1))),
                                 isin (z, z-succ (numeral (x − 1)))),
                          f-and (f-equal (y, numeral (x − 1)),
                                 f-equal (z, numeral (x − 1))),
                          f-or (f-and (f-equal (y, numeral (x − 1)),
                                       isin (z, numeral (x − 1))),
                                f-or (f-and (f-equal (z, numeral (x − 1)),
                                             isin (y, numeral (x − 1))),
                                      f-and (isin (y, numeral (x − 1)),
                                             isin (z, numeral (x − 1))))),
                          f-or (f-equal (y, z), f-or (isin (y, z), isin (z, y))),
                          tautconseq-proof (list (f-implies (isin (y,
                                                                   z-succ (numeral (x − 1))),
                                                             f-or (f-equal (y,
                                                                            numeral (x − 1)),
                                                                   isin (y,
                                                                         numeral (x − 1)))),
                                                  f-implies (isin (z,
                                                                   z-succ (numeral (x − 1))),
                                                             f-or (f-equal (z,
                                                                            numeral (x − 1)),
                                                                   isin (z,
                                                                         numeral (x − 1)))),
                                                  f-implies (f-and (isin (y,
                                                                          z-succ (numeral (x − 1))),
                                                                    isin (z,
                                                                          z-succ (numeral (x − 1)))),
                                                             f-or (f-and (f-equal (y,
                                                                                   numeral (x − 1)),
                                                                          f-equal (z,
                                                                                   numeral (x − 1))),
                                                                   f-or (f-and (f-equal (y,
                                                                                         numeral (x − 1)),
                                                                                isin (z,
                                                                                      numeral (x − 1))),
```

```
                                    f-or (f-and (f-equal (z,
                                                          numeral (x - 1)),
                                                 isin (y,
                                                       numeral (x - 1))),
                                         f-and (isin (y,
                                                      numeral (x - 1)),
                                                 isin (z,
                                                       numeral (x - 1))))))),
                 list (in-z-succ-proof1 (numeral (x - 1),
                                         y),
                     in-z-succ-proof1 (numeral (x - 1),
                                       z))),
tautconseq-proof (list (f-implies (f-equal (y,
                                           numeral (x - 1)),
                               f-implies (f-equal (z,
                                                   numeral (x - 1)),
                                          f-equal (y,
                                                   z)))),
                 f-implies (f-and (f-equal (y,
                                            numeral (x - 1)),
                                   f-equal (z,
                                            numeral (x - 1))),
                            f-or (f-equal (y, z),
                                f-or (isin (y,
                                            z),
                                      isin (z,
                                            y))))),
                 list (trans-equal-proof2 (y,
                                           numeral (x - 1),
                                           z))),
tautconseq-proof (list (f-implies (f-equal (y,
                                           numeral (x - 1)),
                               f-implies (isin (z,
                                                numeral (x - 1)),
                                          isin (z,
                                                y))),
                     f-implies (f-equal (z,
                                         numeral (x - 1)),
                               f-implies (isin (y,
                                                numeral (x - 1)),
                                          isin (y,
                                                z))),
                     f-implies (f-and (isin (y,
                                             numeral (x - 1)),
```

```
isin (z,
      numeral (x − 1))),
f-or (f-equal (y,
               z),
      f-or (isin (y,
                  z),
            isin (z,
                  y))))),
f-implies (f-or (f-and (f-equal (y,
                                 numeral (x − 1)),
                        isin (z,
                              numeral (x − 1))),
                 f-or (f-and (f-equal (z,
                                       numeral (x − 1)),
                              isin (y,
                                    numeral (x − 1))),
                       f-and (isin (y,
                                    numeral (x − 1)),
                              isin (z,
                                    numeral (x − 1))))),
           f-or (f-equal (y, z),
                 f-or (isin (y,
                             z),
                       isin (z,
                             y)))),
list (ext-subcase-proof2 (y,
                          numeral (x − 1),
                          z),
      ext-subcase-proof2 (z,
                          numeral (x − 1),
                          y),
      z-trich-numeral-proof (x − 1,
                             y,
                             z)))) endif
```

`;;;;;;; |- "((y in |X|) ^ (z in |X|)) -> ((y = z) ^ ((y in z) v (z in y)))".`

THEOREM: z-trich-numeral-proof-proves
(termp (list (y, z), 1, *symbols*)
$\wedge$ subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), *symbols*)
$\wedge$ z-succ-hyps (*defns*, *symbols*))
$\rightarrow$ proves (z-trich-numeral-proof (x, y, z),
f-implies (f-and (isin (y, numeral (x)), isin (z, numeral (x))),
f-or (f-equal (y, z), f-or (isin (y, z), isin (z, y)))),

```
          given,
          defns,
          symbols)
```

EVENT: Enable z-trich-numeral-proof; name this event ‘g0738’.

DEFINITION:

```
ind-axiom-case-proof (form, x, pfbase, pfind)
=   tautconseq-proof (list (subst (form, x, PHI, 0),
                                ∀ x f-implies (f-and (z-int (x), form),
                                                  subst (form, x, z-succ (x), 0)),
                                ind-axiom (form, x),
                                f-implies (∀ x f-implies (z-int (x), form),
                                              f-implies (z-int (x), form))),
                             f-implies (z-int (x), form),
                             list (pfbase,
                                generalise-proof (x,
                                                     f-implies (f-and (z-int (x), form),
                                                                  subst (form,
                                                                          x,
                                                                          z-succ (x),
                                                                          0)),
                                                     pfind),
                                ind-axiom-proof (form, x),
                                all-elim-proof (x, f-implies (z-int (x), form))))
```

```
;;;;;;; If |- "A[|0|]" and |- "((I x) ^ A[x]) -> A[(S x)]", then
;;;;;;; |- "(I x) -> A[x]".
```

THEOREM: ind-axiom-case

```
 (variable (x)
  ∧   subset (list (fn (3, 1), fn (0, 0)), symbols)
  ∧   formula (f-implies (z-int (x), form), symbols)
  ∧   proves (pfbase, subst (form, x, PHI, 0), given, defns, symbols)
  ∧   proves (pfind,
              f-implies (f-and (z-int (x), form), subst (form, x, z-succ (x), 0)),
              given,
              defns,
              symbols))
 →   proves (ind-axiom-case-proof (form, x, pfbase, pfind),
              f-implies (z-int (x), form),
              given,
              defns,
              symbols)
```

EVENT: Enable ind-axiom-case-proof; name this event ‘g0422’.

THEOREM: atomp-z-int
atomp (z-int (x), *symbols*) = ((p (2, 1) $\in$ *symbols*) $\wedge$ termp (x, 0, *symbols*))

THEOREM: formula-z-int
formula (z-int (x), *symbols*)
= ((p (2, 1) $\in$ *symbols*) $\wedge$ termp (x, 0, *symbols*))

DEFINITION:
phi-in-int-proof (x)
= ind-axiom-case-proof (f-or (f-equal (PHI, x), isin (PHI, x)),
x,
rt-expan-proof (f-equal (PHI, PHI),
isin (PHI, PHI),
ident-axiom-proof (PHI)),
tautconseq-proof (list (f-implies (f-or (f-equal (PHI,
x),
isin (PHI, x)),
isin (PHI,
z-succ (x)))),
f-implies (f-and (z-int (x),
f-or (f-equal (PHI,
x),
isin (PHI, x))),
f-or (f-equal (PHI,
z-succ (x)),
isin (PHI,
z-succ (x)))),
list (iff-split-proof2 (isin (PHI,
z-succ (x)),
f-or (f-equal (PHI,
x),
isin (PHI,
x)),
in-z-succ-proof (x,
PHI)))))

```
;;;;;;; |- "(I x) -> ((|0| = x) v (|0| in x))".  This is the only proof
;;;;;;;in this sequence of events in which the induction axiom schema of Z2 is employed.
```

THEOREM: phi-in-int-proof-proves
(variable (x)
$\wedge$ z-succ-hyps (*defns*, *symbols*)

$\land$ subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), *symbols*)
$\land$ (p (2, 1) $\in$ *symbols*)
$\land$ (*concl* = f-implies (z-int (x), f-or (f-equal (PHI, x), isin (PHI, x)))))
$\rightarrow$ proves (phi-in-int-proof (x), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable phi-in-int-proof; name this event 'g0426'.

DEFINITION:
ext-not-equal-proof (x, y, d, *pfx*, *pfy*)
= tautconseq-proof (list (f-implies (f-equal (x, y), isin (d, x)),
f-not (isin (d, x))),
f-not (f-equal (x, y)),
list (ext-axiom-eq-elim2-proof (x, y, d, *pfy*), *pfx*))

```
;;;;;;; If |-"(not (d in x))" and |- "(d in y)", then |-"(not (x = y))".
```

THEOREM: ext-not-equal-proof-proves
(termp (x, 0, *symbols*)
$\land$ termp (y, 0, *symbols*)
$\land$ termp (d, 0, *symbols*)
$\land$ (p (1, 2) $\in$ *symbols*)
$\land$ proves (*pfx*, f-not (isin (d, x)), *given*, *defns*, *symbols*)
$\land$ proves (*pfy*, isin (d, y), *given*, *defns*, *symbols*))
$\rightarrow$ proves (ext-not-equal-proof (x, y, d, *pfx*, *pfy*),
f-not (f-equal (x, y)),
given,
defns,
symbols)

EVENT: Enable ext-not-equal-proof; name this event 'g0427'.

EVENT: Disable z-opair; name this event 'g0428'.

DEFINITION:
phi-not-in-opair-proof (u, v)
= and-join-proof (f-not (f-equal (PHI, z-opair (u, v))),
f-not (isin (PHI, z-opair (u, v))),
ext-not-equal-proof (PHI,
z-opair (u, v),
z-sing (u),
null-axiom-proof (z-sing (u)),
isin-pair1-proof (z-sing (u),
z-pair (u, v))),

```
tautconseq-proof (list (f-implies (isin (PHI, z-opair (u, v)),
                                   f-or (f-equal (PHI,
                                                  z-sing (u)),
                                         f-equal (PHI,
                                                  z-pair (u,
                                                          v)))),
                       f-not (f-equal (PHI, z-sing (u))),
                       f-not (f-equal (PHI, z-pair (u, v)))),
                  f-not (isin (PHI, z-opair (u, v))),
                  list (iff-split-proof1 (isin (PHI,
                                                z-opair (u,
                                                         v)),
                                          f-or (f-equal (PHI,
                                                         z-sing (u)),
                                                f-equal (PHI,
                                                         z-pair (u,
                                                                 v))),
                                          pair-axiom-proof (PHI,
                                                            z-sing (u),
                                                            z-pair (u,
                                                                    v))),
                       ext-not-equal-proof (PHI,
                                            z-sing (u),
                                            u,
                                            null-axiom-proof (u),
                                            isin-sing1-proof (u)),
                       ext-not-equal-proof (PHI,
                                            z-pair (u, v),
                                            u,
                                            null-axiom-proof (u),
                                            isin-pair1-proof (u,
                                                              v)))))
```

`;;;;;;; |- "((not (|0| in <u, v>)) ^ (not (|0| in <u, v>)))".`

THEOREM: phi-not-in-opair-proof-proves
(subset (list (fn (0, 0), p (1, 2), fn (1, 2)), *symbols*)
 $\wedge$ termp (u, 0, *symbols*)
 $\wedge$ termp (v, 0, *symbols*)
 $\wedge$ (*concl* = f-and (f-not (f-equal (PHI, z-opair (u, v))),
 f-not (isin (PHI, z-opair (u, v))))))
$\rightarrow$ proves (phi-not-in-opair-proof (u, v), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable phi-not-in-opair-proof; name this event ‘g0429’.

EVENT: Enable z-opair; name this event ‘g0430’.

DEFINITION:
not-int-opair-proof $(u,\ v,\ x)$
$=$ tautconseq-proof (list (f-implies (z-int (z-opair $(u,\ v)$),
f-or (f-equal (PHI, z-opair $(u,\ v)$),
isin (PHI, z-opair $(u,\ v)$))),
f-and (f-not (f-equal (PHI, z-opair $(u,\ v)$)),
f-not (isin (PHI, z-opair $(u,\ v)$)))),
f-not (z-int (z-opair $(u,\ v)$)),
list (subst-rule-proof (f-implies (z-int (x),
f-or (f-equal (PHI, x),
isin (PHI, x))),
x,
z-opair $(u,\ v)$,
phi-in-int-proof (x)),
phi-not-in-opair-proof $(u,\ v)$))

```
;;;;;;; |- "(not (I <u v>))".
```

THEOREM: not-int-opair-proof-proves
(variable (x)
$\land$ z-succ-hyps (*defns*, *symbols*)
$\land$ subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), *symbols*)
$\land$ (p (2, 1) $\in$ *symbols*)
$\land$ termp (u, 0, *symbols*)
$\land$ termp (v, 0, *symbols*))
$\rightarrow$ proves (not-int-opair-proof $(u,\ v,\ x)$,
f-not (z-int (z-opair $(u,\ v)$)),
given,
defns,
symbols)

EVENT: Enable not-int-opair-proof; name this event ‘g0868’.

DEFINITION: z-trich (x, *v1*, *v2*) $=$ $\forall$ *v1* $\forall$ *v2* z-trich1 (x, *v1*, *v2*)

DEFINITION:
z-int-defn $(x,\ y,\ z)$ $=$ f-and (z-trans $(x,\ y,\ z)$, z-trich $(x,\ y,\ z)$)

EVENT: Enable pred-defn-proof; name this event ‘g0609’.

THEOREM: subst-f-iff
subst (f-iff $(x,\ y)$, *var*, *term*, 0)
$=$ f-iff (subst (x, *var*, *term*, 0), subst (y, *var*, *term*, 0))

EVENT: Disable z-trans; name this event ‘g0872’.

THEOREM: subset-del
variable (x) → (subset (del (x, y), z) = subset (y, cons (x, z)))

DEFINITION:
z-int-defn-proof (*term*)
= subst-rule-proof (f-iff (z-int (0), z-int-defn (0, 1, 2)),
0,
term,
pred-defn-proof (p (2, 1), list (0), z-int-defn (0, 1, 2)))

EVENT: Enable not-member-varlist; name this event ‘g0613’.

DEFINITION:
z-int-hyps (*defns*, *symbols*)
= ((p (1, 2) ∈ rest-of (p (2, 1), *symbols*))
∧ (assoc (p (2, 1), *defns*) = z-int-defn (0, 1, 2)))

THEOREM: z-int-defn-proof-proves
(termp (*term*, 0, *symbols*)
∧ subset (list (p (1, 2), p (2, 1)), *symbols*)
∧ z-int-hyps (*defns*, *symbols*)
∧ nil-intersect (list (1, 2), collect-free (*term*, 0)))
→ proves (z-int-defn-proof (*term*),
f-iff (z-int (*term*), f-and (z-trans (*term*, 1, 2), z-trich (*term*, 1, 2))),
given,
defns,
symbols)

EVENT: Enable z-int-defn-proof; name this event ‘g0639’.

THEOREM: covering-numeral
covering (numeral (x), *var*, 0) = **nil**

DEFINITION:
z-int-numeral-proof (x)
= f-iff-mp-proof2 (z-int (numeral (x)),
f-and (z-trans (numeral (x), 1, 2),
z-trich (numeral (x), 1, 2)),
z-int-defn-proof (numeral (x)),
and-join-proof (z-trans (numeral (x), 1, 2),
z-trich (numeral (x), 1, 2),

```
generalise-proof(1,
                 ∀ 2 z-trans1(numeral(x),
                              1,
                              2),
                 generalise-proof(2,
                                  z-trans1(numeral(x),
                                           1,
                                           2),
                                  z-trans-numeral-proof(x,
                                                        1,
                                                        2))),
generalise-proof(1,
                 ∀ 2 z-trich1(numeral(x),
                              1,
                              2),
                 generalise-proof(2,
                                  z-trich1(numeral(x),
                                           1,
                                           2),
                                  z-trich-numeral-proof(x,
                                                        1,
                                                        2)))))
```

EVENT: Enable z-int-defn; name this event 'g0744'.

EVENT: Enable z-int-hyps; name this event 'g0614'.

```
;;;;;;; |- "(I |X|)".  At this point we are overloading our notation.  Though |X|
;;;;;;;appears in quotes, it denotes the numeral which is the result of
;;;;;;;evaluating |X| for the given X.  So the metatheorem is that "(I |X|)" is
;;;;;;;a Z2 theorem for any X.
```

THEOREM: z-int-numeral-proof-proves
(z-succ-hyps (*defns*, *symbols*)
$\land$ subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (2, 1), p (1, 2), fn (0, 0)), *symbols*)
$\land$ z-int-hyps (*defns*, *symbols*))
$\rightarrow$ proves (z-int-numeral-proof(x), z-int (numeral (x)), *given*, *defns*, *symbols*)

EVENT: Enable z-int-numeral-proof; name this event 'g0906'.

THEOREM: covering-z-int
covering (z-int (x), *var*, 0) = covering (list (x), *var*, 1)

THEOREM: collect-z-int
collect-free (z-int (x), 0) = collect-free (list (x), 1)

THEOREM: subst-z-int
subst (z-int (x), var, $term$, 0) = z-int (subst (x, var, $term$, 0))

THEOREM: parallel-subst-z-int
parallel-subst (z-int (x), var, $term$, 0)
= z-int (parallel-subst (x, var, $term$, 0))

EVENT: Enable z-int; name this event 'g0907'.

DEFINITION: z-car (x) = list (fn (4, 1), x)

DEFINITION: z-cdr (x) = list (fn (5, 1), x)

EVENT: Disable z-opair; name this event 'g0628'.

THEOREM: covering-z-opair
covering (z-opair (x, y), var, 0)
= covering (list (z-sing (x), z-pair (x, y)), var, 1)

THEOREM: subst-z-opair
subst (z-opair (x, y), var, $term$, 0)
= z-opair (subst (x, var, $term$, 0), subst (y, var, $term$, 0))

THEOREM: parallel-subst-z-opair
parallel-subst (z-opair (x, y), $vars$, $terms$, 0)
= z-opair (parallel-subst (x, $vars$, $terms$, 0),
parallel-subst (y, $vars$, $terms$, 0))

EVENT: Enable z-opair; name this event 'g0608'.

DEFINITION:
z-car-unique-step1-proof (x, y, $y1$, z, $z1$)
= tautconseq-proof (list (f-implies (f-equal (x, z-opair (y, z)),
f-implies (f-equal (x, z-opair ($y1$, $z1$)),
f-equal (z-opair (y, z),
z-opair ($y1$, $z1$)))),
f-implies (f-equal (z-opair (y, z), z-opair ($y1$, $z1$)),
f-equal (y, $y1$))),
f-implies (f-equal (x, z-opair (y, z)),
f-implies (f-equal (x, z-opair ($y1$, $z1$)),
f-equal (y, $y1$))),

```
                  list (trans-equal-proof1 (z-opair (y, z),
                                            x,
                                            z-opair (y1, z1)),
                     z-car-unique-proof (y, z, y1, z1)))
```

```
;;;;;;; |- "(x = <y, z>) -> ((x = <y1, z1>) -> (y = y1))".
```

Theorem: z-car-unique-step1-proof-proves

```
 (termp (list (x, y, y1, z, z1), 1, symbols)
  ∧   subset (list (p (1, 2), fn (1, 2)), symbols))
 →    proves (z-car-unique-step1-proof (x, y, y1, z, z1),
           f-implies (f-equal (x, z-opair (y, z)),
                   f-implies (f-equal (x, z-opair (y1, z1)), f-equal (y, y1))),
           given,
           defns,
           symbols)
```

Event: Enable z-car-unique-step1-proof; name this event ‘g0630’.

Definition:

```
z-car-unique-step2-proof (x, y, y1, z, z1)
=   forsome-intro-proof (z,
                  f-equal (x, z-opair (y, z)),
                  f-implies (forsome (z1, f-equal (x, z-opair (y1, z1))),
                         f-equal (y, y1)),
                  chain-proof (f-equal (x, z-opair (y, z)),
                         f-implies (forsome (z1,
                                         f-equal (x,
                                               z-opair (y1,
                                                     z1))),
                              f-equal (y, y1)),
                         list (∀ z1 f-implies (f-equal (x,
                                               z-opair (y1,
                                                     z1)),
                                         f-equal (y, y1)),
                            f-implies (forsome (z1,
                                           f-equal (x,
                                                 z-opair (y1,
                                                       z1))),
                                 forsome (z1,
                                        f-equal (y, y1)))),
                         list (forall-intro-proof (f-equal (x,
                                                  z-opair (y,
                                                        z)),
```

```
                      f-implies (f-equal (x,
                                          z-opair (y1,
                                                   z1)),
                                 f-equal (y,
                                          y1)),
                      z1,
                      z-car-unique-step1-proof (x,
                                                y,
                                                y1,
                                                z,
                                                z1)),
forall-over-implies-proof (z1,
                           f-equal (x,
                                    z-opair (y1,
                                             z1)),
                           f-equal (y,
                                    y1)),
tautconseq-proof (list (f-implies (forsome (z1,
                                            f-equal (y,
                                                     y1)),
                                   f-equal (y,
                                            y1))),
                  f-implies (f-implies (forsome (z1,
                                                 f-equal (x,
                                                          z-opair (y1,
                                                                   z1))),
                                        forsome (z1,
                                                 f-equal (y,
                                                          y1))),
                             f-implies (forsome (z1,
                                                 f-equal (x,
                                                          z-opair (y1,
                                                                   z1))),
                                        f-equal (y,
                                                 y1))),
                  list (forsome-intro-proof (z1,
                                             f-equal (y,
                                                      y1),
                                             f-equal (y,
                                                      y1),
                                             prop-axiom-proof (f-equal (y,
                                                                        y1))))))))
```

THEOREM: z-car-unique-step2-proof-proves

(var-set (list (y, $y1$, z, $z1$), 4)
 ∧ termp (x, 0, *symbols*)
 ∧ nil-intersect (list (z, $z1$), collect-free (x, 0))
 ∧ subset (list (p (1, 2), fn (1, 2)), *symbols*))
 → proves (z-car-unique-step2-proof(x, y, $y1$, z, $z1$),
  f-implies (forsome (z, f-equal (x, z-opair (y, z))),
   f-implies (forsome ($z1$, f-equal (x, z-opair ($y1$, $z1$))),
    f-equal (y, $y1$))),
  given,
  defns,
  symbols)

EVENT: Enable z-car-unique-step2-proof; name this event 'g0637'.

THEOREM: subst-subst
(($y \notin$ collect-free (exp, flg))
 ∧ free-for (exp, x, y, flg)
 ∧ variable (x)
 ∧ variable (y))
 → (subst (subst (exp, x, y, flg), y, x, flg) = exp)

THEOREM: not-free-subst
($x \notin$ collect-free ($term$, 0))
 → ($x \notin$ collect-free (subst (exp, x, $term$, flg), flg))

DEFINITION:
rename-bndvar-proof (x, y, exp)
= forsome-intro-proof(x,
  subst (subst (exp, x, y, 0), y, x, 0),
  forsome (y, subst (exp, x, y, 0)),
  subst-axiom-proof(subst (exp, x, y, 0), y, x))

THEOREM: rename-bndvar-proof-proves
(($y \notin$ collect-free (exp, 0))
 ∧ free-for (exp, x, y, 0)
 ∧ formula (exp, *symbols*)
 ∧ var-set (list (x, y), 2)
 ∧ ($exp\text{-}inst$ = subst (exp, x, y, 0)))
 → proves (rename-bndvar-proof(x, y, exp),
  f-implies (forsome (x, exp), forsome (y, $exp\text{-}inst$)),
  given,
  defns,
  symbols)

EVENT: Enable rename-bndvar-proof; name this event 'g0640'.

DEFINITION:
z-car-uniqueness-case-proof (x, y, $y1$, z, $z1$)
= tautconseq-proof (list (f-implies (forsome (z, f-equal (x, z-opair ($y1$, z))),
forsome ($z1$, f-equal (x, z-opair ($y1$, $z1$)))),
f-implies (forsome (z, f-equal (x, z-opair (y, z))),
f-implies (forsome ($z1$,
f-equal (x,
z-opair ($y1$,
$z1$))),
f-equal (y, $y1$)))),
f-implies (f-and (forsome (z, f-equal (x, z-opair (y, z))),
forsome (z, f-equal (x, z-opair ($y1$, z)))),
f-equal (y, $y1$)),
list (rename-bndvar-proof (z,
$z1$,
f-equal (x, z-opair ($y1$, z))),
z-car-unique-step2-proof (x, y, $y1$, z, $z1$)))

THEOREM: z-car-uniqueness-case-proof-proves
(var-set (list (y, $y1$, z, $z1$), `4`)
$\wedge$ nil-intersect (list (z, $z1$), collect-free (x, `0`))
$\wedge$ termp (x, `0`, *symbols*)
$\wedge$ subset (list (p (`1`, `2`), fn (`1`, `2`)), *symbols*))
$\rightarrow$ proves (z-car-uniqueness-case-proof (x, y, $y1$, z, $z1$),
f-implies (f-and (forsome (z, f-equal (x, z-opair (y, z))),
forsome (z, f-equal (x, z-opair ($y1$, z)))),
f-equal (y, $y1$)),
given,
defns,
symbols)

EVENT: Enable z-car-uniqueness-case-proof; name this event ‘g0645’.

```
;;;;;;;The definition of the Z2 analogue of CAR.
```

DEFINITION:
z-car-defn (x, y, $y1$, z)
= f-if (forsome ($y1$, forsome (z, f-equal (x, z-opair ($y1$, z)))),
forsome (z, f-equal (x, z-opair (y, z))),
f-equal (y, PHI))

```
;;;;;;;The existence part of the admissibility proof of "(Z-car x)".
```

DEFINITION:

z-car-exis-proof$(x,\ y,\ z)$
= forsome-f-if-case-proof$(y,$
forsome $(y,$ forsome $(z,$ f-equal $(x,$ z-opair $(y,\ z)))),$
forsome $(z,$ f-equal $(x,$ z-opair $(y,\ z))),$
f-equal $(y,$ PHI$),$
prop-axiom-proof(forsome $(y,$
forsome $(z,$
f-equal $(x,$
z-opair $(y,$
$z))))),$
tautconseq-proof(list (forsome $(y,$
f-equal $(y,$ PHI$))),$
f-implies (f-not (forsome $(y,$
forsome $(z,$
f-equal $(x,$
z-opair $(y,$
$z))))),$
forsome $(y,$
f-equal $(y,$
PHI$))),$
list (exis-fun-intro-proof$(y,$
PHI$))))$

THEOREM: z-car-exis-proof-proves
(var-set (list $(y,\ z)$, 2)
∧ subset (list (fn (0, 0), fn (1, 2)), *symbols*)
∧ termp $(x$, 0, *symbols*)
∧ (*concl* = forsome $(y$, z-car-defn $(x,\ y,\ y,\ z))))$
→ proves (z-car-exis-proof$(x,\ y,\ z)$, *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-car-exis-proof; name this event ‘g0646’.

```
;;;;;;;The uniqueness part of the admissibility proof of "(Z-car x)".
```

DEFINITION:
z-car-uniqueness-proof$(x,\ y,\ y1,\ z,\ z1)$
= unique-f-if-proof(forsome $(y,$ forsome $(z,$ f-equal $(x,$ z-opair $(y,\ z)))),$
forsome $(z,$ f-equal $(x,$ z-opair $(y,\ z))),$
forsome $(z,$ f-equal $(x,$ z-opair $(y1,\ z))),$
f-equal $(y,$ PHI$),$
f-equal $(y1,$ PHI$),$
f-equal $(y,\ y1),$
z-car-uniqueness-case-proof$(x,\ y,\ y1,\ z,\ z1),$
unique-fun-intro-proof$(y,\ y1,$ PHI$))$

DEFINITION:
z-car-defn-proof (*term*)
= subst-rule-proof (z-car-defn (0, z-car (0), 1, 2),
0,
term,
fun-defn-proof (fn (4, 1),
list (0),
1,
3,
z-car-defn (0, 1, 1, 2),
z-car-exis-proof (0, 1, 2),
z-car-uniqueness-proof (0, 1, 3, 2, 4)))

EVENT: Enable not-member-delete-equals; name this event 'g0648'.

THEOREM: covering-forsome
variable (x)
$\rightarrow$ (covering (forsome (x, y), *var*, 0)
= **if** memb (*var*, collect-free (forsome (x, y), 0))
then cons (x, covering (y, *var*, 0))
else nil endif)

DEFINITION:
z-car-hyps (*defns*, *symbols*)
= (subset (list (p (1, 2), fn (1, 2), fn (0, 0)), rest-of (fn (4, 1), *symbols*))
$\wedge$ (assoc (fn (4, 1), *defns*) = z-car-defn (0, z-car (0), 1, 2)))

```
;;;;;;;The admissibility of "(Z-car x)".
```

THEOREM: z-car-defn-proof-proves
(subset (list (fn (4, 1), p (1, 2), fn (1, 2), fn (0, 0)), *symbols*)
$\wedge$ z-car-hyps (*defns*, *symbols*)
$\wedge$ termp (*term*, 0, *symbols*)
$\wedge$ (collect-free (*term*, 0) = **nil**)
$\wedge$ (*concl* = z-car-defn (*term*, z-car (*term*), 1, 2)))
$\rightarrow$ proves (z-car-defn-proof (*term*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-car-defn-proof; name this event 'g0649'.

EVENT: Enable z-car-hyps; name this event 'g0678'.

DEFINITION:
z-cdr-unique-step1-proof (x, y, *y1*, z, *z1*)

$=$ tautconseq-proof (list (f-implies (f-equal (x, z-opair (y, z)),
f-implies (f-equal (x, z-opair ($y1$, $z1$)),
f-equal (z-opair (y, z),
z-opair ($y1$, $z1$)))),
f-implies (f-equal (z-opair (y, z), z-opair ($y1$, $z1$)),
f-equal (z, $z1$))),
f-implies (f-equal (x, z-opair (y, z)),
f-implies (f-equal (x, z-opair ($y1$, $z1$)),
f-equal (z, $z1$))),
list (trans-equal-proof1 (z-opair (y, z),
x,
z-opair ($y1$, $z1$)),
z-cdr-unique-proof (y, z, $y1$, $z1$)))

THEOREM: z-cdr-unique-step1-proof-proves
(termp (list (x, y, $y1$, z, $z1$), 1, *symbols*)
$\wedge$ subset (list (p (1, 2), fn (1, 2)), *symbols*))
$\rightarrow$ proves (z-cdr-unique-step1-proof (x, y, $y1$, z, $z1$),
f-implies (f-equal (x, z-opair (y, z)),
f-implies (f-equal (x, z-opair ($y1$, $z1$)), f-equal (z, $z1$))),
given,
defns,
symbols)

EVENT: Enable z-cdr-unique-step1-proof; name this event ‘g0647’.

DEFINITION:
z-cdr-unique-step2-proof (x, y, $y1$, z, $z1$)
$=$ forsome-intro-proof (y,
f-equal (x, z-opair (y, z)),
f-implies (forsome ($y1$, f-equal (x, z-opair ($y1$, $z1$))),
f-equal (z, $z1$)),
chain-proof (f-equal (x, z-opair (y, z)),
f-implies (forsome ($y1$,
f-equal (x,
z-opair ($y1$,
$z1$))),
f-equal (z, $z1$)),
list ($\forall$ $y1$ f-implies (f-equal (x,
z-opair ($y1$,
$z1$)),
f-equal (z, $z1$)),
f-implies (forsome ($y1$,
f-equal (x,

```
                                          z-opair (y1,
                                                   z1))),
                       forsome (y1,
                                f-equal (z, z1)))),
list (forall-intro-proof(f-equal (x,
                                  z-opair (y,
                                           z)),
                         f-implies (f-equal (x,
                                             z-opair (y1,
                                                      z1)),
                                    f-equal (z,
                                             z1)),
                         y1,
                         z-cdr-unique-step1-proof (x,
                                                   y,
                                                   y1,
                                                   z,
                                                   z1)),
      forall-over-implies-proof (y1,
                                 f-equal (x,
                                          z-opair (y1,
                                                   z1)),
                                 f-equal (z,
                                          z1)),
      tautconseq-proof (list (f-implies (forsome (y1,
                                                  f-equal (z,
                                                           z1)),
                                         f-equal (z,
                                                  z1))),
                        f-implies (f-implies (forsome (y1,
                                                       f-equal (x,
                                                                z-opair (y1,
                                                                         z1))),
                                              forsome (y1,
                                                       f-equal (z,
                                                                z1))),
                                   f-implies (forsome (y1,
                                                       f-equal (x,
                                                                z-opair (y1,
                                                                         z1))),
                                              f-equal (z,
                                                       z1))),
                        list (forsome-intro-proof (y1,
                                                   f-equal (z,
```

```
                                                                                          z1),
                                                                                  f-equal (z,
                                                                                          z1),
                                                                                  prop-axiom-proof (f-equal (z,
                                                                                                            z1))))))))
```

THEOREM: z-cdr-unique-step2-proof-proves

```
 (var-set (list (y, y1, z, z1), 4)
  ∧   termp (x, 0, symbols)
  ∧   nil-intersect (list (y, y1), collect-free (x, 0))
  ∧   subset (list (p (1, 2), fn (1, 2)), symbols))
 →    proves (z-cdr-unique-step2-proof (x, y, y1, z, z1),
              f-implies (forsome (y, f-equal (x, z-opair (y, z))),
                         f-implies (forsome (y1, f-equal (x, z-opair (y1, z1))),
                                    f-equal (z, z1))),
              given,
              defns,
              symbols)
```

EVENT: Enable z-cdr-unique-step2-proof; name this event ‘g0650’.

DEFINITION:

```
z-cdr-uniqueness-case-proof (x, y, y1, z, z1)
=   tautconseq-proof (list (f-implies (forsome (y, f-equal (x, z-opair (y, z1))),
                                      forsome (y1, f-equal (x, z-opair (y1, z1)))),
                           f-implies (forsome (y, f-equal (x, z-opair (y, z))),
                                      f-implies (forsome (y1,
                                                          f-equal (x,
                                                                   z-opair (y1,
                                                                            z1))),
                                                 f-equal (z, z1)))),
                      f-implies (f-and (forsome (y, f-equal (x, z-opair (y, z))),
                                        forsome (y, f-equal (x, z-opair (y, z1)))),
                                 f-equal (z, z1)),
                      list (rename-bndvar-proof (y,
                                                 y1,
                                                 f-equal (x, z-opair (y, z1))),
                           z-cdr-unique-step2-proof (x, y, y1, z, z1)))
```

THEOREM: z-cdr-uniqueness-case-proof-proves

```
 (var-set (list (y, y1, z, z1), 4)
  ∧   nil-intersect (list (y, y1), collect-free (x, 0))
  ∧   termp (x, 0, symbols)
  ∧   subset (list (p (1, 2), fn (1, 2)), symbols))
```

```
→    proves (z-cdr-uniqueness-case-proof (x, y, y1, z, z1),
             f-implies (f-and (forsome (y, f-equal (x, z-opair (y, z))),
                                forsome (y, f-equal (x, z-opair (y, z1)))),
                         f-equal (z, z1)),
             given,
             defns,
             symbols)
```

```
;;;;;;;The definition of the Z2 analogue of CDR, "(Z-cdr x)".
```

DEFINITION:

```
z-cdr-defn (x, z, y1, z1)
=   f-if (forsome (z1, forsome (y1, f-equal (x, z-opair (y1, z1)))),
          forsome (y1, f-equal (x, z-opair (y1, z))),
          f-equal (z, PHI))
```

```
;;;;;;;The existence part of the admissibility of "(Z-cdr x)".
```

DEFINITION:

```
z-cdr-exis-proof (x, y, z)
=   forsome-f-if-case-proof (z,
                             forsome (z, forsome (y, f-equal (x, z-opair (y, z)))),
                             forsome (y, f-equal (x, z-opair (y, z))),
                             f-equal (z, PHI),
                             prop-axiom-proof (forsome (z,
                                                        forsome (y,
                                                                 f-equal (x,
                                                                          z-opair (y,
                                                                                   z)))))),
                             tautconseq-proof (list (forsome (z,
                                                             f-equal (z, PHI))),
                                               f-implies (f-not (forsome (z,
                                                                          forsome (y,
                                                                                   f-equal (x,
                                                                                            z-opair (y,
                                                                                                     z)))))),
                                                          forsome (z,
                                                                   f-equal (z,
                                                                            PHI)))),
                                               list (exis-fun-intro-proof (z,
                                                                           PHI))))
```

THEOREM: z-cdr-exis-proof-proves

```
 (var-set (list (y, z), 2)
  ∧    subset (list (fn (0, 0), fn (1, 2)), symbols)
```

$\wedge$ termp (x, 0, *symbols*))
$\rightarrow$ proves (z-cdr-exis-proof (x, y, z),
forsome (z, z-cdr-defn (x, z, y, z)),
given,
defns,
symbols)

EVENT: Enable z-cdr-exis-proof; name this event ‘g0651’.

;;;;;;;The uniqueness part of the admissibility of "(Z-cdr x)".

DEFINITION:
z-cdr-uniqueness-proof (x, y, $y1$, z, $z1$)
= unique-f-if-proof (forsome (z, forsome (y, f-equal (x, z-opair (y, z)))),
forsome (y, f-equal (x, z-opair (y, z))),
forsome (y, f-equal (x, z-opair (y, $z1$))),
f-equal (z, PHI),
f-equal ($z1$, PHI),
f-equal (z, $z1$),
z-cdr-uniqueness-case-proof (x, y, $y1$, z, $z1$),
unique-fun-intro-proof (z, $z1$, PHI))

DEFINITION:
z-cdr-defn-proof (*term*)
= subst-rule-proof (z-cdr-defn (0, z-cdr (0), 1, 2),
0,
term,
fun-defn-proof (fn (5, 1),
list (0),
2,
4,
z-cdr-defn (0, 2, 1, 2),
z-cdr-exis-proof (0, 1, 2),
z-cdr-uniqueness-proof (0, 1, 3, 2, 4)))

DEFINITION:
z-cdr-hyps (*defns*, *symbols*)
= (subset (list (p (1, 2), fn (1, 2), fn (0, 0)), rest-of (fn (5, 1), *symbols*))
$\wedge$ (assoc (fn (5, 1), *defns*) = z-cdr-defn (0, z-cdr (0), 1, 2)))

;;;;;;;The admissibility of "(Z-cdr x)".

THEOREM: z-cdr-defn-proof-proves
(subset (list (fn (5, 1), p (1, 2), fn (1, 2), fn (0, 0)), *symbols*)
$\wedge$ z-cdr-hyps (*defns*, *symbols*)

$\wedge$ termp (*term*, 0, *symbols*)
$\wedge$ (collect-free (*term*, 0) = **nil**)
$\wedge$ (*concl* = z-cdr-defn (*term*, z-cdr (*term*), 1, 2)))
$\rightarrow$ proves (z-cdr-defn-proof (*term*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-cdr-hyps; name this event 'g0679'.

EVENT: Enable z-cdr-defn-proof; name this event 'g0655'.

DEFINITION:
list-num-nequal-proof (*n*, *u*, *v*)
= tautconseq-proof (list (f-implies (f-equal (numeral (*n*), z-opair (*u*, *v*)),
f-implies (z-int (numeral (*n*)),
z-int (z-opair (*u*, *v*)))),
z-int (numeral (*n*)),
f-not (z-int (z-opair (*u*, *v*)))),
f-not (f-equal (numeral (*n*), z-opair (*u*, *v*))),
list (equal-axiom2-proof (p (2, 1),
list (numeral (*n*)),
list (z-opair (*u*, *v*))),
z-int-numeral-proof (*n*),
not-int-opair-proof (*u*, *v*, 0)))

EVENT: Disable z-int; name this event 'g0659'.

THEOREM: equal-axiom2-z-int
(termp (list (*x*, *y*), 1, *symbols*) $\wedge$ (p (2, 1) $\in$ *symbols*))
$\rightarrow$ proves (equal-axiom2-proof (p (2, 1), list (*x*), list (*y*)),
f-implies (f-equal (*x*, *y*), f-implies (z-int (*x*), z-int (*y*))),
given,
defns,
symbols)

EVENT: Enable z-int; name this event 'g0660'.

```
;;;;;;;An important metatheorem: |- "(not (|N| = <u v>))".
```

THEOREM: list-num-nequal-proof-proves
(z-succ-hyps (*defns*, *symbols*)
$\wedge$ subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), *symbols*)
$\wedge$ (p (2, 1) $\in$ *symbols*)
$\wedge$ z-int-hyps (*defns*, *symbols*)
$\wedge$ termp (list (*u*, *v*), 1, *symbols*)

$\land\quad (concl = \text{f-not}\,(\text{f-equal}\,(\text{numeral}\,(n),\ \text{z-opair}\,(u,\ v)))))$
$\rightarrow\quad \text{proves}\,(\text{list-num-nequal-proof}(n,\ u,\ v),\ concl,\ given,\ defns,\ symbols)$

EVENT: Enable list-num-nequal-proof; name this event ‘g0658’.

```
;;;;;;;The Goedel-encoding.  It takes Lisp data-structures built from numbers using
;;;;;;;CONS to sets in Z2 which are formed from numerals using ordered pairing.
;;;;;;;We represent the encoding of X as <X>.
```

DEFINITION:
ncode (x)
= **if** listp (x) **then** z-opair (ncode (car (x)), ncode (cdr (x)))
else numeral (x) **endif**

THEOREM: termp-ncode
((fn (3, 1) $\in$ *symbols*) $\land$ (fn (1, 2) $\in$ *symbols*) $\land$ (fn (0, 0) $\in$ *symbols*))
$\rightarrow$ termp (ncode (x), 0, *symbols*)

THEOREM: collect-free-ncode
collect-free (ncode (x), 0) = **nil**

DEFINITION:
ncode-equal (x, y)
= **if** listp (x)
then if listp (y)
then ncode-equal (car (x), car (y)) $\land$ ncode-equal (cdr (x), cdr (y))
else f endif
elseif listp (y) **then f**
elseif $x \simeq 0$ **then** $y \simeq 0$
elseif $y \simeq 0$ **then f**
else ncode-equal ($x - 1$, $y - 1$) **endif**

EVENT: Enable z-trans; name this event ‘g0756’.

EVENT: Enable z-trich; name this event ‘g0757’.

DEFINITION:
ncode-nequal-step-proof (*x1*, *y1*, x, y, *pf1*, *pf2*)
= detach-proof (f-not (f-equal (*x1*, *y1*)),
f-not (f-equal (x, y)),
pf1,
contra-pos-proof1 (f-equal (x, y), f-equal (*x1*, *y1*), *pf2*))

DEFINITION: z-trans2 (x) = z-trans (x, 1, 2)

DEFINITION: z-trich2 (x) = z-trich $(x, \mathtt{1}, \mathtt{2})$

DEFINITION:
ncode-nequal-proof (x, y)
= **if** listp (x)
then if listp (y)
then if $\neg$ ncode-equal (car (x), car (y))
then ncode-nequal-step-proof (ncode (car (x)),
ncode (car (y)),
ncode (x),
ncode (y),
ncode-nequal-proof (car (x),
car (y)),
z-car-unique-proof (ncode (car (x)),
ncode (cdr (x)),
ncode (car (y)),
ncode (cdr (y))))
else ncode-nequal-step-proof (ncode (cdr (x)),
ncode (cdr (y)),
ncode (x),
ncode (y),
ncode-nequal-proof (cdr (x),
cdr (y)),
z-cdr-unique-proof (ncode (car (x)),
ncode (cdr (x)),
ncode (car (y)),
ncode (cdr (y)))) **endif**
else ncode-nequal-step-proof (ncode (y),
ncode (x),
ncode (x),
ncode (y),
list-num-nequal-proof $(y$,
ncode (car (x)),
ncode (cdr (x))),
symmetry-proof1 (ncode (x),
ncode (y))) **endif**
elseif listp (y)
then list-num-nequal-proof $(x$, ncode (car (y)), ncode (cdr (y)))
elseif $x \simeq \mathtt{0}$
then ext-not-equal-proof (ncode (x),
ncode (y),
ncode $(y - 1)$,
null-axiom-proof (ncode $(y - 1)$),
x-in-z-succ-x-proof (ncode $(y - 1)$))

```
elseif y ≃ 0
then ncode-nequal-step-proof (ncode (y),
                              ncode (x),
                              ncode (x),
                              ncode (y),
                              ext-not-equal-proof (ncode (y),
                                                   ncode (x),
                                                   ncode (x - 1),
                                                   null-axiom-proof (ncode (x - 1)),
                                                   x-in-z-succ-x-proof (ncode (x - 1))),
                              symmetry-proof1 (ncode (x), ncode (y)))
else tautconseq-proof (list (z-trans2 (ncode (x - 1)),
                             z-trans2 (ncode (y - 1)),
                             f-implies (z-trans2 (ncode (x - 1)),
                                        f-implies (z-trans2 (ncode (y - 1)),
                                                   f-implies (f-equal (ncode (x),
                                                                       ncode (y)),
                                                              f-equal (ncode (x - 1),
                                                                       ncode (y - 1))))),
                             f-not (f-equal (ncode (x - 1),
                                             ncode (y - 1)))),
                       f-not (f-equal (ncode (x), ncode (y))),
                       list (tautconseq-proof (list (z-int (ncode (x - 1)),
                                                     f-iff (z-int (ncode (x - 1)),
                                                            f-and (z-trans2 (ncode (x - 1)),
                                                                   z-trich2 (ncode (x - 1))))),
                                               z-trans2 (ncode (x - 1)),
                                               list (z-int-numeral-proof (x - 1),
                                                     z-int-defn-proof (ncode (x - 1)))),
                             tautconseq-proof (list (z-int (ncode (y - 1)),
                                                     f-iff (z-int (ncode (y - 1)),
                                                            f-and (z-trans2 (ncode (y - 1)),
                                                                   z-trich2 (ncode (y - 1))))),
                                               z-trans2 (ncode (y - 1)),
                                               list (z-int-numeral-proof (y - 1),
                                                     z-int-defn-proof (ncode (y - 1)))),
                             z-succ-equal-proof (ncode (x - 1),
                                                 ncode (y - 1),
                                                 1,
                                                 2),
                             ncode-nequal-proof (x - 1, y - 1))) endif

;;;;;;;A crucial metatheorem: If X is different from Y, then |- "(not (<X>=<Y>))".
```

THEOREM: ncode-nequal-proof-proves

(subset (list (fn (3, 1), p (2, 1), p (1, 2), fn (2, 2), fn (1, 2), fn (0, 0)), *symbols*)
 $\wedge$ z-succ-hyps (*defns*, *symbols*)
 $\wedge$ z-int-hyps (*defns*, *symbols*)
 $\wedge$ ($\neg$ ncode-equal (x, y)))
$\rightarrow$ proves (ncode-nequal-proof (x, y),
f-not (f-equal (ncode (x), ncode (y))),
given,
defns,
symbols)

EVENT: Enable ncode-nequal-proof; name this event 'g0758'.

THEOREM: ncode-nequal-proof-proves-again
(subset (list (fn (3, 1), p (2, 1), p (1, 2), fn (2, 2), fn (1, 2), fn (0, 0)), *symbols*)
 $\wedge$ z-succ-hyps (*defns*, *symbols*)
 $\wedge$ z-int-hyps (*defns*, *symbols*)
 $\wedge$ ($\neg$ ncode-equal (x, y))
 $\wedge$ (*concl* = f-not (f-equal (ncode (x), ncode (y)))))
$\rightarrow$ proves (ncode-nequal-proof (x, y), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable ncode-nequal-proof-proves; name this event 'g0760'.

```
;;;;;;;What we would like to do next is to demonstrate the admissibility of
;;;;;;;Z2 analogues of several Lisp functions which are used to define the Lisp
;;;;;;;interpreter.  These include the primitive Lisp functions like LISTP, CAR, CDR,
;;;;;;;CONS, ZEROP, NUMBERP, ADD1, SUB1, and EQUAL.  The definitions of these Z2
;;;;;;;functions involve the use of the logical-if connective.  For example, the
;;;;;;;definition of the analogue of CAR, "(Z-car x)", is
;;;;;;;         "(if (forsome y (forsome z (x = <y, z>)))
;;;;;;;           then (forsome z (x = <(Z-car x), z>))
;;;;;;;           else ((Z-car x) = |0|))".
;;;;;;;The definition of the analogue of LISTP, "(Z-listp x)", is
;;;;;;;         "(if (forsome y (forsome z (x = <y, z>)))
;;;;;;;           then ((Z-car x) = |1|)
;;;;;;;           else ((Z-car x) = |0|))".
;;;;;;;Since we will be proving a number of metatheorem regarding the logical-if
;;;;;;;connective, it is convenient to have a representation for such expressions
;;;;;;;which is easily manipulated without having to expand the abbreviation "(if A B C)".
;;;;;;;The shell Z-IF is used to represent if-trees.
```

EVENT: Add the shell *z-if*, with recognizer function symbol *z-ifp* and 3 accessors: *z-test*, with type restriction (none-of) and default value zero; *z-left*, with type restriction (none-of) and default value zero; *z-right*, with type restriction (none-of) and default value zero.

```
;;;;;;;TERM-Z-IF checks if every test position of the if-tree EXP is a formula,
;;;;;;;and every leaf expression is a term in which VAR does not occur free.
;;;;;;;We will call such an if-tree a `term-if-tree'.  This is useful in representing
;;;;;;;a definition by cases where each case has the form "((f x) = exp)".
;;;;;;;The term-if-tree corresponding to the definition of "(Z-listp x)" is
;;;;;;;(Z-IF "(forsome y (forsome z (x = <y, z>)))"  |1|  |0|).
```

DEFINITION:

term-z-if (*exp*, *var*, *symbols*)

```
=   if z-ifp (exp)
    then formula (z-test (exp), symbols)
           ∧   (var ∉ collect-free (z-test (exp), 0))
           ∧   term-z-if (z-left (exp), var, symbols)
           ∧   term-z-if (z-right (exp), var, symbols)
    else termp (exp, 0, symbols) ∧ (var ∉ collect-free (exp, 0)) endif
```

```
;;;;;;;MAKE-TERM-Z-IF makes an formula out of the term-if-tree by replacing each occurrence
;;;;;;;of Z-IF by the logical-if connective, and returning "(var = exp)" for each
;;;;;;;leaf term "exp".  Thus MAKE-TERM-Z-IF applied to the term-if-tree for "Z-listp"
;;;;;;;with the variable "value" for VAR returns the definition for "Z-listp".
```

DEFINITION:

make-term-z-if (*var*, *exp*)

```
=   if z-ifp (exp)
    then f-if (z-test (exp),
               make-term-z-if (var, z-left (exp)),
               make-term-z-if (var, z-right (exp)))
    else f-equal (var, exp) endif
```

DEFINITION:

term-z-if-exis-proof (*var*, *exp*)

```
=   if z-ifp (exp)
    then forsome-f-if-case-proof (var,
                                  z-test (exp),
                                  make-term-z-if (var, z-left (exp)),
                                  make-term-z-if (var, z-right (exp)),
                                  tautconseq-proof (list (forsome (var,
                                                                   make-term-z-if (var,
                                                                                   z-left (exp)))),
                                                         f-implies (z-test (exp),
                                                                    forsome (var,
                                                                             make-term-z-if (var,
                                                                                             z-left (exp)))),
                                                         list (term-z-if-exis-proof (var,
                                                                                     z-left (exp)))),
```

tautconseq-proof (list (forsome (var,
make-term-z-if (var,
z-right (exp)))),
f-implies (f-not (z-test (exp)),
forsome (var,
make-term-z-if (var,
z-right (exp)))),
list (term-z-if-exis-proof (var,
z-right (exp)))))

else exis-fun-intro-proof (var, exp) **endif**

THEOREM: formula-make-term-z-if
(term-z-if (exp, var, $symbols$) $\wedge$ variable (var))
$\rightarrow$ formula (make-term-z-if (var, exp), $symbols$)

```
;;;;;;;Many of the functions are defined as formulas constructed from
;;;;;;;term-if-trees generated by the Z-IF shell.  The lemma below is used to
;;;;;;;carry out the existence part of the admissibility of such definitions.
;;;;;;;The crucial part in the proof of this lemma is the observation that
;;;;;;;if "value" is not free in TEST, then from
;;;;;;; "(if TEST (forsome value EXPL) (forsome value EXPR))",
;;;;;;;we can derive "(forsome value (if TEST EXPL EXPR))".
```

THEOREM: term-z-if-exis-proof-proves
(variable (var) $\wedge$ term-z-if (exp, var, $symbols$))
$\rightarrow$ proves (term-z-if-exis-proof (var, exp),
forsome (var, make-term-z-if (var, exp)),
$given$,
$defns$,
$symbols$)

EVENT: Enable term-z-if-exis-proof; name this event ‘g0761’.

THEOREM: term-z-if-exis-proof-proves1
(variable (var)
$\wedge$ term-z-if (exp, var, $symbols$)
$\wedge$ ($concl$ = forsome (var, make-term-z-if (var, exp))))
$\rightarrow$ proves (term-z-if-exis-proof (var, exp), $concl$, $given$, $defns$, $symbols$)

EVENT: Enable term-z-if-exis-proof-proves; name this event ‘g0762’.

```
;;;;;;;We would now like to show for example that the definition of "Z-listp"
;;;;;;;does in fact make it the Z2 analogue of the Lisp function LISTP.
;;;;;;;That is |- "((Z-listp <X>) = <(BOOL-FIX (LISTP X))>)",
```

```
;;;;;;;where BOOL-FIX converts F to 0, and T to 1.  Again, we overload the notation
;;;;;;;by permitting Lisp functions within quotes to denote their values.
;;;;;;;The predicate PROVES-Z-IF
;;;;;;;requires careful explanation.  Its first argument PF-TREE is a proof-tree
;;;;;;;that mimics the structure of the if-tree EXP.  A proof-tree is of the
;;;;;;;form:  <<cond proof1 proof2> proof-tree-left proof-tree-right>.
;;;;;;;PROVES-Z-IF checks that when EXP is of the form (Z-IF test left right),
;;;;;;;then if cond is T, proof1 is a proof of test, and proof-tree-left is
;;;;;;;the proof-tree corresponding to left; and if cond is F, then proof2 is
;;;;;;;a proof of the negation of test, and proof-tree-right is the proof-tree
;;;;;;;corresponding to right.  In the base case, PROVES-Z-IF checks that PF-TREE
;;;;;;;is a proof of "(EXP = VAL)".  Once this has been checked, we can later evaluate
;;;;;;;the term-if-tree in terms of the values of the leaf nodes by constructing the
;;;;;;;appropriate proof-tree.
```

DEFINITION:
proves-z-if (*pf-tree*, *exp*, *var*, *val*, *given*, *defns*, *symbols*)
= **if** z-ifp (*exp*)
  then if caar (*pf-tree*)
    then proves (cadar (*pf-tree*), z-test (*exp*), *given*, *defns*, *symbols*)
      ∧ proves-z-if (cadr (*pf-tree*),
          z-left (*exp*),
          var,
          val,
          given,
          defns,
          symbols)
    else proves (caddar (*pf-tree*),
        f-not (z-test (*exp*)),
        given,
        defns,
        symbols)
      ∧ proves-z-if (caddr (*pf-tree*),
          z-right (*exp*),
          var,
          val,
          given,
          defns,
          symbols) **endif**
  else proves (*pf-tree*, f-equal (*exp*, *val*), *given*, *defns*, *symbols*) **endif**

DEFINITION:
z-if-eval-proof (*pf-tree*, *exp*, *var*, *val*)
= **if** z-ifp (*exp*)

```
then if caar (pf-tree)
     then chain-proof (make-term-z-if (var, exp),
                       f-equal (var, val),
                       list (make-term-z-if (var, z-left (exp))),
                       list (f-if-elim-proof1 (z-test (exp),
                                               make-term-z-if (var,
                                                               z-left (exp)),
                                               make-term-z-if (var,
                                                               z-right (exp)),
                                               cadar (pf-tree)),
                             z-if-eval-proof (cadr (pf-tree),
                                              z-left (exp),
                                              var,
                                              val)))
     else chain-proof (make-term-z-if (var, exp),
                       f-equal (var, val),
                       list (make-term-z-if (var, z-right (exp))),
                       list (f-if-elim-proof2 (z-test (exp),
                                               make-term-z-if (var,
                                                               z-left (exp)),
                                               make-term-z-if (var,
                                                               z-right (exp)),
                                               caddar (pf-tree)),
                             z-if-eval-proof (caddr (pf-tree),
                                              z-right (exp),
                                              var,
                                              val))) endif
else detach-proof (f-equal (exp, val),
                   f-implies (f-equal (var, exp), f-equal (var, val)),
                   pf-tree,
                   detach-proof (f-equal (var, var),
                                 f-implies (f-equal (exp, val),
                                            f-implies (f-equal (var,
                                                                exp),
                                                       f-equal (var,
                                                                val))),
                                 ident-axiom-proof (var),
                                 equal-axiom2-proof (p (0, 2),
                                                     list (var, exp),
                                                     list (var, val)))) endif
```

THEOREM: formula-make-term-z-if-again
(term-z-if (*exp*, *var*, *symbols*) $\wedge$ termp (*var*, 0, *symbols*))
$\rightarrow$ formula (make-term-z-if (*var*, *exp*), *symbols*)

```
;;;;;;;If each leaf term "exp" in EXP has VAL
;;;;;;;as its value (this is checked by PROVES-Z-IF), then the if-expression
;;;;;;;which has "(var = exp)" for each leaf term "exp" in EXP can be proved
;;;;;;;to imply "(var = VAL)".
```

THEOREM: z-if-eval-proof-proves
(term-z-if (*exp*, *var*, *symbols*)
$\wedge$ termp (list (*var*, *val*), 1, *symbols*)
$\wedge$ proves-z-if (*pf-tree*, *exp*, *var*, *val*, *given*, *defns*, *symbols*))
$\rightarrow$ proves (z-if-eval-proof (*pf-tree*, *exp*, *var*, *val*),
f-implies (make-term-z-if (*var*, *exp*), f-equal (*var*, *val*)),
given,
defns,
symbols)

EVENT: Enable z-if-eval-proof; name this event 'g0765'.

DEFINITION:
z-if-eval-proof1 (*pf-tree*, *exp*, *var*, *val*, *pf*)
= detach-proof (make-term-z-if (*var*, *exp*),
f-equal (*var*, *val*),
pf,
z-if-eval-proof (*pf-tree*, *exp*, *var*, *val*))

THEOREM: z-if-eval-proof-proves1
(term-z-if (*exp*, *var*, *symbols*)
$\wedge$ termp (list (*var*, *val*), 1, *symbols*)
$\wedge$ proves-z-if (*pf-tree*, *exp*, *var*, *val*, *given*, *defns*, *symbols*)
$\wedge$ proves (*pf*, make-term-z-if (*var*, *exp*), *given*, *defns*, *symbols*))
$\rightarrow$ proves (z-if-eval-proof1 (*pf-tree*, *exp*, *var*, *val*, *pf*),
f-equal (*var*, *val*),
given,
defns,
symbols)

EVENT: Enable z-if-eval-proof1; name this event 'g0766'.

DEFINITION:
stack-and-implies (*exp*, *hyps*)
= **if** listp (*hyps*) **then** f-implies (list-f-and (*hyps*), *exp*)
else *exp* **endif**

DEFINITION:
form-z-if-exis-step-proof (*test*, *left*, *right*, *var*, *tests*, *pf1*, *pf2*)

$=$ tautconseq-proof (list (stack-and-implies (forsome (var, $left$),
                                                cons ($test$, $tests$)),
                    stack-and-implies (forsome (var, $right$),
                                                cons (f-not ($test$), $tests$)),
                    f-implies (f-if ($test$,
                                        forsome (var, $left$),
                                        forsome (var, $right$)),
                                   forsome (var, f-if ($test$, $left$, $right$)))),
                stack-and-implies (forsome (var, f-if ($test$, $left$, $right$)),
                                            $tests$),
                list ($pf1$, $pf2$, forsome-f-if-proof (var, $test$, $left$, $right$)))

```
;;;;;;;In the definitions which use term-if-trees, the existence and uniqueness of the
;;;;;;;object being defined follows from that of the leaf terms.  There are other
;;;;;;;definitions where the leaf nodes of the if-trees are formulas (a ‘form-if-tree’),
;;;;;;;and the existence/uniqueness of the defined object in each leaf node case requires
;;;;;;;the conjunction of the tests leading up to it.  The definition of "(Z-car x)",
;;;;;;;              "(if (forsome y (forsome z (x = <y z>)))
;;;;;;;                then (forsome z (x = <(Z-car x), z>))
;;;;;;;                else ((Z-car x) = |0|))".
;;;;;;;is a good example of this.  The lemma below combines the
;;;;;;;existence proofs in each of the left/right cases to form the existence proof
;;;;;;;for the if-subtree rooted at the corresponding test node.
```

THEOREM: form-z-if-exis-step-proof-proves
(variable (var)
$\wedge$ ($var \notin$ collect-free ($test$, 0))
$\wedge$ proves ($pf1$,
        stack-and-implies (forsome (var, $left$), cons ($test$, $tests$)),
        $given$,
        $defns$,
        $symbols$)
$\wedge$ proves ($pf2$,
        stack-and-implies (forsome (var, $right$), cons (f-not ($test$), $tests$)),
        $given$,
        $defns$,
        $symbols$)
$\wedge$ ($concl$ = stack-and-implies (forsome (var, f-if ($test$, $left$, $right$)),
                                        $tests$)))
$\rightarrow$ proves (form-z-if-exis-step-proof ($test$, $left$, $right$, var, $tests$, $pf1$, $pf2$),
        $concl$,
        $given$,
        $defns$,
        $symbols$)

EVENT: Enable form-z-if-exis-step-proof; name this event 'g0769'.

```
;;;;;;;FORM-Z-IF checks if EXP is a form-if-tree.
```

DEFINITION:
form-z-if (*exp*, *symbols*)
= **if** z-ifp (*exp*)
  then formula (z-test (*exp*), *symbols*)
    ∧ form-z-if (z-left (*exp*), *symbols*)
    ∧ form-z-if (z-right (*exp*), *symbols*)
  else formula (*exp*, *symbols*) **endif**

```
;;;;;;;MAKE-FORM-Z-IF returns the if-expression represented by the form-if-tree EXP.
```

DEFINITION:
make-form-z-if (*exp*)
= **if** z-ifp (*exp*)
  then f-if (z-test (*exp*),
      make-form-z-if (z-left (*exp*)),
      make-form-z-if (z-right (*exp*)))
  else *exp* **endif**

```
;;;;;;;PROVES-FORM-Z-IF-EXIS checks that for every leaf formula in the form-if-tree EXP,
;;;;;;;PF-TREE contains a corresponding proof that the conjunction of the tests leading
;;;;;;;to that leaf node "exp" implies "(forsome var exp)".
```

DEFINITION:
proves-form-z-if-exis (*pf-tree*, *exp*, *var*, *tests*, *given*, *defns*, *symbols*)
= **if** z-ifp (*exp*)
  then (*var* ∉ collect-free (z-test (*exp*), 0))
    ∧ proves-form-z-if-exis (car (*pf-tree*),
        z-left (*exp*),
        var,
        cons (z-test (*exp*), *tests*),
        given,
        defns,
        symbols)
    ∧ proves-form-z-if-exis (cadr (*pf-tree*),
        z-right (*exp*),
        var,
        cons (f-not (z-test (*exp*)), *tests*),
        given,
        defns,
        symbols)

else proves (*pf-tree*,
stack-and-implies (forsome (*var*, *exp*), *tests*),
given,
defns,
symbols) **endif**

DEFINITION:
form-z-if-exis-proof (*pf-tree*, *exp*, *var*, *tests*)
= **if** z-ifp (*exp*)
then form-z-if-exis-step-proof (z-test (*exp*),
make-form-z-if (z-left (*exp*)),
make-form-z-if (z-right (*exp*)),
var,
tests,
form-z-if-exis-proof (car (*pf-tree*),
z-left (*exp*),
var,
cons (z-test (*exp*),
tests)),
form-z-if-exis-proof (cadr (*pf-tree*),
z-right (*exp*),
var,
cons (f-not (z-test (*exp*)),
tests)))
else *pf-tree* **endif**

THEOREM: formula-make-form
form-z-if (*exp*, *symbols*) $\rightarrow$ formula (make-form-z-if (*exp*), *symbols*)

```
;;;;;;;The lemma below demonstrates that the existence proofs for the leaf nodes
;;;;;;;can be combined to give the existence proof for the if-expression represented
;;;;;;;by the form-if-tree EXP.
```

THEOREM: form-z-if-exis-proof-proves
(form-z-if (*exp*, *symbols*)
$\land$ variable (*var*)
$\land$ proves-form-z-if-exis (*pf-tree*, *exp*, *var*, *tests*, *given*, *defns*, *symbols*))
$\rightarrow$ proves (form-z-if-exis-proof (*pf-tree*, *exp*, *var*, *tests*),
stack-and-implies (forsome (*var*, make-form-z-if (*exp*)), *tests*),
given,
defns,
symbols)

EVENT: Enable form-z-if-exis-proof; name this event ‘g0770’.

THEOREM: form-z-if-exis-proof-proves1
(form-z-if (*exp*, *symbols*)
∧ variable (*var*)
∧ proves-form-z-if-exis (*pf-tree*, *exp*, *var*, *tests*, *given*, *defns*, *symbols*)
∧ (*concl* = stack-and-implies (forsome (*var*, make-form-z-if (*exp*)), *tests*)))
→ proves (form-z-if-exis-proof (*pf-tree*, *exp*, *var*, *tests*),
concl,
given,
defns,
symbols)

EVENT: Enable form-z-if-exis-proof-proves; name this event 'g0784'.

```
;;;;;;;Next we show how definitions represented by form-if-trees are evaluated.
;;;;;;;PROVES-FORM-Z-IF-EVAL is the form-if-tree analogue of PROVES-Z-IF.
;;;;;;;The only difference is that in the base case, PROVES-FORM-Z-IF-EVAL
;;;;;;;checks if PF-TREE is a proof of "(EXP -> (LHS = VAL))".
```

DEFINITION:
proves-form-z-if-eval (*pf-tree*, *exp*, *lhs*, *val*, *given*, *defns*, *symbols*)
= **if** z-ifp (*exp*)
then if caar (*pf-tree*)
then proves (cadar (*pf-tree*), z-test (*exp*), *given*, *defns*, *symbols*)
∧ proves-form-z-if-eval (cadr (*pf-tree*),
z-left (*exp*),
lhs,
val,
given,
defns,
symbols)
else proves (caddar (*pf-tree*),
f-not (z-test (*exp*)),
given,
defns,
symbols)
∧ proves-form-z-if-eval (caddr (*pf-tree*),
z-right (*exp*),
lhs,
val,
given,
defns,
symbols) **endif**
else proves (*pf-tree*,
f-implies (*exp*, f-equal (*lhs*, *val*)),

given,
defns,
symbols) **endif**

DEFINITION:
form-z-if-eval-proof (*pf-tree*, *exp*, *lhs*, *val*)
= **if** z-ifp (*exp*)
then if caar (*pf-tree*)
then chain-proof (make-form-z-if (*exp*),
f-equal (*lhs*, *val*),
list (make-form-z-if (z-left (*exp*))),
list (f-if-elim-proof1 (z-test (*exp*),
make-form-z-if (z-left (*exp*)),
make-form-z-if (z-right (*exp*)),
cadar (*pf-tree*)),
form-z-if-eval-proof (cadr (*pf-tree*),
z-left (*exp*),
lhs,
val)))
else chain-proof (make-form-z-if (*exp*),
f-equal (*lhs*, *val*),
list (make-form-z-if (z-right (*exp*))),
list (f-if-elim-proof2 (z-test (*exp*),
make-form-z-if (z-left (*exp*)),
make-form-z-if (z-right (*exp*)),
caddar (*pf-tree*)),
form-z-if-eval-proof (caddr (*pf-tree*),
z-right (*exp*),
lhs,
val))) **endif**
else *pf-tree* **endif**

THEOREM: form-z-if-eval-proof-proves
(form-z-if (*exp*, *symbols*)
$\wedge$ termp (list (*lhs*, *val*), `1`, *symbols*)
$\wedge$ proves-form-z-if-eval (*pf-tree*, *exp*, *lhs*, *val*, *given*, *defns*, *symbols*))
$\rightarrow$ proves (form-z-if-eval-proof (*pf-tree*, *exp*, *lhs*, *val*),
f-implies (make-form-z-if (*exp*), f-equal (*lhs*, *val*)),
given,
defns,
symbols)

EVENT: Enable form-z-if-eval-proof; name this event 'g0785'.

DEFINITION:

form-z-if-eval-proof1 (*pf-tree*, *exp*, *lhs*, *val*, *pf*)
= detach-proof (make-form-z-if (*exp*),
f-equal (*lhs*, *val*),
pf,
form-z-if-eval-proof (*pf-tree*, *exp*, *lhs*, *val*))

```
;;;;;;;Now we can show that if we have a proof-tree satisfying PROVES-FORM-Z-IF-EVAL,
;;;;;;;then we can prove that the function can be evaluated as desired, e.g.,
;;;;;;;we can show that "((Z-car <X>) = <(CAR X)>)".
```

THEOREM: form-z-if-eval-proof-proves1
(form-z-if (*exp*, *symbols*)
∧ termp (list (*lhs*, *val*), `1`, *symbols*)
∧ proves-form-z-if-eval (*pf-tree*, *exp*, *lhs*, *val*, *given*, *defns*, *symbols*)
∧ proves (*pf*, make-form-z-if (*exp*), *given*, *defns*, *symbols*))
→ proves (form-z-if-eval-proof1 (*pf-tree*, *exp*, *lhs*, *val*, *pf*),
f-equal (*lhs*, *val*),
given,
defns,
symbols)

EVENT: Enable form-z-if-eval-proof1; name this event ‘g0786’.

THEOREM: nlistp-nil-intersect
($y \simeq$ **nil**) → nil-intersect (x, y)

THEOREM: not-free-free-for
(collect-free (*term*, `0`) $\simeq$ **nil**) → free-for (*exp*, *var*, *term*, *flg*)

THEOREM: listp-append
listp (append (x, y)) = (listp (x) ∨ listp (y))

EVENT: Enable free-for; name this event ‘g0666’.

THEOREM: not-free-for-list
(collect-free (*termlist*, `1`) $\simeq$ **nil**) → free-for-list (*exp*, *varlist*, *termlist*)

EVENT: Enable free-for-list; name this event ‘g0671’.

EVENT: Enable listp-append; name this event ‘g0680’.

DEFINITION:
listp-cond-proof1 (x, y)

= detach-proof (f-equal (z-opair (x, y), z-opair (x, y)),
forsome (1,
forsome (2, f-equal (z-opair (x, y), z-opair (1, 2)))),
ident-axiom-proof (z-opair (x, y)),
subst-thm-a-proof (f-equal (z-opair (x, y), z-opair (1, 2)),
list (1, 2),
list (x, y)))

Theorem: listp-cond-proof1-proves
(subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (2, 1), p (1, 2), fn (0, 0)), *symbols*)
$\wedge$ z-int-hyps (*defns*, *symbols*)
$\wedge$ z-succ-hyps (*defns*, *symbols*)
$\wedge$ termp (list (x, y), 1, *symbols*)
$\wedge$ (collect-free (x, 0) = **nil**)
$\wedge$ (collect-free (y, 0) = **nil**))
$\rightarrow$ proves (listp-cond-proof1 (x, y),
forsome (1, forsome (2, f-equal (z-opair (x, y), z-opair (1, 2)))),
given,
defns,
symbols)

Event: Enable listp-cond-proof1; name this event 'g0685'.

Definition:
numeral-not-listp-proof (n, x, y, u, v)
= f-iff-mp-proof2 (f-not (forsome (x,
forsome (y,
f-equal (numeral (n), z-opair (u, v))))),
$\forall$ x f-not (forsome (y, f-equal (numeral (n), z-opair (u, v)))),
forsome-forall-proof (x,
forsome (y,
f-equal (numeral (n),
z-opair (u, v)))),
generalise-proof (x,
f-not (forsome (y,
f-equal (numeral (n),
z-opair (u, v)))),
f-iff-mp-proof2 (f-not (forsome (y,
f-equal (numeral (n),
z-opair (u,
v)))),
$\forall$ y f-not (f-equal (numeral (n),
z-opair (u,
v))),

```
                                          forsome-forall-proof (y,
                                                               f-equal (numeral (n),
                                                                        z-opair (u,
                                                                                 v))),
                                          generalise-proof (y,
                                                            f-not (f-equal (numeral (n),
                                                                            z-opair (u,
                                                                                     v))),
                                                            list-num-nequal-proof (n,
                                                                                   u,
                                                                                   v)))))
```

`;;;;;;;No numeral is an ordered pair.`

THEOREM: numeral-not-listp-proof-proves
(z-succ-hyps (*defns*, *symbols*)
 ∧ subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (2, 1), p (1, 2), fn (0, 0)),
            symbols)
 ∧ z-int-hyps (*defns*, *symbols*)
 ∧ var-list (list (x, y, u, v), 4)
 ∧ (n-*bar* = numeral (n)))
→ proves (numeral-not-listp-proof (n, x, y, u, v),
           f-not (forsome (x, forsome (y, f-equal (n-*bar*, z-opair (u, v))))),
           given,
           defns,
           symbols)

EVENT: Enable numeral-not-listp-proof; name this event ‘g0693’.

DEFINITION:
z-car-ncode-proof (x)
= form-z-if-eval-proof1 (list (list (listp (x),

```
                          listp-cond-proof1 (ncode (car (x)),
                                             ncode (cdr (x))),
                          numeral-not-listp-proof (x, 1, 2, 1, 2)),
                     forsome-intro-proof (2,
                                          f-equal (z-opair (ncode (car (x)),
                                                            ncode (cdr (x))),
                                                   z-opair (z-car (ncode (x)),
                                                            2)),
                                          f-equal (z-car (ncode (x)),
                                                   ncode (car (x))),
                                          chain-proof (f-equal (z-opair (ncode (car (x)),
                                                                         ncode (cdr (x))),
```

```
                                        z-opair (z-car (ncode (x)),
                                                 2)),
                                    f-equal (z-car (ncode (x)),
                                             ncode (car (x))),
                                    list (f-equal (ncode (car (x)),
                                                   z-car (ncode (x)))),
                                    list (z-car-unique-proof (ncode (car (x)),
                                                              ncode (cdr (x)),
                                                              z-car (ncode (x)),
                                                              2),
                                          symmetry-proof1 (ncode (car (x)),
                                                           z-car (ncode (x)))))),
        prop-axiom-proof (f-equal (z-car (ncode (x)),
                                   PHI))),
      z-if (forsome (1,
                     forsome (2,
                              f-equal (ncode (x),
                                       z-opair (1, 2)))),
            forsome (2,
                     f-equal (ncode (x),
                              z-opair (z-car (ncode (x)), 2))),
            f-equal (z-car (ncode (x)), PHI)),
      z-car (ncode (x)),
      ncode (car (x)),
      z-car-defn-proof (ncode (x)))
```

```
;;;;;;;FORM-Z-IF-EVAL-PROOF-PROVES1 is used to prove that "(Z-car x)" is
;;;;;;;the Z2 analogue of CAR, in the lemma below.
```

THEOREM: z-car-ncode-proof-proves
(z-succ-hyps (*defns*, *symbols*)
 $\wedge$ subset (list (fn (4, 1),
              fn (3, 1),
              fn (2, 2),
              fn (1, 2),
              p (2, 1),
              p (1, 2),
              fn (0, 0)),
         symbols)
 $\wedge$ z-int-hyps (*defns*, *symbols*)
 $\wedge$ z-car-hyps (*defns*, *symbols*)
 $\wedge$ (*concl* = f-equal (z-car (ncode (x)), ncode (car (x)))))
$\rightarrow$ proves (z-car-ncode-proof (x), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-car-ncode-proof; name this event 'g0691'.

DEFINITION:
listp-cond-proof2 (x, y)
```
=   detach-proof (f-equal (z-opair (x, y), z-opair (x, y)),
                  forsome (2,
                           forsome (1, f-equal (z-opair (x, y), z-opair (1, 2)))),
                  ident-axiom-proof (z-opair (x, y)),
                  subst-thm-a-proof (f-equal (z-opair (x, y), z-opair (1, 2)),
                                     list (2, 1),
                                     list (y, x)))
```

THEOREM: listp-cond-proof2-proves
```
 (subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (2, 1), p (1, 2), fn (0, 0)), symbols)
  ∧   z-int-hyps (defns, symbols)
  ∧   z-succ-hyps (defns, symbols)
  ∧   termp (list (x, y), 1, symbols)
  ∧   (collect-free (x, 0) = nil)
  ∧   (collect-free (y, 0) = nil))
 →    proves (listp-cond-proof2 (x, y),
              forsome (2, forsome (1, f-equal (z-opair (x, y), z-opair (1, 2)))),
              given,
              defns,
              symbols)
```

EVENT: Enable listp-cond-proof2; name this event 'g0692'.

DEFINITION:
z-cdr-ncode-proof (x)
```
=   form-z-if-eval-proof1 (list (list (listp (x),
                                        listp-cond-proof2 (ncode (car (x)),
                                                           ncode (cdr (x))),
                                        numeral-not-listp-proof (x, 2, 1, 1, 2)),
                                  forsome-intro-proof (1,
                                                       f-equal (z-opair (ncode (car (x)),
                                                                         ncode (cdr (x))),
                                                                z-opair (1,
                                                                         z-cdr (ncode (x)))),
                                                       f-equal (z-cdr (ncode (x)),
                                                                ncode (cdr (x))),
                                                       chain-proof (f-equal (z-opair (ncode (car (x)),
                                                                                      ncode (cdr (x))),
                                                                             z-opair (1,
                                                                                      z-cdr (ncode (x)))),
                                                                    f-equal (z-cdr (ncode (x)),
                                                                             ncode (cdr (x))),
```

```
                                            list (f-equal (ncode (cdr (x)),
                                                           z-cdr (ncode (x)))),
                                            list (z-cdr-unique-proof (ncode (car (x)),
                                                                       ncode (cdr (x)),
                                                                       1,
                                                                       z-cdr (ncode (x))),
                                                  symmetry-proof1 (ncode (cdr (x)),
                                                                   z-cdr (ncode (x)))))),
                 prop-axiom-proof (f-equal (z-cdr (ncode (x)),
                                            PHI))),
              z-if (forsome (2,
                             forsome (1,
                                      f-equal (ncode (x),
                                               z-opair (1, 2)))),
                 forsome (1,
                          f-equal (ncode (x),
                                   z-opair (1, z-cdr (ncode (x))))),
                 f-equal (z-cdr (ncode (x)), PHI)),
              z-cdr (ncode (x)),
              ncode (cdr (x)),
              z-cdr-defn-proof (ncode (x)))
```

```
;;;;;;;"(Z-cdr x)" is the Z2 analogue of CDR.
```

THEOREM: z-cdr-ncode-proof-proves

```
 (z-succ-hyps (defns, symbols)
  ∧   subset (list (fn (5, 1),
                    fn (3, 1),
                    fn (2, 2),
                    fn (1, 2),
                    p (2, 1),
                    p (1, 2),
                    fn (0, 0)),
               symbols)
  ∧   z-int-hyps (defns, symbols)
  ∧   z-cdr-hyps (defns, symbols)
  ∧   (concl = f-equal (z-cdr (ncode (x)), ncode (cdr (x)))))
 →    proves (z-cdr-ncode-proof (x), concl, given, defns, symbols)
```

EVENT: Enable z-cdr-ncode-proof; name this event 'g0694'.

THEOREM: col-free-z-car

collect-free (z-car (x), 0) = collect-free (list (x), 1)

THEOREM: covering-z-car
covering (z-car (x), var, 0) = covering (list (x), var, 1)

THEOREM: subst-z-car
subst (z-car (x), var, $term$, 0) = z-car (subst (x, var, $term$, 0))

THEOREM: parallel-subst-z-car
parallel-subst (z-car (x), $vars$, $terms$, 0)
= z-car (parallel-subst (x, $vars$, $terms$, 0))

THEOREM: termp-z-car
termp (z-car (x), 0, $symbols$)
= ((fn (4, 1) $\in$ $symbols$) $\land$ termp (x, 0, $symbols$))

EVENT: Enable z-car; name this event 'g0696'.

THEOREM: col-free-z-cdr
collect-free (z-cdr (x), 0) = collect-free (list (x), 1)

THEOREM: covering-z-cdr
covering (z-cdr (x), var, 0) = covering (list (x), var, 1)

THEOREM: subst-z-cdr
subst (z-cdr (x), var, $term$, 0) = z-cdr (subst (x, var, $term$, 0))

THEOREM: parallel-subst-z-cdr
parallel-subst (z-cdr (x), $vars$, $terms$, 0)
= z-cdr (parallel-subst (x, $vars$, $terms$, 0))

THEOREM: termp-z-cdr
termp (z-cdr (x), 0, $symbols$)
= ((fn (5, 1) $\in$ $symbols$) $\land$ termp (x, 0, $symbols$))

EVENT: Enable z-cdr; name this event 'g0697'.

DEFINITION:
term-z-if-unique-proof (x, y, exp)
= **if** z-ifp (exp)
then unique-f-if-proof (z-test (exp),
make-term-z-if (x, z-left (exp)),
make-term-z-if (y, z-left (exp)),
make-term-z-if (x, z-right (exp)),
make-term-z-if (y, z-right (exp)),
f-equal (x, y),
term-z-if-unique-proof (x, y, z-left (exp)),
term-z-if-unique-proof (x, y, z-right (exp)))
else unique-fun-intro-proof (x, y, exp) **endif**

```
;;;;;;;The uniqueness part of the admissibility proofs for definitions constructed
;;;;;;;by term-if-trees follows from the uniqueness proofs of the leaf nodes.
;;;;;;;The lemma below proves that any definition constructed by a term-if-tree
;;;;;;;automatically has the uniqueness property, since terms represent unique values.
```

THEOREM: term-z-if-unique-proof-proves
(term-z-if(*exp*, x, *symbols*)
 ∧ term-z-if(*exp*, y, *symbols*)
 ∧ var-list(list(x, y), 2))
→ proves(term-z-if-unique-proof(x, y, *exp*),
  f-implies(f-and(make-term-z-if(x, *exp*), make-term-z-if(y, *exp*)),
    f-equal(x, y)),
  given,
  defns,
  symbols)

EVENT: Enable term-z-if-unique-proof; name this event 'g0695'.

THEOREM: term-z-if-unique-proof-proves1
(term-z-if(*exp*, x, *symbols*)
 ∧ term-z-if(*exp*, y, *symbols*)
 ∧ var-list(list(x, y), 2)
 ∧ (*concl* = f-implies(f-and(make-term-z-if(x, *exp*),
    make-term-z-if(y, *exp*)),
   f-equal(x, y))))
→ proves(term-z-if-unique-proof(x, y, *exp*), *concl*, *given*, *defns*, *symbols*)

DEFINITION:
z-listp-exis-proof(x, y, z)
= term-z-if-exis-proof(y,
  z-if(forsome(y,
    forsome(z, f-equal(x, z-opair(y, z)))),
   z-succ(PHI),
   PHI))

DEFINITION:
z-listp-unique-proof(x, y, *y1*, z)
= term-z-if-unique-proof(y,
  y1,
  z-if(forsome(y,
    forsome(z, f-equal(x, z-opair(y, z)))),
   z-succ(PHI),
   PHI))

```
;;;;;;;The definition for "(Z-listp x)".
```

DEFINITION:
z-listp-defn (x, y, $y1$, $z1$)
= f-if (forsome ($y1$, forsome ($z1$, f-equal (x, z-opair ($y1$, $z1$)))),
f-equal (y, z-succ (PHI)),
f-equal (y, PHI))

DEFINITION:
bool-fix (x)
= **if** x **then** 1
else 0 **endif**

```
;;;;;;;The expression representing "(Z-listp x)".
```

DEFINITION: z-listp (x) = list (fn (6, 1), x)

```
;;;;;;;The admissibility proof for "(Z-listp x)".
```

DEFINITION:
z-listp-defn-proof (*term*)
= subst-rule-proof (z-listp-defn (0, z-listp (0), 1, 2),
0,
term,
fun-defn-proof (fn (6, 1),
list (0),
1,
3,
z-listp-defn (0, 1, 1, 2),
z-listp-exis-proof (0, 1, 2),
z-listp-unique-proof (0, 1, 3, 2)))

```
;;;;;;;DEFNS must contain the definition of "Z-listp" associated with the
;;;;;;;function symbol for "Z-listp" and the functions used to define
;;;;;;;"Z-listp" must follow "Z-listp" in SYMBOLS.
```

DEFINITION:
z-listp-hyps (*defns*, *symbols*)
= ((assoc (fn (6, 1), *defns*) = z-listp-defn (0, z-listp (0), 1, 2))
$\land$ subset (list (fn (3, 1), fn (1, 2), fn (0, 0)),
rest-of (fn (6, 1), *symbols*)))

EVENT: Disable free-for; name this event 'g0699'.

```
;;;;;;;The definition for "Z-listp" is proved to be admissible in the lemma below.
```

THEOREM: z-listp-defn-proof-proves
(subset (list (fn (6, 1), fn (3, 1), fn (1, 2), fn (0, 0)), *symbols*)
$\wedge$ z-listp-hyps (*defns*, *symbols*)
$\wedge$ termp (*term*, 0, *symbols*)
$\wedge$ (collect-free (*term*, 0) = **nil**)
$\wedge$ (*concl* = z-listp-defn (*term*, z-listp (*term*), 1, 2)))
$\rightarrow$ proves (z-listp-defn-proof (*term*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-listp-defn-proof; name this event ‘g0703’.

EVENT: Enable z-listp-hyps; name this event ‘g0706’.

DEFINITION:
z-listp-ncode-proof (*x*)
= z-if-eval-proof1 (list (list (listp (*x*),
listp-cond-proof1 (ncode (car (*x*)),
ncode (cdr (*x*))),
numeral-not-listp-proof (*x*, 1, 2, 1, 2)),
ident-axiom-proof (z-succ (PHI)),
ident-axiom-proof (PHI)),
z-if (forsome (1,
forsome (2,
f-equal (ncode (*x*), z-opair (1, 2)))),
z-succ (PHI),
PHI),
z-listp (ncode (*x*)),
ncode (bool-fix (listp (*x*))),
z-listp-defn-proof (ncode (*x*)))

```
;;;;;;;|- "((Z-listp <X>) = <(BOOL-FIX (LISTP X))>)".
```

THEOREM: z-listp-ncode-proof-proves
(subset (list (fn (6, 1), fn (3, 1), p (2, 1), p (1, 2), fn (2, 2), fn (1, 2), fn (0, 0)),
symbols)
$\wedge$ z-succ-hyps (*defns*, *symbols*)
$\wedge$ z-int-hyps (*defns*, *symbols*)
$\wedge$ z-listp-hyps (*defns*, *symbols*)
$\wedge$ (*concl* = f-equal (z-listp (ncode (*x*)), ncode (bool-fix (listp (*x*)))))
$\rightarrow$ proves (z-listp-ncode-proof (*x*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-listp-ncode-proof; name this event ‘g0711’.

```
;;;;;;;A similar sequence of events for "(Z-add1 x)", the Z2 analogue of ADD1.
```

Definition:
z-add1-exis-proof (x, y, z)
= term-z-if-exis-proof (y,
z-if (forsome (y,
forsome (z, f-equal (x, z-opair (y, z)))),
z-succ (PHI),
z-succ (x)))

Definition:
z-add1-unique-proof (x, y, $y1$, z)
= term-z-if-unique-proof (y,
$y1$,
z-if (forsome (y,
forsome (z, f-equal (x, z-opair (y, z)))),
z-succ (PHI),
z-succ (x)))

Definition:
z-add1-defn (x, y, $y1$, $z1$)
= f-if (forsome ($y1$, forsome ($z1$, f-equal (x, z-opair ($y1$, $z1$)))),
f-equal (y, z-succ (PHI)),
f-equal (y, z-succ (x)))

Definition: z-add1 (x) = list (fn (7, 1), x)

Definition:
z-add1-defn-proof (*term*)
= subst-rule-proof (z-add1-defn (0, z-add1 (0), 1, 2),
0,
term,
fun-defn-proof (fn (7, 1),
list (0),
1,
3,
z-add1-defn (0, 1, 1, 2),
z-add1-exis-proof (0, 1, 2),
z-add1-unique-proof (0, 1, 3, 2)))

Definition:
z-add1-hyps (*defns*, *symbols*)
= ((assoc (fn (7, 1), *defns*) = z-add1-defn (0, z-add1 (0), 1, 2))
$\wedge$ subset (list (fn (3, 1), fn (1, 2), fn (0, 0)),
rest-of (fn (7, 1), *symbols*)))

Theorem: z-add1-defn-proof-proves

(subset (list (fn (7, 1), fn (3, 1), fn (1, 2), fn (0, 0)), *symbols*)
∧ z-add1-hyps (*defns*, *symbols*)
∧ termp (*term*, 0, *symbols*)
∧ (collect-free (*term*, 0) = **nil**)
∧ (*concl* = z-add1-defn (*term*, z-add1 (*term*), 1, 2)))
→ proves (z-add1-defn-proof (*term*), *concl*, *given*, *defns*, *symbols*)

Event: Enable z-add1-defn-proof; name this event 'g0712'.

Event: Enable z-add1-hyps; name this event 'g0713'.

Definition:
z-add1-ncode-proof (*x*)
= z-if-eval-proof1 (list (list (listp (*x*),
listp-cond-proof1 (ncode (car (*x*)),
ncode (cdr (*x*))),
numeral-not-listp-proof (*x*, 1, 2, 1, 2)),
ident-axiom-proof (z-succ (PHI)),
ident-axiom-proof (z-succ (ncode (*x*)))),
z-if (forsome (1,
forsome (2,
f-equal (ncode (*x*), z-opair (1, 2)))),
z-succ (PHI),
z-succ (ncode (*x*))),
z-add1 (ncode (*x*)),
ncode (1 + *x*),
z-add1-defn-proof (ncode (*x*)))

Theorem: z-add1-ncode-proof-proves
(subset (list (fn (7, 1), fn (3, 1), p (2, 1), p (1, 2), fn (2, 2), fn (1, 2), fn (0, 0)),
symbols)
∧ z-succ-hyps (*defns*, *symbols*)
∧ z-int-hyps (*defns*, *symbols*)
∧ z-add1-hyps (*defns*, *symbols*)
∧ (*concl* = f-equal (z-add1 (ncode (*x*)), ncode (1 + *x*))))
→ proves (z-add1-ncode-proof (*x*), *concl*, *given*, *defns*, *symbols*)

Event: Enable z-add1-ncode-proof; name this event 'g0714'.

```
;;;;;;;The sequence of events introducing "(Z-numberp x)", the Z2 analogue of NUMBERP.
```

Definition:
z-numberp-defn (*x*, *y*)
= f-if (z-int (*x*), f-equal (*y*, z-succ (PHI)), f-equal (*y*, PHI))

DEFINITION:
z-numberp-exis-proof (x, y)
= term-z-if-exis-proof (y, z-if (z-int (x), z-succ (PHI), PHI))

DEFINITION:
z-numberp-unique-proof (x, y, $y1$)
= term-z-if-unique-proof (y, $y1$, z-if (z-int (x), z-succ (PHI), PHI))

DEFINITION: z-numberp (x) = list (fn (8, 1), x)

DEFINITION:
z-numberp-defn-proof (*term*)
= subst-rule-proof (z-numberp-defn (0, z-numberp (0)),
0,
term,
fun-defn-proof (fn (8, 1),
list (0),
1,
2,
z-numberp-defn (0, 1),
z-numberp-exis-proof (0, 1),
z-numberp-unique-proof (0, 1, 2)))

DEFINITION:
z-numberp-hyps (*defns*, *symbols*)
= ((assoc (fn (8, 1), *defns*) = z-numberp-defn (0, z-numberp (0)))
∧ subset (list (fn (3, 1), p (2, 1), fn (0, 0)),
rest-of (fn (8, 1), *symbols*)))

EVENT: Disable z-int; name this event ‘g0716’.

THEOREM: collect-free-z-int
collect-free (z-int (x), 0) = collect-free (list (x), 1)

EVENT: Enable z-int; name this event ‘g0717’.

THEOREM: z-numberp-defn-proof-proves
(subset (list (fn (8, 1), fn (3, 1), p (2, 1), fn (0, 0)), *symbols*)
∧ z-numberp-hyps (*defns*, *symbols*)
∧ termp (*term*, 0, *symbols*)
∧ (collect-free (*term*, 0) = **nil**)
∧ (*concl* = z-numberp-defn (*term*, z-numberp (*term*))))
→ proves (z-numberp-defn-proof (*term*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-numberp-defn-proof; name this event 'g0725'.

EVENT: Enable z-numberp-hyps; name this event 'g0726'.

DEFINITION:
z-numberp-ncode-proof (x)
= z-if-eval-proof1 (list (list (¬ listp (x),
    z-int-numeral-proof (x),
    not-int-opair-proof (ncode (car (x)),
        ncode (cdr (x)),
        0)),
    ident-axiom-proof (z-succ (PHI)),
    ident-axiom-proof (PHI)),
  z-if (z-int (ncode (x)), z-succ (PHI), PHI),
  z-numberp (ncode (x)),
  ncode (bool-fix (¬ listp (x))),
  z-numberp-defn-proof (ncode (x)))

THEOREM: z-int-numeral-proof-proves-again
(z-succ-hyps (*defns*, *symbols*)
∧ subset (list (fn (3, 1), fn (2, 2), fn (1, 2), p (1, 2), fn (0, 0)), *symbols*)
∧ (p (2, 1) ∈ *symbols*)
∧ z-int-hyps (*defns*, *symbols*)
∧ (*x-bar* = numeral (x)))
→ proves (z-int-numeral-proof (x), z-int (*x-bar*), *given*, *defns*, *symbols*)

EVENT: Enable z-int-numeral-proof-proves; name this event 'g0727'.

THEOREM: z-numberp-ncode-proof-proves
(subset (list (fn (8, 1), fn (3, 1), p (2, 1), p (1, 2), fn (2, 2), fn (1, 2), fn (0, 0)),
  symbols)
∧ z-succ-hyps (*defns*, *symbols*)
∧ z-numberp-hyps (*defns*, *symbols*)
∧ z-int-hyps (*defns*, *symbols*)
∧ (*concl* = f-equal (z-numberp (ncode (x)), ncode (bool-fix (¬ listp (x))))))
→ proves (z-numberp-ncode-proof (x), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-numberp-ncode-proof; name this event 'g0728'.

```
;;;;;;;Sequence of events introducing "(Z-zerop x)", the Z2 analogue of ZEROP.
```

DEFINITION:
z-zerop-exis-proof (x, y)

$=$ term-z-if-exis-proof (y,
  z-if (z-int (x),
    z-if (f-equal (x, PHI), z-succ (PHI), PHI),
    z-succ (PHI)))

DEFINITION:
z-zerop-unique-proof (x, y, $y1$)
$=$ term-z-if-unique-proof (y,
  $y1$,
  z-if (z-int (x),
    z-if (f-equal (x, PHI), z-succ (PHI), PHI),
    z-succ (PHI)))

DEFINITION:
z-zerop-defn (x, y)
$=$ f-if (z-int (x),
  f-if (f-equal (x, PHI), f-equal (y, z-succ (PHI)), f-equal (y, PHI)),
  f-equal (y, z-succ (PHI)))

DEFINITION: z-zerop (x) $=$ list (fn (9, 1), x)

DEFINITION:
z-zerop-defn-proof (*term*)
$=$ subst-rule-proof (z-zerop-defn (0, z-zerop (0)),
  0,
  term,
  fun-defn-proof (fn (9, 1),
    list (0),
    1,
    2,
    z-zerop-defn (0, 1),
    z-zerop-exis-proof (0, 1),
    z-zerop-unique-proof (0, 1, 2)))

DEFINITION:
z-zerop-hyps (*defns*, *symbols*)
$=$ ((assoc (fn (9, 1), *defns*) $=$ z-zerop-defn (0, z-zerop (0)))
  $\wedge$ subset (list (fn (3, 1), p (2, 1), fn (0, 0)),
    rest-of (fn (9, 1), *symbols*)))

THEOREM: z-zerop-defn-proof-proves
(subset (list (fn (9, 1), fn (3, 1), p (2, 1), fn (0, 0)), *symbols*)
$\wedge$ z-zerop-hyps (*defns*, *symbols*)
$\wedge$ termp (*term*, 0, *symbols*)
$\wedge$ (collect-free (*term*, 0) $=$ **nil**)
$\wedge$ (*concl* $=$ z-zerop-defn (*term*, z-zerop (*term*))))
$\rightarrow$ proves (z-zerop-defn-proof (*term*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-zerop-defn-proof; name this event 'g0763'.

EVENT: Enable z-zerop-hyps; name this event 'g0764'.

DEFINITION:
z-zerop-ncode-proof(x)
$=$ z-if-eval-proof1 (list (list ($\neg$ listp (x),
z-int-numeral-proof(x),
not-int-opair-proof (ncode (car (x)),
ncode (cdr (x)),
0)),
list (list ($x \simeq$ 0,
ident-axiom-proof (PHI),
ncode-nequal-proof (x, 0)),
ident-axiom-proof (z-succ (PHI)),
ident-axiom-proof (PHI)),
ident-axiom-proof (z-succ (PHI))),
z-if (z-int (ncode (x)),
z-if (f-equal (ncode (x), PHI), z-succ (PHI), PHI),
z-succ (PHI)),
z-zerop (ncode (x)),
ncode (bool-fix ($x \simeq$ 0)),
z-zerop-defn-proof (ncode (x)))

THEOREM: z-zerop-ncode-proof-proves
(subset (list (fn (9, 1), fn (3, 1), p (2, 1), p (1, 2), fn (2, 2), fn (1, 2), fn (0, 0)),
symbols)
$\wedge$ z-succ-hyps (*defns*, *symbols*)
$\wedge$ z-zerop-hyps (*defns*, *symbols*)
$\wedge$ z-int-hyps (*defns*, *symbols*)
$\wedge$ (*concl* $=$ f-equal (z-zerop (ncode (x)), ncode (bool-fix ($x \simeq$ 0)))))
$\rightarrow$ proves (z-zerop-ncode-proof (x), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-zerop-ncode-proof; name this event 'g0767'.

DEFINITION:
form-z-if-unique-proves (x, y, *exp*, *pftree*, *given*, *defns*, *symbols*)
$=$ **if** z-ifp (*exp*)
then ($x \notin$ collect-free (z-test (*exp*), 0))
$\wedge$ ($y \notin$ collect-free (z-test (*exp*), 0))
$\wedge$ form-z-if-unique-proves (x,
y,
z-left (*exp*),

```
                                                car (pftree),
                                                given,
                                                defns,
                                                symbols)
        ∧   form-z-if-unique-proves (x,
                                                y,
                                                z-right (exp),
                                                cdr (pftree),
                                                given,
                                                defns,
                                                symbols)
   else proves (pftree,
                f-implies (f-and (exp, subst (exp, x, y, 0)), f-equal (x, y)),
                given,
                defns,
                symbols) endif
```

DEFINITION:

```
form-z-if-unique-proof (x, y, exp, pftree)
=  if z-ifp (exp)
   then unique-f-if-proof (z-test (exp),
                           make-form-z-if (z-left (exp)),
                           subst (make-form-z-if (z-left (exp)), x, y, 0),
                           make-form-z-if (z-right (exp)),
                           subst (make-form-z-if (z-right (exp)), x, y, 0),
                           f-equal (x, y),
                           form-z-if-unique-proof (x,
                                                   y,
                                                   z-left (exp),
                                                   car (pftree)),
                           form-z-if-unique-proof (x,
                                                   y,
                                                   z-right (exp),
                                                   cdr (pftree)))
   else pftree endif
```

THEOREM: form-z-if-unique-proof-proves

```
 (form-z-if (exp, symbols)
  ∧   var-list (list (x, y), 2)
  ∧   form-z-if-unique-proves (x, y, exp, pftree, given, defns, symbols))
 →   proves (form-z-if-unique-proof (x, y, exp, pftree),
            f-implies (f-and (make-form-z-if (exp),
                              subst (make-form-z-if (exp), x, y, 0)),
                       f-equal (x, y)),
```

$given$,
$defns$,
$symbols$)

EVENT: Enable form-z-if-eval-proof; name this event 'g0700'.

THEOREM: form-z-if-unique-proof-proves1
(form-z-if (exp, $symbols$)
∧ var-list (list (x, y), 2)
∧ form-z-if-unique-proves (x, y, exp, $pftree$, $given$, $defns$, $symbols$)
∧ ($concl$ = f-implies (f-and (make-form-z-if (exp),
subst (make-form-z-if (exp), x, y, 0)),
f-equal (x, y))))
→ proves (form-z-if-unique-proof (x, y, exp, $pftree$),
$concl$,
$given$,
$defns$,
$symbols$)

EVENT: Enable form-z-if-unique-proof-proves; name this event 'g0701'.

DEFINITION:
trans-not-equal-proof (x, y, z, $pf1$, $pf2$)
= tautconseq-proof (list (f-implies (f-equal (y, x),
f-implies (f-equal (y, z), f-equal (x, z))),
f-not (f-equal (x, z)),
f-equal (y, z)),
f-not (f-equal (y, x)),
list (trans-equal-proof1 (x, y, z), $pf1$, $pf2$))

THEOREM: trans-not-equal-proof-proves
(proves ($pf1$, f-not (f-equal (x, z)), $given$, $defns$, $symbols$)
∧ proves ($pf2$, f-equal (y, z), $given$, $defns$, $symbols$))
→ proves (trans-not-equal-proof (x, y, z, $pf1$, $pf2$),
f-not (f-equal (y, x)),
$given$,
$defns$,
$symbols$)

EVENT: Enable trans-not-equal-proof; name this event 'g0840'.

DEFINITION:
flip-equals-proof (x, y, pf)
= detach-proof (f-equal (x, y), f-equal (y, x), pf, symmetry-proof1 (x, y))

THEOREM: flip-equals-proof-proves
proves (pf, f-equal (x, y), $given$, $defns$, $symbols$)
$\rightarrow$ proves (flip-equals-proof (x, y, pf), f-equal (y, x), $given$, $defns$, $symbols$)

EVENT: Enable flip-equals-proof; name this event 'g0841'.

DEFINITION:
rename-forall-proof (x, y, exp)
= forall-intro-proof ($\forall$ x exp,
subst (exp, x, y, 0),
y,
subst-thm-b-proof (exp, list (x), list (y)))

EVENT: Disable free-for-list; name this event 'g0731'.

THEOREM: parallel-subst-subst-case
parallel-subst (exp, list (x), list ($term$), flg) = subst (exp, x, $term$, flg)

THEOREM: rename-forall-proof-proves
((y $\notin$ collect-free (exp, 0))
$\wedge$ formula (exp, $symbols$)
$\wedge$ free-for (exp, x, y, 0)
$\wedge$ var-set (list (x, y), 2)
$\wedge$ ($exp\text{-}inst$ = subst (exp, x, y, 0)))
$\rightarrow$ proves (rename-forall-proof (x, y, exp),
f-implies ($\forall$ x exp, $\forall$ y $exp\text{-}inst$),
$given$,
$defns$,
$symbols$)

EVENT: Enable free-for-list; name this event 'g0732'.

EVENT: Enable parallel-subst-subst-case; name this event 'g0733'.

EVENT: Enable form-z-if-unique-proof; name this event 'g0734'.

```
;;;;;;;Sequence of events introducing "(Z-sub1 x)", the Z2 analogue of SUB1.
```

DEFINITION: z-sub1 (x) = list (fn (10, 1), x)

DEFINITION:
z-sub1-defn (x, y, z)
= f-if (z-int (x),

```
        f-if (forsome (z, f-and (z-int (z), f-equal (z-succ (z), x))),
              f-and (z-int (y), f-equal (z-succ (y), x)),
              f-equal (y, PHI)),
        f-equal (y, PHI))
```

DEFINITION:
z-sub1-expr $(x,\ y,\ z)$

```
=   z-if (z-int (x),
          z-if (forsome (z, f-and (z-int (z), f-equal (z-succ (z), x))),
                f-and (z-int (y), f-equal (z-succ (y), x)),
                f-equal (y, PHI)),
          f-equal (y, PHI))
```

DEFINITION:
z-sub1-exis-proof$(x,\ y)$

```
=   form-z-if-exis-proof(list (list (tautconseq-proof (nil,
                                                        f-implies (f-and (forsome (y,
                                                                                   f-and (z-int (y),
                                                                                          f-equal (z-succ (y),
                                                                                                   x))),
                                                                          z-int (x)),
                                                                   forsome (y,
                                                                            f-and (z-int (y),
                                                                                   f-equal (z-succ (y),
                                                                                            x)))),
                                                        nil),
                                     tautconseq-proof (list (forsome (y,
                                                                      f-equal (y,
                                                                               PHI))),
                                                        f-implies (f-and (f-not (forsome (y,
                                                                                          f-and (z-int (y),
                                                                                                 f-equal (z-succ (y),
                                                                                                          x)))),
                                                                          z-int (x)),
                                                                   forsome (y,
                                                                            f-equal (y,
                                                                                     PHI))),
                                                        list (exis-fun-intro-proof (y,
                                                                                    PHI)))),
                                tautconseq-proof (list (forsome (y,
                                                                 f-equal (y, PHI))),
                                                   f-implies (f-not (z-int (x)),
                                                              forsome (y,
                                                                       f-equal (y,
```

```
                                                    PHI))),
                                    list (exis-fun-intro-proof (y,
                                                   PHI)))),
                        z-sub1-expr (x, y, y),
                        y,
                        nil)
```

DEFINITION:

```
z-int-z-succ-equal-proof (x, y)
=   tautconseq-proof (list (f-implies (z-int (x),
                                       f-and (z-trans2 (x), z-trich2 (x))),
                            f-implies (z-int (y),
                                       f-and (z-trans2 (y), z-trich2 (y))),
                            f-implies (z-trans2 (x),
                                       f-implies (z-trans2 (y),
                                                  f-implies (f-equal (z-succ (x),
                                                                      z-succ (y)),
                                                             f-equal (x, y))))),
                      f-implies (z-int (x),
                                 f-implies (z-int (y),
                                            f-implies (f-equal (z-succ (x),
                                                                z-succ (y)),
                                                       f-equal (x, y)))),
                      list (iff-split-proof1 (z-int (x),
                                              f-and (z-trans2 (x), z-trich2 (x)),
                                              z-int-defn-proof (x)),
                            iff-split-proof1 (z-int (y),
                                              f-and (z-trans2 (y), z-trich2 (y)),
                                              z-int-defn-proof (y)),
                            z-succ-equal-proof (x, y, 1, 2)))
```

THEOREM: z-int-z-succ-equal-proof-proves

```
 (termp (x, 0, symbols)
  ∧   termp (y, 0, symbols)
  ∧   z-int-hyps (defns, symbols)
  ∧   z-succ-hyps (defns, symbols)
  ∧   subset (list (p (2, 1), p (1, 2), fn (3, 1), fn (2, 2), fn (1, 2), fn (0, 0)),
              symbols)
  ∧   nil-intersect (list (1, 2), collect-free (list (x, y), 1)))
 →   proves (z-int-z-succ-equal-proof (x, y),
             f-implies (z-int (x),
                        f-implies (z-int (y),
                                   f-implies (f-equal (z-succ (x), z-succ (y)),
                                              f-equal (x, y)))),
```

```
            given,
            defns,
            symbols)
```

EVENT: Enable z-int-z-succ-equal-proof; name this event 'g0739'.

DEFINITION:
z-sub1-unique-proof (*x*, *y*, *y1*)

```
=   form-z-if-unique-proof (y,
                            y1,
                            z-sub1-expr (x, y, y),
                            cons (cons (tautconseq-proof (list (f-implies (f-equal (z-succ (y),
                                                                                     x),
                                                                            f-implies (f-equal (z-succ (y1),
                                                                                                x),
                                                                                       f-equal (z-succ (y),
                                                                                                z-succ (y1)))),
                                                                 f-implies (z-int (y),
                                                                            f-implies (z-int (y1),
                                                                                       f-implies (f-equal (z-succ (y),
                                                                                                           z-succ (y1)),
                                                                                                  f-equal (y,
                                                                                                           y1)))),
                                                                 f-implies (f-and (f-and (z-int (y),
                                                                                          f-equal (z-succ (y),
                                                                                                   x)),
                                                                                   f-and (z-int (y1),
                                                                                          f-equal (z-succ (y1),
                                                                                                   x))),
                                                                            f-equal (y,
                                                                                     y1)),
                                                           list (trans-equal-proof2 (z-succ (y),
                                                                                     x,
                                                                                     z-succ (y1)),
                                                                 z-int-z-succ-equal-proof (y,
                                                                                           y1))),
                                        unique-fun-intro-proof (y, y1, PHI)),
                                  unique-fun-intro-proof (y, y1, PHI)))
```

DEFINITION:
z-sub1-defn-proof (*term*)

```
=   subst-rule-proof (z-sub1-defn (0, z-sub1 (0), 3),
                      0,
                      term,
```

fun-defn-proof (fn (10, 1),
list (0),
3,
4,
z-sub1-defn (0, 3, 3),
z-sub1-exis-proof (0, 3),
z-sub1-unique-proof (0, 3, 4)))

DEFINITION:
z-sub1-hyps (*defns*, *symbols*)
= ((assoc (fn (10, 1), *defns*) = z-sub1-defn (0, z-sub1 (0), 3))
∧ z-int-hyps (*defns*, rest-of (fn (10, 1), *symbols*))
∧ z-succ-hyps (*defns*, rest-of (fn (10, 1), *symbols*))
∧ subset (list (fn (3, 1), p (2, 1), p (1, 2), fn (2, 2), fn (1, 2), fn (0, 0)),
rest-of (fn (10, 1), *symbols*)))

THEOREM: z-sub1-defn-proof-proves
(subset (list (fn (10, 1), fn (3, 1), p (2, 1), fn (0, 0)), *symbols*)
∧ z-sub1-hyps (*defns*, *symbols*)
∧ termp (*term*, 0, *symbols*)
∧ nil-intersect (list (3, 4), collect-free (*term*, 0))
∧ (*concl* = z-sub1-defn (*term*, z-sub1 (*term*), 3)))
→ proves (z-sub1-defn-proof (*term*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-sub1-defn-proof; name this event 'g0787'.

EVENT: Enable z-sub1-hyps; name this event 'g0788'.

DEFINITION:
enable-nzerop-proof (x, y)
= detach-proof (f-and (z-int (ncode ($x - 1$)),
f-equal (z-succ (ncode ($x - 1$)), z-succ (ncode ($x - 1$)))),
forsome (y,
f-and (z-int (y),
f-equal (z-succ (y), z-succ (ncode ($x - 1$))))),
and-join-proof (z-int (ncode ($x - 1$)),
f-equal (z-succ (ncode ($x - 1$)),
z-succ (ncode ($x - 1$))),
z-int-numeral-proof ($x - 1$),
ident-axiom-proof (z-succ (ncode ($x - 1$)))),
subst-thm-a-proof (f-and (z-int (y),
f-equal (z-succ (y),
z-succ (ncode ($x - 1$)))),
list (y),

list (ncode (x $-$ 1))))

THEOREM: enable-nzerop-proof-proves
(($x \not\simeq$ 0)
$\wedge$ subset (list (fn (3, 1), p (2, 1), p (1, 2), fn (2, 2), fn (1, 2), fn (0, 0)),
symbols)
$\wedge$ z-int-hyps (*defns*, *symbols*)
$\wedge$ z-succ-hyps (*defns*, *symbols*)
$\wedge$ variable (y)
$\wedge$ (*concl* = forsome (y, f-and (z-int (y), f-equal (z-succ (y), ncode (x))))))
$\rightarrow$ proves (enable-nzerop-proof (x, y), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable enable-nzerop-proof; name this event 'g0790'.

DEFINITION:
disable-nzerop-proof (x, y)
= f-iff-mp-proof2 (f-not (forsome (y,
f-and (z-int (y),
f-equal (z-succ (y), ncode (x))))),
$\forall$ y f-not (f-and (z-int (y), f-equal (z-succ (y), ncode (x)))),
forsome-forall-proof (y,
f-and (z-int (y),
f-equal (z-succ (y), ncode (x)))),
generalise-proof (y,
f-not (f-and (z-int (y),
f-equal (z-succ (y),
ncode (x)))),
tautconseq-proof (list (f-implies (f-equal (z-succ (y),
PHI),
f-equal (PHI,
z-succ (y))),
f-not (f-equal (PHI,
z-succ (y)))),
f-not (f-and (z-int (y),
f-equal (z-succ (y),
PHI))),
list (symmetry-proof1 (z-succ (y),
PHI),
ext-not-equal-proof (PHI,
z-succ (y),
y,
null-axiom-proof (y),
x-in-z-succ-x-proof (y))))))

THEOREM: disable-nzerop-proof-proves

$((\neg\ \text{listp}\,(x))$
$\land\ \ (x \simeq \mathtt{0})$
$\land\ \ \text{subset}\,(\text{list}\,(\text{fn}\,(\mathtt{3},\ \mathtt{1}),\ \text{p}\,(\mathtt{2},\ \mathtt{1}),\ \text{p}\,(\mathtt{1},\ \mathtt{2}),\ \text{fn}\,(\mathtt{2},\ \mathtt{2}),\ \text{fn}\,(\mathtt{1},\ \mathtt{2}),\ \text{fn}\,(\mathtt{0},\ \mathtt{0})),$
$\qquad\qquad \mathit{symbols})$
$\land\ \ \text{z-int-hyps}\,(\mathit{defns},\ \mathit{symbols})$
$\land\ \ \text{z-succ-hyps}\,(\mathit{defns},\ \mathit{symbols})$
$\land\ \ \text{variable}\,(y)$
$\land\ \ (\mathit{concl} = \text{f-not}\,(\text{forsome}\,(y,$
$\qquad\qquad\qquad\qquad \text{f-and}\,(\text{z-int}\,(y),\ \text{f-equal}\,(\text{z-succ}\,(y),\ \text{ncode}\,(x)))))))$
$\to\ \ \text{proves}\,(\text{disable-nzerop-proof}\,(x,\ y),\ \mathit{concl},\ \mathit{given},\ \mathit{defns},\ \mathit{symbols})$

EVENT: Enable disable-nzerop-proof; name this event ‘g0791’.

DEFINITION:

nzerop-z-sub1-ncode-proof (x)
$=\ \ \text{tautconseq-proof}\,(\text{list}\,(\text{z-int}\,(\text{ncode}\,(x - 1)),$
$\qquad\qquad \text{f-implies}\,(\text{z-int}\,(\text{z-sub1}\,(\text{ncode}\,(x))),$
$\qquad\qquad\qquad \text{f-implies}\,(\text{z-int}\,(\text{ncode}\,(x - 1)),$
$\qquad\qquad\qquad\qquad \text{f-implies}\,(\text{f-equal}\,(\text{z-succ}\,(\text{z-sub1}\,(\text{ncode}\,(x))),$
$\qquad\qquad\qquad\qquad\qquad\qquad \text{z-succ}\,(\text{ncode}\,(x - 1))),$
$\qquad\qquad\qquad\qquad\qquad \text{f-equal}\,(\text{z-sub1}\,(\text{ncode}\,(x)),$
$\qquad\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(x - 1)))))),$
$\qquad\qquad \text{f-implies}\,(\text{f-and}\,(\text{z-int}\,(\text{z-sub1}\,(\text{ncode}\,(x))),$
$\qquad\qquad\qquad \text{f-equal}\,(\text{z-succ}\,(\text{z-sub1}\,(\text{ncode}\,(x))),$
$\qquad\qquad\qquad\qquad \text{z-succ}\,(\text{ncode}\,(x - 1)))),$
$\qquad\qquad\qquad \text{f-equal}\,(\text{z-sub1}\,(\text{ncode}\,(x)),\ \text{ncode}\,(x - 1))),$
$\qquad\qquad \text{list}\,(\text{z-int-numeral-proof}\,(x - 1),$
$\qquad\qquad\qquad \text{z-int-z-succ-equal-proof}\,(\text{z-sub1}\,(\text{ncode}\,(x)),$
$\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(x - 1))))$

DEFINITION:

z-sub1-ncode-proof (x)
$=\ \ \text{form-z-if-eval-proof1}\,(\text{list}\,(\text{list}\,(x \simeq \mathbf{nil},$
$\qquad\qquad\qquad \text{z-int-numeral-proof}\,(x),$
$\qquad\qquad\qquad \text{not-int-opair-proof}\,(\text{ncode}\,(\text{car}\,(x)),$
$\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\text{cdr}\,(x)),$
$\qquad\qquad\qquad\qquad\qquad \mathtt{0})),$
$\qquad\qquad \text{list}\,(\text{list}\,(x \not\simeq \mathtt{0},$
$\qquad\qquad\qquad\qquad \text{enable-nzerop-proof}\,(x,\ \mathtt{3}),$
$\qquad\qquad\qquad\qquad \text{disable-nzerop-proof}\,(x,\ \mathtt{3})),$
$\qquad\qquad\qquad \text{nzerop-z-sub1-ncode-proof}\,(x),$
$\qquad\qquad\qquad \text{tautconseq-proof}\,(\mathbf{nil},$
$\qquad\qquad\qquad\qquad\qquad \text{f-implies}\,(\text{f-equal}\,(\text{z-sub1}\,(\text{ncode}\,(x)),$

```
                                                    PHI),
                                            f-equal (z-sub1 (ncode (x)),
                                                    PHI)),
                                   nil)),
                  tautconseq-proof (nil,
                               f-implies (f-equal (z-sub1 (ncode (x)),
                                                  PHI),
                                          f-equal (z-sub1 (ncode (x)),
                                                  PHI)),
                               nil)),
              z-sub1-expr (ncode (x), z-sub1 (ncode (x)), 3),
              z-sub1 (ncode (x)),
              ncode (x - 1),
              z-sub1-defn-proof (ncode (x)))
```

THEOREM: z-sub1-ncode-proof-proves
(subset (list (fn (10, 1), fn (3, 1), p (2, 1), p (1, 2), fn (2, 2), fn (1, 2), fn (0, 0)),
symbols)
$\land$ z-succ-hyps (*defns*, *symbols*)
$\land$ z-sub1-hyps (*defns*, *symbols*)
$\land$ z-int-hyps (*defns*, *symbols*)
$\land$ (*concl* = f-equal (z-sub1 (ncode (x)), ncode ($x - 1$))))
$\rightarrow$ proves (z-sub1-ncode-proof (x), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-sub1-ncode-proof; name this event 'g0793'.

```
;;;;;;;Sequence of events introducing "(Z-equal x y)", the Z2 analogue of EQUAL.
```

DEFINITION: z-equal (x, y) = list (fn (11, 2), x, y)

DEFINITION:
z-equal-defn (x, y, z)
= f-if (f-equal (x, y), f-equal (z, z-succ (PHI)), f-equal (z, PHI))

DEFINITION:
z-equal-defn-proof (x, y)

```
=   instance-proof (z-equal-defn (0, 1, z-equal (0, 1)),
                    list (0, 1),
                    list (x, y),
                    fun-defn-proof (fn (11, 2),
                                    list (0, 1),
                                    2,
                                    3,
                                    z-equal-defn (0, 1, 2),
```

```
term-z-if-exis-proof (2,
                      z-if (f-equal (0, 1),
                            z-succ (PHI),
                            PHI)),
term-z-if-unique-proof (2,
                        3,
                        z-if (f-equal (0,
                                       1),
                              z-succ (PHI),
                              PHI))))
```

DEFINITION:
z-equal-hyps (*defns*, *symbols*)
= ((assoc (fn (11, 2), *defns*) = z-equal-defn (0, 1, z-equal (0, 1)))
  ∧ subset (list (fn (3, 1), fn (0, 0)), rest-of (fn (11, 2), *symbols*)))

EVENT: Disable free-for-list; name this event ‘g0740’.

THEOREM: z-equal-defn-proof-proves
(subset (list (fn (11, 2), fn (3, 1), fn (0, 0)), *symbols*)
 ∧ z-equal-hyps (*defns*, *symbols*)
 ∧ termp (list (x, y), 1, *symbols*)
 ∧ (*concl* = z-equal-defn (x, y, z-equal (x, y))))
→ proves (z-equal-defn-proof (x, y), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-equal-defn-proof; name this event ‘g0794’.

EVENT: Enable z-equal-hyps; name this event ‘g0795’.

THEOREM: ncode-equal-ncode-equal
ncode-equal (x, y) → (ncode (y) = ncode (x))

DEFINITION:
z-equal-ncode-proof (x, y)
```
=  z-if-eval-proof1 (list (list (ncode-equal (x, y),
                                 ident-axiom-proof (ncode (x)),
                                 ncode-nequal-proof (x, y)),
                           ident-axiom-proof (z-succ (PHI)),
                           ident-axiom-proof (PHI)),
                     z-if (f-equal (ncode (x), ncode (y)), z-succ (PHI), PHI),
                     z-equal (ncode (x), ncode (y)),
                     ncode (bool-fix (ncode-equal (x, y))),
                     z-equal-defn-proof (ncode (x), ncode (y)))
```

THEOREM: z-equal-ncode-proof-proves

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{subset}\,(\text{list}\,(\text{fn}\,(11,2),\ \text{p}\,(2,1),\ \text{p}\,(1,2),\ \text{fn}\,(3,1),\ \text{fn}\,(2,2),\ \text{fn}\,(1,2),\ \text{fn}\,(0,0)),} \\
\multicolumn{2}{l}{\qquad\qquad \mathit{symbols})} \\
\wedge & \text{z-int-hyps}\,(\mathit{defns},\ \mathit{symbols}) \\
\wedge & \text{z-succ-hyps}\,(\mathit{defns},\ \mathit{symbols}) \\
\wedge & \text{z-equal-hyps}\,(\mathit{defns},\ \mathit{symbols}) \\
\wedge & (\mathit{concl} = \text{f-equal}\,(\text{z-equal}\,(\text{ncode}\,(x),\ \text{ncode}\,(y)), \\
& \qquad\qquad\qquad \text{ncode}\,(\text{bool-fix}\,(\text{ncode-equal}\,(x,\ y)))))) \\
\rightarrow & \text{proves}\,(\text{z-equal-ncode-proof}\,(x,\ y),\ \mathit{concl},\ \mathit{given},\ \mathit{defns},\ \mathit{symbols})
\end{array}
$$

EVENT: Enable z-equal-ncode-proof; name this event ‘g0796’.

```
;;;;;;;Sequence of events introducing "(Z-cons x y)", the Z2 analogue of CONS.
```

DEFINITION: $\text{z-cons}\,(x,\ y) = \text{list}\,(\text{fn}\,(12,\ 2),\ x,\ y)$

DEFINITION: $\text{z-cons-defn}\,(x,\ y,\ z) = \text{f-equal}\,(z,\ \text{z-opair}\,(x,\ y))$

DEFINITION:

$$
\begin{array}{l}
\text{z-cons-defn-proof}\,(x,\ y) \\
= \quad \text{instance-proof}\,(\text{z-cons-defn}\,(0,\ 1,\ \text{z-cons}\,(0,\ 1)), \\
\qquad\qquad\qquad \text{list}\,(0,\ 1), \\
\qquad\qquad\qquad \text{list}\,(x,\ y), \\
\qquad\qquad\qquad \text{fun-defn-proof}\,(\text{fn}\,(12,\ 2), \\
\qquad\qquad\qquad\qquad\qquad \text{list}\,(0,\ 1), \\
\qquad\qquad\qquad\qquad\qquad 2, \\
\qquad\qquad\qquad\qquad\qquad 3, \\
\qquad\qquad\qquad\qquad\qquad \text{z-cons-defn}\,(0,\ 1,\ 2), \\
\qquad\qquad\qquad\qquad\qquad \text{exis-fun-intro-proof}\,(2,\ \text{z-opair}\,(0,\ 1)), \\
\qquad\qquad\qquad\qquad\qquad \text{unique-fun-intro-proof}\,(2, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad 3, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{z-opair}\,(0,\ 1))))
\end{array}
$$

DEFINITION:

$$
\begin{array}{l}
\text{z-cons-hyps}\,(\mathit{defns},\ \mathit{symbols}) \\
= \quad ((\text{assoc}\,(\text{fn}\,(12,\ 2),\ \mathit{defns}) = \text{z-cons-defn}\,(0,\ 1,\ \text{z-cons}\,(0,\ 1))) \\
\qquad \wedge \quad (\text{fn}\,(1,\ 2) \in \text{rest-of}\,(\text{fn}\,(12,\ 2),\ \mathit{symbols})))
\end{array}
$$

THEOREM: z-cons-defn-proof-proves

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{subset}\,(\text{list}\,(\text{fn}\,(12,\ 2),\ \text{fn}\,(1,\ 2)),\ \mathit{symbols})} \\
\wedge & \text{z-cons-hyps}\,(\mathit{defns},\ \mathit{symbols}) \\
\wedge & \text{termp}\,(\text{list}\,(x,\ y),\ 1,\ \mathit{symbols}) \\
\wedge & (\text{collect-free}\,(x,\ 0) = \mathbf{nil}) \\
\wedge & (\text{collect-free}\,(y,\ 0) = \mathbf{nil}) \\
\wedge & (\mathit{concl} = \text{z-cons-defn}\,(x,\ y,\ \text{z-cons}\,(x,\ y)))) \\
\rightarrow & \text{proves}\,(\text{z-cons-defn-proof}\,(x,\ y),\ \mathit{concl},\ \mathit{given},\ \mathit{defns},\ \mathit{symbols})
\end{array}
$$

EVENT: Enable z-cons-defn-proof; name this event 'g0800'.

EVENT: Enable z-cons-hyps; name this event 'g0801'.

EVENT: Enable z-cons-defn-proof; name this event 'g0741'.

EVENT: Enable z-cons-hyps; name this event 'g0742'.

THEOREM: nzerop-ncode-equal
$((x \not\simeq 0) \lor (y \not\simeq 0)) \rightarrow (\text{ncode-equal}(x, y) = (x = y))$

DEFINITION:
list-and-join-proof (*explist*, *pflist*)
= **if** listp (*explist*)
  then if listp (cdr (*explist*))
    then and-join-proof (car (*explist*),
      list-f-and (cdr (*explist*)),
      car (*pflist*),
      list-and-join-proof (cdr (*explist*), cdr (*pflist*)))
    else car (*pflist*) **endif**
  else *pflist* **endif**

THEOREM: list-and-join-proof-proves
(listp (*explist*) $\land$ proves-list (*pflist*, *explist*, *given*, *defns*, *symbols*))
$\rightarrow$ proves (list-and-join-proof (*explist*, *pflist*),
  list-f-and (*explist*),
  given,
  defns,
  symbols)

EVENT: Enable list-and-join-proof; name this event 'g0804'.

DEFINITION:
compose-proof (*fn*, *val*, *args*, *vals*, *pflist*, *pf*)
= list-detach-proof (list-f-equal (list (cons (*fn*, *args*),
      cons (*fn*, *vals*),
      cons (*fn*, *args*)),
    list (cons (*fn*, *args*), *val*, cons (*fn*, *vals*))),
  f-equal (cons (*fn*, *args*), *val*),
  list (ident-axiom-proof (cons (*fn*, *args*)),
    pf,
    list-detach-proof (list-f-equal (*args*, *vals*),

$$
\begin{array}{l}
\qquad\qquad\qquad \text{f-equal}\,(\text{cons}\,(fn,\ args),\\
\qquad\qquad\qquad\qquad\quad \text{cons}\,(fn,\ vals)),\\
\qquad\qquad\qquad pflist,\\
\qquad\qquad\qquad \text{equal-axiom1-proof}\,(fn,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad args,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad vals))),\\
\text{equal-axiom2-proof}\,(\text{p}\,(\texttt{0},\ \texttt{2}),\\
\qquad\qquad\qquad \text{list}\,(\text{cons}\,(fn,\ args),\ \text{cons}\,(fn,\ vals)),\\
\qquad\qquad\qquad \text{list}\,(\text{cons}\,(fn,\ args),\ val)))
\end{array}
$$

```
;;;;;;;So far we have seen how to introduce functions defined by cases.  Another
;;;;;;;important way of defining functions is by composition, i.e., "(f x1 ... xn)"
;;;;;;;can be defined as "(g (h1 x1...xn) ... (hn x1...xn))".  (It is worth noting
;;;;;;;that it is exactly because we make heavy use of function composition that
;;;;;;;we introduced function symbols corresponding to the Lisp primitives.
;;;;;;;When we were working with the predicates representing the definitions,
;;;;;;;composition was extremely tedious to simulate.)  We can of course
;;;;;;;combine definition by cases and by composition.  The remaining important
;;;;;;;mode of definition is recursion, and we will see how this is carried out, later.
;;;;;;;The lemma below asserts that if |- "(arg1=val1)", . . ., |- "(argn=valn)",
;;;;;;;and |- "((f val1 ... valn) = val)", then |- "((f arg1 ... argn) = val)".
```

THEOREM: compose-proof-proves

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{function}\,(fn)}\\
\land & \text{symb}\,(fn,\ symbols)\\
\land & \text{termp}\,(args,\ \texttt{1},\ symbols)\\
\land & \text{termp}\,(vals,\ \texttt{1},\ symbols)\\
\land & \text{termp}\,(val,\ \texttt{0},\ symbols)\\
\land & (\text{length}\,(args) = \text{degree}\,(fn))\\
\land & (\text{length}\,(vals) = \text{degree}\,(fn))\\
\land & \text{proves-list}\,(pflist,\ \text{list-f-equal}\,(args,\ vals),\ given,\ defns,\ symbols)\\
\land & \text{proves}\,(pf,\ \text{f-equal}\,(\text{cons}\,(fn,\ vals),\ val),\ given,\ defns,\ symbols)\\
\land & (concl = \text{f-equal}\,(\text{cons}\,(fn,\ args),\ val)))\\
\to & \text{proves}\,(\text{compose-proof}\,(fn,\ val,\ args,\ vals,\ pflist,\ pf),\\
 & \qquad\quad concl,\\
 & \qquad\quad given,\\
 & \qquad\quad defns,\\
 & \qquad\quad symbols)
\end{array}
$$

EVENT: Enable compose-proof; name this event ‘g0805’.

```
;;;;;;;Perhaps the most commonly used composition is the iterated use of CAR/CDR.
;;;;;;;The function CAR-CDR recurses down LIST and if (CAR LIST) is 1, it applies
;;;;;;;CAR to the result of (CAR-CDR X (CDR LIST)).  Otherwise, it returns
;;;;;;;(CDR (CAR-CDR X (CDR LIST))).
```

Definition:
car-cdr (x, *list*)
= **if** listp (*list*)
then if car (*list*) = 1 **then** car (car-cdr (x, cdr (*list*)))
else cdr (car-cdr (x, cdr (*list*))) **endif**
else x **endif**

```
;;;;;;;Z-CAR-CDR composes "Z-car" and "Z-cdr" in a manner similar to CAR-CDR.
```

Definition:
z-car-cdr (x, *list*)
= **if** listp (*list*)
then if car (*list*) = 1 **then** z-car (z-car-cdr (x, cdr (*list*)))
else z-cdr (z-car-cdr (x, cdr (*list*))) **endif**
else x **endif**

Theorem: termp-z-car-cdr
(subset (list (fn (5, 1), fn (4, 1)), *symbols*) $\land$ termp (x, 0, *symbols*))
$\rightarrow$ termp (z-car-cdr (x, *list*), 0, *symbols*)

Definition:
z-car-cdr-proof (x, *list*)
= **if** listp (*list*)
then if car (*list*) = 1
then compose-proof (fn (4, 1),
ncode (car-cdr (x, *list*)),
list (z-car-cdr (ncode (x), cdr (*list*))),
list (ncode (car-cdr (x, cdr (*list*)))),
list (z-car-cdr-proof (x, cdr (*list*))),
z-car-ncode-proof (car-cdr (x, cdr (*list*))))
else compose-proof (fn (5, 1),
ncode (car-cdr (x, *list*)),
list (z-car-cdr (ncode (x), cdr (*list*))),
list (ncode (car-cdr (x, cdr (*list*)))),
list (z-car-cdr-proof (x, cdr (*list*))),
z-cdr-ncode-proof (car-cdr (x, cdr (*list*)))) **endif**
else ident-axiom-proof (ncode (x)) **endif**

Event: Disable z-car; name this event ‘g0806’.

Event: Disable z-cdr; name this event ‘g0807’.

```
;;;;;;;Z-CAR-CDR is the Z2 analogue of CAR-CDR.
```

Theorem: z-car-cdr-proof-proves

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{subset}\,(\text{list}\,(\text{fn}\,(\mathtt{5},\ \mathtt{1}),}\\
 & \quad\quad\quad \text{fn}\,(\mathtt{4},\ \mathtt{1}),\\
 & \quad\quad\quad \text{fn}\,(\mathtt{3},\ \mathtt{1}),\\
 & \quad\quad\quad \text{fn}\,(\mathtt{2},\ \mathtt{2}),\\
 & \quad\quad\quad \text{p}\,(\mathtt{2},\ \mathtt{1}),\\
 & \quad\quad\quad \text{fn}\,(\mathtt{1},\ \mathtt{2}),\\
 & \quad\quad\quad \text{p}\,(\mathtt{1},\ \mathtt{2}),\\
 & \quad\quad\quad \text{fn}\,(\mathtt{0},\ \mathtt{0})),\\
 & \quad\quad \mathit{symbols})\\
\wedge & \text{z-car-hyps}\,(\mathit{defns},\ \mathit{symbols})\\
\wedge & \text{z-cdr-hyps}\,(\mathit{defns},\ \mathit{symbols})\\
\wedge & \text{z-succ-hyps}\,(\mathit{defns},\ \mathit{symbols})\\
\wedge & \text{z-int-hyps}\,(\mathit{defns},\ \mathit{symbols})\\
\wedge & (\mathit{concl} = \text{f-equal}\,(\text{z-car-cdr}\,(\text{ncode}\,(x),\ \mathit{list}),\ \text{ncode}\,(\text{car-cdr}\,(x,\ \mathit{list})))))\\
\rightarrow & \text{proves}\,(\text{z-car-cdr-proof}\,(x,\ \mathit{list}),\ \mathit{concl},\ \mathit{given},\ \mathit{defns},\ \mathit{symbols})
\end{array}
$$

Event: Enable z-car-cdr-proof; name this event ‘g0808’.

Definition:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{compose-car-proof}\,(\mathit{fn},\ \mathit{val},\ \mathit{arg},\ \mathit{pfval})}\\
= & \text{compose-proof}\,(\mathit{fn},\\
 & \quad\quad \mathit{val},\\
 & \quad\quad \text{list}\,(\text{z-car}\,(\text{ncode}\,(\mathit{arg}))),\\
 & \quad\quad \text{list}\,(\text{ncode}\,(\text{car}\,(\mathit{arg}))),\\
 & \quad\quad \text{list}\,(\text{z-car-ncode-proof}\,(\mathit{arg})),\\
 & \quad\quad \mathit{pfval})
\end{array}
$$

Theorem: compose-car-proof-proves

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{function}\,(\mathit{fn})}\\
\wedge & \text{symb}\,(\mathit{fn},\ \mathit{symbols})\\
\wedge & (\text{degree}\,(\mathit{fn}) = \mathtt{1})\\
\wedge & \text{termp}\,(\mathit{val},\ \mathtt{0},\ \mathit{symbols})\\
\wedge & \text{z-succ-hyps}\,(\mathit{defns},\ \mathit{symbols})\\
\wedge & \text{subset}\,(\text{list}\,(\text{fn}\,(\mathtt{4},\ \mathtt{1}),\ \text{fn}\,(\mathtt{3},\ \mathtt{1}),\ \text{fn}\,(\mathtt{2},\ \mathtt{2}),\ \text{fn}\,(\mathtt{1},\ \mathtt{2}),\ \text{p}\,(\mathtt{1},\ \mathtt{2}),\ \text{fn}\,(\mathtt{0},\ \mathtt{0})),\\
 & \quad\quad \mathit{symbols})\\
\wedge & (\text{p}\,(\mathtt{2},\ \mathtt{1}) \in \mathit{symbols})\\
\wedge & \text{z-int-hyps}\,(\mathit{defns},\ \mathit{symbols})\\
\wedge & \text{z-car-hyps}\,(\mathit{defns},\ \mathit{symbols})\\
\wedge & \text{proves}\,(\mathit{pfval},\\
 & \quad\quad \text{f-equal}\,(\text{list}\,(\mathit{fn},\ \text{ncode}\,(\text{car}\,(\mathit{arg}))),\ \mathit{val}),\\
 & \quad\quad \mathit{given},\\
 & \quad\quad \mathit{defns},\\
 & \quad\quad \mathit{symbols})
\end{array}
$$

$\wedge \quad (concl = \text{f-equal}\,(\text{list}\,(fn,\ \text{z-car}\,(\text{ncode}\,(arg))),\ val)))$
$\rightarrow \quad \text{proves}\,(\text{compose-car-proof}\,(fn,\ val,\ arg,\ pfval),\ concl,\ given,\ defns,\ symbols)$

EVENT: Enable compose-car-proof; name this event 'g0821'.

DEFINITION:
compose-car-cadr-proof (fn, val, arg, $pfval$)

$$
\begin{aligned}
= \quad & \text{compose-proof}\,(fn,\\
&\quad val,\\
&\quad \text{list}\,(\text{z-car}\,(\text{ncode}\,(arg)),\ \text{z-car}\,(\text{z-cdr}\,(\text{ncode}\,(arg)))),\\
&\quad \text{list}\,(\text{ncode}\,(\text{car}\,(arg)),\ \text{ncode}\,(\text{cadr}\,(arg))),\\
&\quad \text{list}\,(\text{z-car-ncode-proof}\,(arg),\\
&\quad\quad \text{z-car-cdr-proof}\,(arg,\ \text{list}\,(1,\ 2))),\\
&\quad pfval)
\end{aligned}
$$

THEOREM: compose-car-cadr-proof-proves

$$
\begin{aligned}
& (\text{function}\,(fn)\\
\wedge \quad & \text{symb}\,(fn,\ symbols)\\
\wedge \quad & (\text{degree}\,(fn) = 2)\\
\wedge \quad & \text{termp}\,(val,\ 0,\ symbols)\\
\wedge \quad & \text{z-succ-hyps}\,(defns,\ symbols)\\
\wedge \quad & \text{subset}\,(\text{list}\,(\text{fn}\,(5,\ 1),\\
& \quad \text{fn}\,(4,\ 1),\\
& \quad \text{fn}\,(3,\ 1),\\
& \quad \text{fn}\,(2,\ 2),\\
& \quad \text{fn}\,(1,\ 2),\\
& \quad \text{p}\,(1,\ 2),\\
& \quad \text{fn}\,(0,\ 0)),\\
& \quad symbols)\\
\wedge \quad & (\text{p}\,(2,\ 1) \in symbols)\\
\wedge \quad & \text{z-int-hyps}\,(defns,\ symbols)\\
\wedge \quad & \text{z-car-hyps}\,(defns,\ symbols)\\
\wedge \quad & \text{z-cdr-hyps}\,(defns,\ symbols)\\
\wedge \quad & \text{proves}\,(pfval,\\
& \quad \text{f-equal}\,(\text{list}\,(fn,\ \text{ncode}\,(\text{car}\,(arg)),\ \text{ncode}\,(\text{cadr}\,(arg))),\ val),\\
& \quad given,\\
& \quad defns,\\
& \quad symbols)\\
\wedge \quad & (concl = \text{f-equal}\,(\text{list}\,(fn,\\
& \quad\quad \text{z-car}\,(\text{ncode}\,(arg)),\\
& \quad\quad \text{z-car}\,(\text{z-cdr}\,(\text{ncode}\,(arg)))),\\
& \quad val)))\\
\rightarrow \quad & \text{proves}\,(\text{compose-car-cadr-proof}\,(fn,\ val,\ arg,\ pfval),\\
& \quad concl,
\end{aligned}
$$

given,
defns,
symbols)

EVENT: Enable compose-car-cadr-proof; name this event 'g0822'.

```
;;;;;;;Next we would like to define the Z2 analogue of APPLY-SUBR, the part
;;;;;;;of the Lisp interpreter that evaluates the Lisp primitives.  This is
;;;;;;;done in three parts to make the proof manageable.  In the first part
;;;;;;;we represent APPLY-SUBR1 which evaluates LISTP, CONS, CAR, and CDR.
;;;;;;;These operations are represented by the symbols 6, 7, 8, and 9, respectively,
;;;;;;;to the Lisp interpreter.  APPLY-SUBR1 is represented by "(Z-apply-subr1 f args)".
```

DEFINITION: z-apply-subr1 (*fn*, *args*) = list (fn (13, 2), *fn*, *args*)

DEFINITION:
z-apply-subr1-defn (*fn*, *args*, *y*)
= f-if (f-equal (*fn*, ncode (6)),
  f-equal (*y*, z-listp (z-car (*args*))),
  f-if (f-equal (*fn*, ncode (7)),
   f-equal (*y*, z-cons (z-car (*args*), z-car (z-cdr (*args*)))),
   f-if (f-equal (*fn*, ncode (8)),
    f-equal (*y*, z-car (z-car (*args*))),
    f-equal (*y*, z-cdr (z-car (*args*))))))

DEFINITION:
z-apply-subr1-exis-proof(*fn*, *args*, *y*)
= term-z-if-exis-proof(*y*,
  z-if (f-equal (*fn*, ncode (6)),
   z-listp (z-car (*args*)),
   z-if (f-equal (*fn*, ncode (7)),
    z-cons (z-car (*args*), z-car (z-cdr (*args*))),
    z-if (f-equal (*fn*, ncode (8)),
     z-car (z-car (*args*)),
     z-cdr (z-car (*args*))))))

DEFINITION:
z-apply-subr1-unique-proof(*y*, *y1*, *fn*, *args*)
= term-z-if-unique-proof(*y*,
  y1,
  z-if (f-equal (*fn*, ncode (6)),
   z-listp (z-car (*args*)),
   z-if (f-equal (*fn*, ncode (7)),
    z-cons (z-car (*args*),

```
                                        z-car (z-cdr (args))),
                                   z-if (f-equal (fn, ncode (8)),
                                      z-car (z-car (args)),
                                      z-cdr (z-car (args))))))
```

DEFINITION:

```
z-apply-subr1-defn-proof (fn, args)
=   instance-proof (z-apply-subr1-defn (0, 1, z-apply-subr1 (0, 1)),
                    list (0, 1),
                    list (fn, args),
                    fun-defn-proof (fn (13, 2),
                                    list (0, 1),
                                    2,
                                    3,
                                    z-apply-subr1-defn (0, 1, 2),
                                    z-apply-subr1-exis-proof (0, 1, 2),
                                    z-apply-subr1-unique-proof (2, 3, 0, 1)))
```

DEFINITION:

```
z-apply-subr1-hyps (defns, symbols)
=   ((assoc (fn (13, 2), defns)
      =   z-apply-subr1-defn (0, 1, z-apply-subr1 (0, 1)))
     ∧   subset (list (fn (0, 0),
                      fn (3, 1),
                      fn (4, 1),
                      fn (5, 1),
                      fn (6, 1),
                      fn (7, 1),
                      fn (8, 1),
                      fn (9, 1),
                      fn (10, 1),
                      fn (11, 2),
                      fn (12, 2)),
                 rest-of (fn (13, 2), symbols)))
```

THEOREM: z-apply-subr1-defn-proof-proves

```
(subset (list (fn (13, 2),
               fn (12, 2),
               fn (0, 0),
               fn (3, 1),
               fn (4, 1),
               fn (5, 1),
               fn (6, 1),
               fn (7, 1),
               fn (8, 1),
```

fn (9, 1),
fn (10, 1),
fn (11, 2)),
symbols)
$\wedge$ z-apply-subr1-hyps (*defns*, *symbols*)
$\wedge$ termp (list (x, y), 1, *symbols*)
$\wedge$ (*concl* = z-apply-subr1-defn (x, y, z-apply-subr1 (x, y))))
$\rightarrow$ proves (z-apply-subr1-defn-proof (x, y), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-apply-subr1-defn-proof; name this event 'g0810'.

EVENT: Enable z-apply-subr1-hyps; name this event 'g0811'.

```
;;;;;;;APPLY-SUBR2 evaluates the Lisp primitives ZEROP, ADD1, SUB1, and those
;;;;;;;evaluated by APPLY-SUBR1.  It is represented by "(Z-apply-subr2 f args)".
```

DEFINITION: z-apply-subr2 (*fn*, *args*) = list (fn (14, 2), *fn*, *args*)

DEFINITION:
z-apply-subr2-defn (*fn*, *args*, *y*)
= f-if (f-equal (*fn*, ncode (3)),
f-equal (y, z-zerop (z-car (*args*))),
f-if (f-equal (*fn*, ncode (4)),
f-equal (y, z-add1 (z-car (*args*))),
f-if (f-equal (*fn*, ncode (5)),
f-equal (y, z-sub1 (z-car (*args*))),
f-equal (y, z-apply-subr1 (*fn*, *args*)))))

DEFINITION:
z-apply-subr2-exis-proof (*fn*, *args*, *y*)
= term-z-if-exis-proof (y,
z-if (f-equal (*fn*, ncode (3)),
z-zerop (z-car (*args*)),
z-if (f-equal (*fn*, ncode (4)),
z-add1 (z-car (*args*)),
z-if (f-equal (*fn*, ncode (5)),
z-sub1 (z-car (*args*)),
z-apply-subr1 (*fn*, *args*)))))

DEFINITION:
z-apply-subr2-unique-proof (*fn*, *args*, *y*, *y1*)
= term-z-if-unique-proof (y,
y1,
z-if (f-equal (*fn*, ncode (3)),

```
                          z-zerop (z-car (args)),
                          z-if (f-equal (fn, ncode (4)),
                               z-add1 (z-car (args)),
                               z-if (f-equal (fn, ncode (5)),
                                    z-sub1 (z-car (args)),
                                    z-apply-subr1 (fn, args)))))
```

DEFINITION:

```
z-apply-subr2-defn-proof (fn, args)
=   instance-proof (z-apply-subr2-defn (0, 1, z-apply-subr2 (0, 1)),
                    list (0, 1),
                    list (fn, args),
                    fun-defn-proof (fn (14, 2),
                                    list (0, 1),
                                    2,
                                    3,
                                    z-apply-subr2-defn (0, 1, 2),
                                    z-apply-subr2-exis-proof (0, 1, 2),
                                    z-apply-subr2-unique-proof (0, 1, 2, 3)))
```

DEFINITION:

```
z-apply-subr2-hyps (defns, symbols)
=   ((assoc (fn (14, 2), defns)
      =   z-apply-subr2-defn (0, 1, z-apply-subr2 (0, 1)))
     ∧   subset (list (fn (0, 0),
                      fn (3, 1),
                      fn (4, 1),
                      fn (5, 1),
                      fn (6, 1),
                      fn (7, 1),
                      fn (8, 1),
                      fn (9, 1),
                      fn (10, 1),
                      fn (11, 2),
                      fn (12, 2),
                      fn (13, 2)),
                 rest-of (fn (14, 2), symbols)))
```

THEOREM: z-apply-subr2-defn-proof-proves

```
 (subset (list (fn (14, 2),
                fn (13, 2),
                fn (12, 2),
                fn (0, 0),
                fn (3, 1),
                fn (4, 1),
```

fn (5, 1),
fn (6, 1),
fn (7, 1),
fn (8, 1),
fn (9, 1),
fn (10, 1),
fn (11, 2)),
symbols)
∧ z-apply-subr2-hyps (*defns*, *symbols*)
∧ termp (list (x, y), 1, *symbols*)
∧ (*concl* = z-apply-subr2-defn (x, y, z-apply-subr2 (x, y))))
→ proves (z-apply-subr2-defn-proof (x, y), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-apply-subr2-defn-proof; name this event 'g0812'.

EVENT: Enable z-apply-subr2-hyps; name this event 'g0813'.

```
;;;;;;;Finally, APPLY-SUBR evaluates the Lisp primitives EQUAL, NUMBERP,
;;;;;;;and those evaluated by APPLY-SUBR2.  It is represented by
;;;;;;;"(Z-apply-subr f args)".
```

DEFINITION: z-apply-subr (*fn*, *args*) = list (fn (15, 2), *fn*, *args*)

DEFINITION:
z-apply-subr-defn (*fn*, *args*, y)
= f-if (f-equal (*fn*, ncode (1)),
f-equal (y, z-equal (z-car (*args*), z-car (z-cdr (*args*)))),
f-if (f-equal (*fn*, ncode (2)),
f-equal (y, z-numberp (z-car (*args*))),
f-equal (y, z-apply-subr2 (*fn*, *args*))))

DEFINITION:
z-apply-subr-exis-proof (*fn*, *args*, y)
= term-z-if-exis-proof (y,
z-if (f-equal (*fn*, ncode (1)),
z-equal (z-car (*args*), z-car (z-cdr (*args*))),
z-if (f-equal (*fn*, ncode (2)),
z-numberp (z-car (*args*)),
z-apply-subr2 (*fn*, *args*))))

DEFINITION:
z-apply-subr-unique-proof (*fn*, *args*, y, *y1*)
= term-z-if-unique-proof (y,
y1,

```
                        z-if (f-equal (fn, ncode (1)),
                              z-equal (z-car (args), z-car (z-cdr (args))),
                              z-if (f-equal (fn, ncode (2)),
                                    z-numberp (z-car (args)),
                                    z-apply-subr2 (fn, args))))
```

Definition:
z-apply-subr-defn-proof (*fn*, *args*)

```
=   instance-proof (z-apply-subr-defn (0, 1, z-apply-subr (0, 1)),
                    list (0, 1),
                    list (fn, args),
                    fun-defn-proof (fn (15, 2),
                                    list (0, 1),
                                    2,
                                    3,
                                    z-apply-subr-defn (0, 1, 2),
                                    z-apply-subr-exis-proof (0, 1, 2),
                                    z-apply-subr-unique-proof (0, 1, 2, 3)))
```

Definition:
z-apply-subr-hyps (*defns*, *symbols*)

```
=   ((assoc (fn (15, 2), defns)
      =   z-apply-subr-defn (0, 1, z-apply-subr (0, 1)))
     ∧   subset (list (fn (0, 0),
                       fn (3, 1),
                       fn (4, 1),
                       fn (5, 1),
                       fn (6, 1),
                       fn (7, 1),
                       fn (8, 1),
                       fn (9, 1),
                       fn (10, 1),
                       fn (11, 2),
                       fn (12, 2),
                       fn (13, 2),
                       fn (14, 2)),
                 rest-of (fn (15, 2), symbols)))
```

Theorem: z-apply-subr-defn-proof-proves

```
 (subset (list (fn (15, 2),
                fn (14, 2),
                fn (13, 2),
                fn (12, 2),
                fn (0, 0),
                fn (3, 1),
```

```
                  fn (4, 1),
                  fn (5, 1),
                  fn (6, 1),
                  fn (7, 1),
                  fn (8, 1),
                  fn (9, 1),
                  fn (10, 1),
                  fn (11, 2)),
              symbols)
  ∧   z-apply-subr-hyps (defns, symbols)
  ∧   termp (list (x, y), 1, symbols)
  ∧   (concl = z-apply-subr-defn (x, y, z-apply-subr (x, y))))
 →    proves (z-apply-subr-defn-proof (x, y), concl, given, defns, symbols)
```

EVENT: Enable z-apply-subr-defn-proof; name this event 'g0814'.

EVENT: Enable z-apply-subr-hyps; name this event 'g0815'.

DEFINITION:

```
apply-subr1 (fn, args)
=    if fn = 6 then bool-fix (listp (car (args)))
     elseif fn = 7 then cons (car (args), cadr (args))
     elseif fn = 8 then car (car (args))
     else cdr (car (args)) endif
```

DEFINITION:

```
z-apply-subr1-ncode-proof (fn, args)
=    z-if-eval-proof1 (list (list (fn = 6,
                                   ident-axiom-proof (ncode (6)),
                                   ncode-nequal-proof (fn, 6)),
                              compose-car-proof (fn (6, 1),
                                                 ncode (bool-fix (listp (car (args)))),
                                                 args,
                                                 z-listp-ncode-proof (car (args))),
                              list (list (fn = 7,
                                          ident-axiom-proof (ncode (7)),
                                          ncode-nequal-proof (fn, 7)),
                                    compose-car-cadr-proof (fn (12, 2),
                                                            ncode (cons (car (args),
                                                                         cadr (args))),
                                                            args,
                                                            z-cons-defn-proof (ncode (car (args)),
                                                                               ncode (cadr (args)))),
```

```
                      list (list (fn = 8,
                                  ident-axiom-proof (ncode (8)),
                                  ncode-nequal-proof (fn, 8)),
                            compose-car-proof (fn (4, 1),
                                               ncode (car (car (args))),
                                               args,
                                               z-car-ncode-proof (car (args))),
                            compose-car-proof (fn (5, 1),
                                               ncode (cdr (car (args))),
                                               args,
                                               z-cdr-ncode-proof (car (args)))))),
               z-if (f-equal (ncode (fn), ncode (6)),
                     z-listp (z-car (ncode (args))),
                     z-if (f-equal (ncode (fn), ncode (7)),
                           z-cons (z-car (ncode (args)),
                                   z-car (z-cdr (ncode (args)))),
                           z-if (f-equal (ncode (fn), ncode (8)),
                                 z-car (z-car (ncode (args))),
                                 z-cdr (z-car (ncode (args)))))),
               z-apply-subr1 (ncode (fn), ncode (args)),
               ncode (apply-subr1 (fn, args)),
               z-apply-subr1-defn-proof (ncode (fn), ncode (args)))
```

THEOREM: z-apply-subr1-ncode-proof-proves-step

```
 (subset (list (fn (13, 2),
                fn (12, 2),
                fn (0, 0),
                fn (3, 1),
                fn (4, 1),
                fn (5, 1),
                fn (6, 1),
                fn (7, 1),
                fn (8, 1),
                fn (9, 1),
                fn (10, 1),
                fn (11, 2),
                fn (1, 2),
                fn (2, 2),
                p (2, 1),
                p (1, 2)),
         symbols)
  ∧  z-apply-subr-hyps (defns, symbols)
  ∧  z-int-hyps (defns, symbols)
  ∧  z-apply-subr1-hyps (defns, symbols)
```

$\land$ z-succ-hyps (*defns*, *symbols*)
$\land$ z-car-hyps (*defns*, *symbols*)
$\land$ z-cdr-hyps (*defns*, *symbols*)
$\land$ z-cons-hyps (*defns*, *symbols*)
$\land$ z-listp-hyps (*defns*, *symbols*)
$\land$ (*concl* = f-equal (z-apply-subr1 (ncode (*fn*), ncode (*args*)),
ncode (apply-subr1 (*fn*, *args*)))))
$\rightarrow$ proves (z-apply-subr1-ncode-proof (*fn*, *args*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-apply-subr1-ncode-proof; name this event 'g0820'.

EVENT: Enable apply-subr1; name this event 'g0825'.

EVENT: Enable sub1; name this event 'g0826'.

DEFINITION:
apply-subr2 (*fn*, *args*)
= **if** *fn* = `3` **then** bool-fix (car (*args*) $\simeq$ `0`)
elseif *fn* = `4` **then** 1 + car (*args*)
elseif *fn* = `5` **then** car (*args*) $-$ 1
else apply-subr1 (*fn*, *args*) **endif**

DEFINITION:
z-apply-subr2-ncode-proof (*fn*, *args*)
= z-if-eval-proof1 (list (list (*fn* = `3`,
ident-axiom-proof (ncode (`3`)),
ncode-nequal-proof (*fn*, `3`)),
compose-car-proof (fn (`9`, `1`),
ncode (bool-fix (car (*args*) $\simeq$ `0`)),
args,
z-zerop-ncode-proof (car (*args*))),
list (list (*fn* = `4`,
ident-axiom-proof (ncode (`4`)),
ncode-nequal-proof (*fn*, `4`)),
compose-car-proof (fn (`7`, `1`),
ncode (1 + car (*args*)),
args,
z-add1-ncode-proof (car (*args*))),
list (list (*fn* = `5`,
ident-axiom-proof (ncode (`5`)),
ncode-nequal-proof (*fn*, `5`)),
compose-car-proof (fn (`10`, `1`),
ncode (car (*args*) $-$ 1),

```
                                                       args,
                                                       z-sub1-ncode-proof (car (args))),
                                         z-apply-subr1-ncode-proof (fn, args)))),
                                 z-if (f-equal (ncode (fn), ncode (3)),
                                    z-zerop (z-car (ncode (args))),
                                    z-if (f-equal (ncode (fn), ncode (4)),
                                       z-add1 (z-car (ncode (args))),
                                       z-if (f-equal (ncode (fn), ncode (5)),
                                          z-sub1 (z-car (ncode (args))),
                                          z-apply-subr1 (ncode (fn), ncode (args))))),
                                 z-apply-subr2 (ncode (fn), ncode (args)),
                                 ncode (apply-subr2 (fn, args)),
                                 z-apply-subr2-defn-proof (ncode (fn), ncode (args)))
```

THEOREM: z-apply-subr2-ncode-proof-proves-step

```
 (subset (list (fn (14, 2),
                fn (13, 2),
                fn (12, 2),
                fn (0, 0),
                fn (3, 1),
                fn (4, 1),
                fn (5, 1),
                fn (6, 1),
                fn (7, 1),
                fn (8, 1),
                fn (9, 1),
                fn (10, 1),
                fn (11, 2),
                fn (1, 2),
                fn (2, 2),
                p (2, 1),
                p (1, 2)),
          symbols)
  ∧  z-apply-subr-hyps (defns, symbols)
  ∧  z-apply-subr2-hyps (defns, symbols)
  ∧  z-int-hyps (defns, symbols)
  ∧  z-apply-subr1-hyps (defns, symbols)
  ∧  z-succ-hyps (defns, symbols)
  ∧  z-car-hyps (defns, symbols)
  ∧  z-cdr-hyps (defns, symbols)
  ∧  z-cons-hyps (defns, symbols)
  ∧  z-listp-hyps (defns, symbols)
  ∧  z-add1-hyps (defns, symbols)
  ∧  z-sub1-hyps (defns, symbols)
```

$\wedge$ z-zerop-hyps (*defns*, *symbols*)
$\wedge$ (*concl* = f-equal (z-apply-subr2 (ncode (*fn*), ncode (*args*)),
ncode (apply-subr2 (*fn*, *args*)))))
$\rightarrow$ proves (z-apply-subr2-ncode-proof (*fn*, *args*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-apply-subr2-ncode-proof; name this event 'g0827'.

EVENT: Enable apply-subr2; name this event 'g0828'.

DEFINITION:
apply-subr (*fn*, *args*)
= **if** *fn* = 1 **then** bool-fix (ncode-equal (car (*args*), cadr (*args*)))
elseif *fn* = 2 **then** bool-fix ($\neg$ listp (car (*args*)))
else apply-subr2 (*fn*, *args*) **endif**

DEFINITION:
z-apply-subr-ncode-proof (*fn*, *args*)
= z-if-eval-proof1 (list (list (*fn* = 1,
ident-axiom-proof (ncode (1)),
ncode-nequal-proof (*fn*, 1)),
compose-car-cadr-proof (fn (11, 2),
ncode (bool-fix (ncode-equal (car (*args*),
cadr (*args*)))),
args,
z-equal-ncode-proof (car (*args*),
cadr (*args*))),
list (list (*fn* = 2,
ident-axiom-proof (ncode (2)),
ncode-nequal-proof (*fn*, 2)),
compose-car-proof (fn (8, 1),
ncode (bool-fix ($\neg$ listp (car (*args*)))),
args,
z-numberp-ncode-proof (car (*args*))),
z-apply-subr2-ncode-proof (*fn*, *args*))),
z-if (f-equal (ncode (*fn*), ncode (1)),
z-equal (z-car (ncode (*args*)),
z-car (z-cdr (ncode (*args*)))),
z-if (f-equal (ncode (*fn*), ncode (2)),
z-numberp (z-car (ncode (*args*))),
z-apply-subr2 (ncode (*fn*), ncode (*args*)))),
z-apply-subr (ncode (*fn*), ncode (*args*)),
ncode (apply-subr (*fn*, *args*)),
z-apply-subr-defn-proof (ncode (*fn*), ncode (*args*)))

DEFINITION:
z-apply-subr-all-hyps (*given*, *defns*, *symbols*)
= (z-apply-subr-hyps (*defns*, *symbols*)
∧ z-apply-subr2-hyps (*defns*, *symbols*)
∧ z-int-hyps (*defns*, *symbols*)
∧ z-apply-subr1-hyps (*defns*, *symbols*)
∧ z-succ-hyps (*defns*, *symbols*)
∧ z-car-hyps (*defns*, *symbols*)
∧ z-cdr-hyps (*defns*, *symbols*)
∧ z-cons-hyps (*defns*, *symbols*)
∧ z-listp-hyps (*defns*, *symbols*)
∧ z-add1-hyps (*defns*, *symbols*)
∧ z-sub1-hyps (*defns*, *symbols*)
∧ z-zerop-hyps (*defns*, *symbols*)
∧ z-equal-hyps (*defns*, *symbols*)
∧ z-numberp-hyps (*defns*, *symbols*))

THEOREM: z-apply-subr-ncode-proof-proves
(subset (list (fn (15, 2),
fn (14, 2),
fn (13, 2),
fn (12, 2),
fn (0, 0),
fn (3, 1),
fn (4, 1),
fn (5, 1),
fn (6, 1),
fn (7, 1),
fn (8, 1),
fn (9, 1),
fn (10, 1),
fn (11, 2),
fn (1, 2),
fn (2, 2),
p (2, 1),
p (1, 2)),
symbols)
∧ z-apply-subr-all-hyps (*given*, *defns*, *symbols*)
∧ (*concl* = f-equal (z-apply-subr (ncode (*fn*), ncode (*args*)),
ncode (apply-subr (*fn*, *args*)))))
→ proves (z-apply-subr-ncode-proof (*fn*, *args*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-apply-subr-ncode-proof; name this event ‘g0829’.

EVENT: Enable apply-subr; name this event 'g0830'.

```
;;;;;;;Thus far we have shown how we can derive Z2 analogues of the Lisp primitives
;;;;;;;and the function APPLY-SUBR.  These functions will be used in demonstrating
;;;;;;;the representability of the Lisp interpreter.  A major obstacle that has
;;;;;;;to be overcome is the representability (in Z2) of recursively defined Lisp functions.
;;;;;;;These will be represented by specifying the partial graph (the set of value/argument
;;;;;;;ordered pairs used in the computation of the value of the function at
;;;;;;;a specific argument).  This partial graph will first be constructed as a
;;;;;;;list of pairs for the computation of the Lisp function and then translated
;;;;;;;into the corresponding set.  The specification of the partial graph in Z2 will be
;;;;;;;shown to hold of this translation.  It will also be shown that any set
;;;;;;;satisfying the specification.
;;;;;;;The function GRAPH-TRANS below translates a list into the corresponding set
;;;;;;;so that the list (X1 . (X2 . ... (Xn . NIL))) is converted to the set
;;;;;;; "{<X1>, <X2>, ..., <Xn>}".  The translation of list X will be represented as [X].
```

DEFINITION:
graph-trans (x)

```
=   if listp (x)  then z-union (z-sing (ncode (car (x))), graph-trans (cdr (x)))
    else PHI endif
```

DEFINITION:
in-graph-proof (x, *graph*)

```
=   if listp (graph)
    then if x = car (graph)
         then f-iff-mp-proof2 (isin (ncode (x),
                                     z-union (z-sing (ncode (x)),
                                              graph-trans (cdr (graph)))),
                               f-or (isin (ncode (x), z-sing (ncode (x))),
                                     isin (ncode (x), graph-trans (cdr (graph)))),
                               union-axiom-proof (ncode (x),
                                                  z-sing (ncode (x)),
                                                  graph-trans (cdr (graph))),
                               rt-expan-proof (isin (ncode (x),
                                                     z-sing (ncode (x))),
                                               isin (ncode (x),
                                                     graph-trans (cdr (graph))),
                                               isin-sing1-proof (ncode (x))))
         else f-iff-mp-proof2 (isin (ncode (x),
                                     z-union (z-sing (ncode (car (graph))),
                                              graph-trans (cdr (graph)))),
                               f-or (isin (ncode (x),
                                           z-sing (ncode (car (graph)))),
```

isin (ncode (x),
graph-trans (cdr (*graph*)))),
union-axiom-proof (ncode (x),
z-sing (ncode (car (*graph*))),
graph-trans (cdr (*graph*))),
expan-proof (isin (ncode (x),
z-sing (ncode (car (*graph*)))),
isin (ncode (x),
graph-trans (cdr (*graph*))),
in-graph-proof (x,
cdr (*graph*)))) **endif**

else nil endif

THEOREM: termp-graph-trans
((fn (0, 0) $\in$ *symbols*)
$\land$ (fn (3, 1) $\in$ *symbols*)
$\land$ (fn (1, 2) $\in$ *symbols*)
$\land$ (fn (2, 2) $\in$ *symbols*))
$\to$ termp (graph-trans (x), 0, *symbols*)

```
;;;;;;;If X is in GRAPH, then |- "(<X> in [GRAPH])".
```

THEOREM: in-graph-proof-proves
((fn (0, 0) $\in$ *symbols*)
$\land$ (fn (3, 1) $\in$ *symbols*)
$\land$ (fn (1, 2) $\in$ *symbols*)
$\land$ (fn (2, 2) $\in$ *symbols*)
$\land$ (p (1, 2) $\in$ *symbols*)
$\land$ ($x \in$ *graph*)
$\land$ (*x-bar* = ncode (x))
$\land$ (*trace* = graph-trans (*graph*)))
$\to$ proves (in-graph-proof (x, *graph*), isin (*x-bar*, *trace*), *given*, *defns*, *symbols*)

EVENT: Enable in-graph-proof; name this event 'g0831'.

DEFINITION:
opair-equal-proof (x, y, u, v)
= tautconseq-proof (list (f-implies (f-equal (x, u),
f-implies (f-equal (y, v),
f-equal (z-pair (x, y),
z-pair (u, v)))),
f-implies (f-equal (x, u),
f-implies (f-equal (x, u),
f-equal (z-sing (x), z-sing (u)))),

$$
\begin{array}{l}
\quad\text{f-implies}\,(\text{f-equal}\,(\text{z-sing}\,(x),\ \text{z-sing}\,(u)),\\
\qquad\qquad\text{f-implies}\,(\text{f-equal}\,(\text{z-pair}\,(x,\ y),\\
\qquad\qquad\qquad\qquad\qquad\text{z-pair}\,(u,\ v)),\\
\qquad\qquad\qquad\quad\text{f-equal}\,(\text{z-opair}\,(x,\ y),\\
\qquad\qquad\qquad\qquad\qquad\text{z-opair}\,(u,\ v))))),\\
\text{f-implies}\,(\text{f-equal}\,(x,\ u),\\
\qquad\text{f-implies}\,(\text{f-equal}\,(y,\ v),\\
\qquad\qquad\text{f-equal}\,(\text{z-opair}\,(x,\ y),\\
\qquad\qquad\qquad\text{z-opair}\,(u,\ v)))),\\
\text{list}\,(\text{equal-axiom1-proof}\,(\text{fn}\,(\mathtt{1},\ \mathtt{2}),\ \text{list}\,(x,\ y),\ \text{list}\,(u,\ v)),\\
\quad\text{equal-axiom1-proof}\,(\text{fn}\,(\mathtt{1},\ \mathtt{2}),\ \text{list}\,(x,\ x),\ \text{list}\,(u,\ u)),\\
\quad\text{equal-axiom1-proof}\,(\text{fn}\,(\mathtt{1},\ \mathtt{2}),\\
\qquad\qquad\qquad\text{list}\,(\text{z-sing}\,(x),\ \text{z-pair}\,(x,\ y)),\\
\qquad\qquad\qquad\text{list}\,(\text{z-sing}\,(u),\ \text{z-pair}\,(u,\ v)))))
\end{array}
$$

Event: Disable z-opair; name this event ‘g0832’.

Event: Disable z-sing; name this event ‘g0833’.

Theorem: opair-equal-proof-proves

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{fn}\,(\mathtt{1},\ \mathtt{2}) \in \mathit{symbols})}\\
\ \wedge & \text{termp}\,(x,\ \mathtt{0},\ \mathit{symbols})\\
\ \wedge & \text{termp}\,(y,\ \mathtt{0},\ \mathit{symbols})\\
\ \wedge & \text{termp}\,(u,\ \mathtt{0},\ \mathit{symbols})\\
\ \wedge & \text{termp}\,(v,\ \mathtt{0},\ \mathit{symbols}))\\
\rightarrow & \text{proves}\,(\text{opair-equal-proof}\,(x,\ y,\ u,\ v),\\
& \qquad\text{f-implies}\,(\text{f-equal}\,(x,\ u),\\
& \qquad\qquad\text{f-implies}\,(\text{f-equal}\,(y,\ v),\\
& \qquad\qquad\qquad\text{f-equal}\,(\text{z-opair}\,(x,\ y),\ \text{z-opair}\,(u,\ v)))),\\
& \qquad\mathit{given},\\
& \qquad\mathit{defns},\\
& \qquad\mathit{symbols})
\end{array}
$$

Event: Enable z-sing; name this event ‘g0834’.

Event: Enable z-opair; name this event ‘g0835’.

Event: Enable opair-equal-proof; name this event ‘g0836’.

Definition:
in-graph-trans-proof $(x,\ y,\ z)$
$=\quad$ tautconseq-proof (list (f-iff (isin $(x$, graph-trans (cons $(y,\ z))),$

```
                    f-or (isin (x, z-sing (ncode (y))),
                          isin (x, graph-trans (z)))),
               f-iff (isin (x, z-sing (ncode (y))),
                      f-equal (x, ncode (y)))),
          f-iff (isin (x, graph-trans (cons (y, z))),
                 f-or (f-equal (x, ncode (y)),
                       isin (x, graph-trans (z)))),
          list (union-axiom-proof (x,
                                   z-sing (ncode (y)),
                                   graph-trans (z)),
                isin-sing-proof (ncode (y), x)))
```

Definition:

```
isin-graph-trans-proof (x, y)
=    f-iff-mp-proof2 (isin (ncode (x), graph-trans (cons (x, y))),
                      f-or (f-equal (ncode (x), ncode (x)),
                            isin (ncode (x), graph-trans (y))),
                      in-graph-trans-proof (ncode (x), x, y),
                      rt-expan-proof (f-equal (ncode (x), ncode (x)),
                                      isin (ncode (x), graph-trans (y)),
                                      ident-axiom-proof (ncode (x))))
```

Definition:

```
graph-trans-merge-proof (x, y, exp1, exp2, pf1, pf2)
=    tautconseq-proof (list (f-iff (isin (exp1, graph-trans (cons (x, y))),
                                    f-or (f-equal (exp1, ncode (x)),
                                          isin (exp1, graph-trans (y)))),
                             f-implies (f-equal (exp1, ncode (x)), exp2),
                             f-implies (isin (exp1, graph-trans (y)), exp2)),
                       f-implies (isin (exp1, graph-trans (cons (x, y))), exp2),
                       list (in-graph-trans-proof (exp1, x, y), pf1, pf2))
```

```
;;;;;;; |- "((x in [(CONS Y Z)]) <-> ((x = <Y>) v (x in [Z])))".
```

Theorem: in-graph-trans-proof-proves

```
 (subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), symbols)
  ∧    termp (x, 0, symbols)
  ∧    (y-bar = ncode (y))
  ∧    (z-bar = graph-trans (z)))
 →    proves (in-graph-trans-proof (x, y, z),
              f-iff (isin (x, graph-trans (cons (y, z))),
                     f-or (f-equal (x, y-bar), isin (x, z-bar))),
              given,
              defns,
              symbols)
```

EVENT: Enable in-graph-trans-proof; name this event ‘g0488’.

```
;;;;;;; |- "(<X> in [(CONS X Y)])".
```

THEOREM: isin-graph-trans-proof-proves
((*x-bar* = ncode (*x*))
 ∧ subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), *symbols*))
→ proves (isin-graph-trans-proof (*x*, *y*),
  isin (*x-bar*, graph-trans (cons (*x*, *y*))),
  given,
  defns,
  symbols)

```
;;;;;;; If |- "((a = <X>) -> B)", and |- "((a in [Y]) -> B)", then |- "((a in [CONS X Y]) -> B)".
;;;;;;;This lemma is really important because we prove that every member of the
;;;;;;;partial graph satisfies the specification, by induction on the list from which
;;;;;;;the graph is constructed.  This lemma permits the case-split required to carry
;;;;;;;out such an induction.  This will be clarified later.
```

THEOREM: graph-trans-merge
(subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), *symbols*)
 ∧ termp (*exp1*, 0, *symbols*)
 ∧ formula (*exp2*, *symbols*)
 ∧ proves (*pf1*,
  f-implies (f-equal (*exp1*, ncode (*x*)), *exp2*),
  given,
  defns,
  symbols)
 ∧ proves (*pf2*,
  f-implies (isin (*exp1*, graph-trans (*y*)), *exp2*),
  given,
  defns,
  symbols))
→ proves (graph-trans-merge-proof (*x*, *y*, *exp1*, *exp2*, *pf1*, *pf2*),
  f-implies (isin (*exp1*, graph-trans (cons (*x*, *y*))), *exp2*),
  given,
  defns,
  symbols)

EVENT: Enable isin-graph-trans-proof; name this event ‘g0490’.

EVENT: Enable graph-trans-merge-proof; name this event ‘g0491’.

THEOREM: nlistp-graph-trans

$(x \simeq \mathbf{nil}) \rightarrow (\text{graph-trans}\,(x) = \text{PHI})$

THEOREM: collect-free-graph-trans
$\text{collect-free}\,(\text{graph-trans}\,(x),\ \mathtt{0}) = \mathbf{nil}$

EVENT: Enable graph-trans; name this event 'g0459'.

DEFINITION:
in-graph-step-proof (exp, x, y, pf)
= f-iff-mp-proof2 (isin (exp, graph-trans (cons (x, y))),
f-or (f-equal (exp, ncode (x)), isin (exp, graph-trans (y))),
in-graph-trans-proof (exp, x, y),
expan-proof (f-equal (exp, ncode (x)),
isin (exp, graph-trans (y)),
pf))

```
;;;;;;; If |- "(exp in [Y])", then |- "(exp in [(CONS X Y)])".
```

THEOREM: in-graph-step-proof-proves
(subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), $symbols$)
$\wedge$ termp (exp, 0, $symbols$)
$\wedge$ proves (pf, isin (exp, graph-trans (y)), $given$, $defns$, $symbols$)
$\wedge$ ($trace$ = graph-trans (cons (x, y))))
$\rightarrow$ proves (in-graph-step-proof (exp, x, y, pf),
isin (exp, $trace$),
$given$,
$defns$,
$symbols$)

EVENT: Enable in-graph-step-proof; name this event 'g0837'.

```
;;;;;;;Z-LIST converts a list of the form (X1. (... (Xn . NIL))) to <<X1>, ..., <Xn>>.
```

DEFINITION:
z-list (exp)
= **if** listp (exp)
then if listp (cdr (exp)) **then** z-opair (car (exp), z-list (cdr (exp)))
else car (exp) **endif**
else nil endif

THEOREM: termp-z-list
((fn (1, 2) $\in$ $symbols$) $\wedge$ listp ($list$) $\wedge$ termp ($list$, 1, $symbols$))
$\rightarrow$ termp (z-list ($list$), 0, $symbols$)

Definition:

```
z-list-equal-proof(list1, list2, pf)
=   if listp (list1) ∧ listp (list2)
    then if listp (cdr (list1)) ∧ listp (cdr (list2))
         then and-join-proof (f-equal (car (list1), car (list2)),
                              list-f-and (list-f-equal (cdr (list1),
                                                        cdr (list2))),
                              detach-proof (f-equal (z-list (list1),
                                                     z-list (list2)),
                                            f-equal (car (list1), car (list2)),
                                            pf,
                                            z-car-unique-proof (car (list1),
                                                                z-list (cdr (list1)),
                                                                car (list2),
                                                                z-list (cdr (list2)))),
                              z-list-equal-proof (cdr (list1),
                                                  cdr (list2),
                                                  detach-proof (f-equal (z-list (list1),
                                                                         z-list (list2)),
                                                                f-equal (z-list (cdr (list1)),
                                                                         z-list (cdr (list2))),
                                                                pf,
                                                                z-cdr-unique-proof (car (list1),
                                                                                    z-list (cdr (list1)),
                                                                                    car (list2),
                                                                                    z-list (cdr (list2))))))
         else pf endif
    else nil endif
```

Definition:

```
len-equal (x, y)
=   if listp (x) ∧ listp (y) then len-equal (cdr (x), cdr (y))
    else (x ≃ nil) ∧ (y ≃ nil) endif
```

Theorem: z-list-equal-proof-proves

```
 (subset (list (p (1, 2), fn (1, 2)), symbols)
  ∧   termp (list1, 1, symbols)
  ∧   termp (list2, 1, symbols)
  ∧   listp (list1)
  ∧   listp (list2)
  ∧   len-equal (list1, list2)
  ∧   proves (pf, f-equal (z-list (list1), z-list (list2)), given, defns, symbols))
 →    proves (z-list-equal-proof (list1, list2, pf),
              list-f-and (list-f-equal (list1, list2)),
```

```
        given,
        defns,
        symbols)
```

Event: Enable z-list-equal-proof; name this event ‘g0838’.

```
;;;;;;;PROVES-FORM-Z-IF-OK is similar to PROVES-Z-IF except that in the base case,
;;;;;;;it merely checks if PFTREE is a proof of EXP.
```

Definition:

```
proves-form-z-if-ok (pftree, exp, given, defns, symbols)
=  if z-ifp (exp)
   then if caar (pftree)
        then proves (cadar (pftree), z-test (exp), given, defns, symbols)
             ∧  proves-form-z-if-ok (cadr (pftree),
                                     z-left (exp),
                                     given,
                                     defns,
                                     symbols)
        else proves (caddar (pftree),
                     f-not (z-test (exp)),
                     given,
                     defns,
                     symbols)
             ∧  proves-form-z-if-ok (caddr (pftree),
                                     z-right (exp),
                                     given,
                                     defns,
                                     symbols) endif
   else proves (pftree, exp, given, defns, symbols) endif
```

Definition:

```
form-z-if-ok-proof (pftree, exp)
=  if z-ifp (exp)
   then if caar (pftree)
        then f-if-intro-proof1 (z-test (exp),
                                make-form-z-if (z-left (exp)),
                                make-form-z-if (z-right (exp)),
                                cadar (pftree),
                                form-z-if-ok-proof (cadr (pftree),
                                                    z-left (exp)))
        else f-if-intro-proof2 (z-test (exp),
                                make-form-z-if (z-left (exp)),
                                make-form-z-if (z-right (exp)),
```

```
                              caddar (pftree),
                              form-z-if-ok-proof (caddr (pftree),
                                                  z-right (exp))) endif
    else pftree endif
```

```
;;;;;;;From a proof-tree containing proofs of each of the test nodes of an form-if-tree
;;;;;;;and the leaf node that they lead upto, we can construct a proof of the entire
;;;;;;;if-expression represented by the form-if-tree.
```

Theorem: form-z-if-ok-proof-proves

```
 (form-z-if (exp, symbols)
  ∧   proves-form-z-if-ok (pftree, exp, given, defns, symbols)
  ∧   (concl = make-form-z-if (exp)))
 →    proves (form-z-if-ok-proof (pftree, exp), concl, given, defns, symbols)
```

Event: Enable form-z-if-ok-proof; name this event ‘g0839’.

Definition:

```
equal-z-list-proof (x, y, pflist)
=   if listp (x) ∧ listp (y)
    then if listp (cdr (x)) ∧ listp (cdr (y))
         then list-detach-proof (list (f-equal (car (x), car (y)),
                                       f-equal (z-list (cdr (x)),
                                                z-list (cdr (y)))),
                                 f-equal (z-list (x), z-list (y)),
                                 list (car (pflist),
                                       equal-z-list-proof (cdr (x),
                                                           cdr (y),
                                                           cdr (pflist))),
                                 opair-equal-proof (car (x),
                                                    z-list (cdr (x)),
                                                    car (y),
                                                    z-list (cdr (y))))
         else car (pflist) endif
    else nil endif
```

Theorem: equal-z-list-proof-proves

```
 ((fn (1, 2) ∈ symbols)
  ∧   termp (x, 1, symbols)
  ∧   termp (y, 1, symbols)
  ∧   listp (x)
  ∧   listp (y)
  ∧   len-equal (x, y)
  ∧   proves-list (pflist, list-f-equal (x, y), given, defns, symbols))
```

$\rightarrow$ proves (equal-z-list-proof (x, y, *pflist*),
f-equal (z-list (x), z-list (y)),
given,
defns,
symbols)

Event: Enable equal-z-list-proof; name this event 'g0842'.

Theorem: equal-z-list-proof-proves1
((fn (1, 2) $\in$ *symbols*)
$\land$ termp (x, 1, *symbols*)
$\land$ termp (y, 1, *symbols*)
$\land$ listp (x)
$\land$ listp (y)
$\land$ len-equal (x, y)
$\land$ proves-list (*pflist*, list-f-equal (x, y), *given*, *defns*, *symbols*)
$\land$ (*concl* = f-equal (z-list (x), z-list (y))))
$\rightarrow$ proves (equal-z-list-proof (x, y, *pflist*), *concl*, *given*, *defns*, *symbols*)

Event: Enable equal-z-list-proof-proves; name this event 'g0843'.

Theorem: nlistp-ncode
($x \simeq$ **nil**) $\rightarrow$ (ncode (x) = numeral (x))

Definition:
flip-not-equals-proof (x, y, *pf*)
= detach-proof (f-not (f-equal (x, y)),
f-not (f-equal (y, x)),
pf,
contra-pos-proof1 (f-equal (y, x),
f-equal (x, y),
symmetry-proof1 (y, x)))

Theorem: term-list-var-list
var-list (*vars*, length (*vars*)) $\rightarrow$ term-list (*vars*, *symbols*)

Definition:
form-equal-case-proof (*vars*, *terms*, *exp*, *pf*)
= tautconseq-proof (list (parallel-subst (*exp*, *vars*, *terms*, 0),
f-implies (list-f-and (list-f-equal (*vars*, *terms*)),
f-iff (*exp*,
parallel-subst (*exp*,
vars,
terms,

```
                                                 0)))),
                         f-implies (list-f-and (list-f-equal (vars, terms)), exp),
                         list (pf,
                               form-equal-proof (exp,
                                                 parallel-subst (exp,
                                                                 vars,
                                                                 terms,
                                                                 0),
                                                 list-f-equal (vars, terms))))
```

THEOREM: term-list-termp
term-list (*terms*, *symbols*) $\rightarrow$ termp (*terms*, 1, *symbols*)

```
;;;;;;;If |- "[a1/x1, ..., an/xn]A", then |- "(((x1=a1) ^ ... ^ (xn=an)) -> A)".
```

THEOREM: form-equal-case-proof-proves
```
 (formula (exp, symbols)
  ∧   proves (pf, parallel-subst (exp, vars, terms, 0), given, defns, symbols)
  ∧   var-list (vars, length (vars))
  ∧   term-list (terms, symbols)
  ∧   listp (list-f-equal (vars, terms))
  ∧   (collect-free (terms, 1) = nil)
  ∧   (concl = f-implies (list-f-and (list-f-equal (vars, terms)), exp)))
 →   proves (form-equal-case-proof (vars, terms, exp, pf),
             concl,
             given,
             defns,
             symbols)
```

EVENT: Enable form-equal-case-proof; name this event 'g0931'.

DEFINITION:
```
z-list-equal-imp-proof (x, y)
=   if listp (x) ∧ listp (y)
    then if listp (cdr (x)) ∧ listp (cdr (y))
         then tautconseq-proof (list (f-implies (f-equal (z-list (x), z-list (y)),
                                                 f-equal (car (x), car (y))),
                                      f-implies (f-equal (z-list (x), z-list (y)),
                                                 list-f-and (list-f-equal (cdr (x),
                                                                           cdr (y))))),
                                f-implies (f-equal (z-list (x), z-list (y)),
                                           list-f-and (list-f-equal (x, y))),
                                list (z-car-unique-proof (car (x),
                                                          z-list (cdr (x)),
```

car (y),
z-list (cdr (y))),
chain-proof (f-equal (z-list (x),
z-list (y)),
list-f-and (list-f-equal (cdr (x),
cdr (y))),
list (f-equal (z-list (cdr (x)),
z-list (cdr (y)))),
list (z-cdr-unique-proof (car (x),
z-list (cdr (x)),
car (y),
z-list (cdr (y))),
z-list-equal-imp-proof (cdr (x),
cdr (y))))))
else tautconseq-proof (**nil**,
f-implies (f-equal (car (x), car (y)),
f-equal (car (x), car (y))),
nil) **endif**
else nil endif

THEOREM: termp-car-cdr
(listp (x) ∧ termp (x, 1, *symbols*))
→ (termp (car (x), 0, *symbols*) ∧ termp (cdr (x), 1, *symbols*))

THEOREM: z-list-equal-imp-proof-proves
(subset (list (p (1, 2), fn (1, 2)), *symbols*)
∧ termp (*list1*, 1, *symbols*)
∧ termp (*list2*, 1, *symbols*)
∧ listp (*list1*)
∧ listp (*list2*)
∧ len-equal (*list1*, *list2*))
→ proves (z-list-equal-imp-proof (*list1*, *list2*),
f-implies (f-equal (z-list (*list1*), z-list (*list2*)),
list-f-and (list-f-equal (*list1*, *list2*))),
given,
defns,
symbols)

EVENT: Enable z-list-equal-imp-proof; name this event ‘g0845’.

THEOREM: z-list-equal-imp-proof-proves1
(subset (list (p (1, 2), fn (1, 2)), *symbols*)
∧ termp (*list1*, 1, *symbols*)
∧ termp (*list2*, 1, *symbols*)

$\wedge$ listp (*list1*)
$\wedge$ listp (*list2*)
$\wedge$ len-equal (*list1*, *list2*)
$\wedge$ (*concl* = f-implies (f-equal (z-list (*list1*), z-list (*list2*)),
list-f-and (list-f-equal (*list1*, *list2*)))))
$\rightarrow$ proves (z-list-equal-imp-proof (*list1*, *list2*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-list-equal-imp-proof-proves; name this event 'g0846'.

DEFINITION:
equal-isin-proof (x, y, z, *pf1*, *pf2*)
= list-detach-proof (list (f-equal (x, y), f-equal (z, z), isin (x, z)),
isin (y, z),
list (*pf1*, ident-axiom-proof (z), *pf2*),
equal-axiom2-proof (p (1, 2), list (x, z), list (y, z)))

```
;;;;;;;If |- "(x = y)", and |- "(x in z)", then |- "(y in z)".
```

THEOREM: equal-isin-proof-proves
(proves (*pf1*, f-equal (x, y), *given*, *defns*, *symbols*)
$\wedge$ proves (*pf2*, isin (x, z), *given*, *defns*, *symbols*))
$\rightarrow$ proves (equal-isin-proof (x, y, z, *pf1*, *pf2*), isin (y, z), *given*, *defns*, *symbols*)

```
;;;;;;;As it turns out, in order to represent the Lisp interpreter in Z2, we only need
;;;;;;;represent two recursive functions, the function GET below, and the Lisp interpreter
;;;;;;;EV itself.  The Lisp interpreter is a Lisp function which evaluates quoted
;;;;;;;Lisp objects which represent pure-Lisp expressions.  These e-expressions are
;;;;;;;either e-variables (represented by numbers) or e-function symbols (also represented
;;;;;;;by numbers) applied to a list of e-expressions.  The interpreter takes as
;;;;;;;its arguments: the e-expression to be evaluated, the bindings of values to the
;;;;;;;e-variables, and the bindings of definitions to the defined e-function symbols.
;;;;;;;Since e-variables are numbers, their bindings are stored in a list so that the
;;;;;;;binding of the variable N is the N+1th element in the list.  GET does exactly
;;;;;;;this by returning the X+1th element in the list Y.
```

DEFINITION:
get (x, y)
= **if** $x \simeq 0$ **then** car (y)
else get ($x - 1$, cdr (y)) **endif**

```
;;;;;;;We said earlier that the partial graph corresponding to a computation is first
;;;;;;;constructed as a list of value/argument pairs which is then translated into the
;;;;;;;corresponding set.  GET-GRAPH constructs this list of value/argument pairs
;;;;;;;corresponding to the computation of (GET X Y).
```

Definition:
get-graph $(x,\ y)$
$=$ **if** $x \simeq 0$ **then** list (cons (get $(x,\ y)$, cons $(x,\ y)$))
else cons (cons (get $(x,\ y)$, cons $(x,\ y)$), get-graph $(x - 1$, cdr (y))) **endif**

Theorem: member-get-graph
cons (get $(x,\ y)$, cons $(x,\ y)$) $\in$ get-graph $(x,\ y)$

Theorem: in-graph-cdr-get-graph
$(x \not\simeq 0) \rightarrow$ (cons (get $(x,\ y)$, cons $(x - 1$, cdr (y))) $\in$ get-graph $(x,\ y)$)

Definition:
a-get-defn-part-ok-proof1 $(x,\ y)$

```
=    form-z-if-ok-proof(list (list (x ≃ 0,
                                    z-zerop-ncode-proof (x),
                                    trans-not-equal-proof (ncode (1),
                                                           z-zerop (ncode (x)),
                                                           ncode (0),
                                                           ncode-nequal-proof (1,
                                                                               0),
                                                           z-zerop-ncode-proof (x))),
                              flip-equals-proof (z-car (ncode (y)),
                                                 ncode (get (x, y)),
                                                 z-car-ncode-proof (y)),
                              equal-isin-proof (z-list (list (ncode (get (x, y)),
                                                              ncode (x − 1),
                                                              ncode (cdr (y)))),
                                                z-list (list (ncode (get (x, y)),
                                                              z-sub1 (ncode (x)),
                                                              z-cdr (ncode (y)))),
                                                graph-trans (get-graph (x, y)),
                                                equal-z-list-proof (list (ncode (get (x,
                                                                                      y)),
                                                                          ncode (x − 1),
                                                                          ncode (cdr (y))),
                                                                    list (ncode (get (x,
                                                                                      y)),
                                                                          z-sub1 (ncode (x)),
                                                                          z-cdr (ncode (y))),
                                                                    list (ident-axiom-proof (ncode (get (x,
                                                                                                         y))),
                                                                          flip-equals-proof (z-sub1 (ncode (x)),
                                                                                             ncode (x − 1),
                                                                                             z-sub1-ncode-proof (x)),
                                                                          flip-equals-proof (z-cdr (ncode (y)),
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\text{cdr}\,(y)),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{z-cdr-ncode-proof}\,(y)))),\\
\qquad\qquad\qquad\qquad \text{in-graph-proof}\,(\text{cons}\,(\text{get}\,(x,\ y),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{cons}\,(x - 1,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{cdr}\,(y))),\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{get-graph}\,(x,\ y)))),\\
\qquad \text{z-if}\,(\text{f-equal}\,(\text{z-zerop}\,(\text{ncode}\,(x)),\ \text{ncode}\,(1)),\\
\qquad\quad \text{f-equal}\,(\text{ncode}\,(\text{get}\,(x,\ y)),\ \text{z-car}\,(\text{ncode}\,(y))),\\
\qquad\quad \text{isin}\,(\text{z-opair}\,(\text{ncode}\,(\text{get}\,(x,\ y)),\\
\qquad\qquad\qquad\qquad \text{z-opair}\,(\text{z-sub1}\,(\text{ncode}\,(x)),\\
\qquad\qquad\qquad\qquad\qquad \text{z-cdr}\,(\text{ncode}\,(y)))),\\
\qquad\qquad \text{graph-trans}\,(\text{get-graph}\,(x,\ y)))))
\end{array}
$$

```
;;;;;;;At this point we present the Z2 formula which represents the Lisp function GET.
;;;;;;;The formula representing the assertion (GET X Y) = VAL asserts the existence
;;;;;;;of a partial graph "trace" to which the tuple <<VAL>, <X>, <Y>> belongs, such
;;;;;;;that all of its members satisfy the Z2 analogue of the definition of GET.
;;;;;;;More formally,
;;;;;;;     "(forsome trace ((<val, x, y> in trace)
;;;;;;;                      ^
;;;;;;;                      (forall v1, x1, y1
;;;;;;;                        ((<v1, x1, y1> in trace)
;;;;;;;                         ->
;;;;;;;                         (if ((Z-zerop x1) = <1>)
;;;;;;;                          then (v1 = (Z-car y1))
;;;;;;;                          else (<v1, (Z-sub1 x1), (Z-cdr y1)> in trace))))))".
;;;;;;;
;;;;;;;Notice how in the second part where we specify that each 3-tuple "<v1,x1,y1>"
;;;;;;;in "trace" satisfies the Z2 analogue of the definition of GET, the analogue of
;;;;;;;the recursive call (GET (SUB1 X)(CDR Y)) is the membership check
;;;;;;;"(<v1, (Z-sub1 x1), (Z-cdr y1)> in trace)".  A-GET-DEFN-PART below constructs
;;;;;;;the definition (second) part of the above formula.
```

DEFINITION:
a-get-defn-part $(x,\ y,\ val,\ trace)$

$$
\begin{array}{ll}
= & \text{f-if}\,(\text{f-equal}\,(\text{z-zerop}\,(x),\ \text{ncode}\,(1)),\\
& \quad \text{f-equal}\,(val,\ \text{z-car}\,(y)),\\
& \quad \text{isin}\,(\text{z-list}\,(\text{list}\,(val,\ \text{z-sub1}\,(x),\ \text{z-cdr}\,(y))),\ trace))
\end{array}
$$

```
;;;;;;;Next we wish to show that the partial graph given by (GET-GRAPH X Y)
;;;;;;;satisfies the formula above.  By this we show that the representing formula
;;;;;;;is `OK'.  This is done in many steps.  One part
;;;;;;;of this is to show that each element of this partial graph satisfies the
;;;;;;;A-GET-DEFN-PART.  This will be done by induction.  The first part is
;;;;;;;to show that "<<(GET X Y)>, <X>, <Y>>" satisfies the definition part
```

```
;;;;;;;with (GRAPH-TRANS (GET-GRAPH X Y)) for "trace".  This is carried out below.
```

THEOREM: a-get-defn-part-ok-proof1-proves

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{subset}\,(\text{list}\,(\text{fn}\,(\mathtt{10},\,\mathtt{1}),}\\
& \quad\text{fn}\,(\mathtt{9},\,\mathtt{1}),\\
& \quad\text{fn}\,(\mathtt{5},\,\mathtt{1}),\\
& \quad\text{fn}\,(\mathtt{4},\,\mathtt{1}),\\
& \quad\text{fn}\,(\mathtt{3},\,\mathtt{1}),\\
& \quad\text{fn}\,(\mathtt{2},\,\mathtt{2}),\\
& \quad\text{fn}\,(\mathtt{1},\,\mathtt{2}),\\
& \quad\text{p}\,(\mathtt{1},\,\mathtt{2}),\\
& \quad\text{p}\,(\mathtt{2},\,\mathtt{1}),\\
& \quad\text{fn}\,(\mathtt{0},\,\mathtt{0})),\\
& \mathit{symbols})\\
\wedge & \text{z-succ-hyps}\,(\mathit{defns},\,\mathit{symbols})\\
\wedge & \text{z-sub1-hyps}\,(\mathit{defns},\,\mathit{symbols})\\
\wedge & \text{z-car-hyps}\,(\mathit{defns},\,\mathit{symbols})\\
\wedge & \text{z-cdr-hyps}\,(\mathit{defns},\,\mathit{symbols})\\
\wedge & \text{z-zerop-hyps}\,(\mathit{defns},\,\mathit{symbols})\\
\wedge & \text{z-int-hyps}\,(\mathit{defns},\,\mathit{symbols})\\
\wedge & (\mathit{concl} = \text{a-get-defn-part}\,(\text{ncode}\,(x),\\
& \qquad\qquad \text{ncode}\,(y),\\
& \qquad\qquad \text{ncode}\,(\text{get}\,(x,\,y)),\\
& \qquad\qquad \text{graph-trans}\,(\text{get-graph}\,(x,\,y)))))\\
\rightarrow & \text{proves}\,(\text{a-get-defn-part-ok-proof1}\,(x,\,y),\,\mathit{concl},\,\mathit{given},\,\mathit{defns},\,\mathit{symbols})
\end{array}
$$

EVENT: Enable a-get-defn-part-ok-proof1; name this event ‘g0847’.

DEFINITION:
in-graph-cdr-proof$(x,\ y,\ \mathit{graph})$

$$
\begin{array}{ll}
= & \text{tautconseq-proof}\,(\text{list}\,(\text{f-implies}\,(\text{f-or}\,(\text{f-equal}\,(x,\,\text{ncode}\,(y)),\\
& \qquad\qquad \text{isin}\,(x,\,\text{graph-trans}\,(\mathit{graph}))),\\
& \qquad \text{isin}\,(x,\,\text{graph-trans}\,(\text{cons}\,(y,\,\mathit{graph})))),\\
& \text{f-implies}\,(\text{isin}\,(x,\,\text{graph-trans}\,(\mathit{graph})),\\
& \qquad \text{isin}\,(x,\,\text{graph-trans}\,(\text{cons}\,(y,\,\mathit{graph})))),\\
& \text{list}\,(\text{iff-split-proof2}\,(\text{isin}\,(x,\\
& \qquad\qquad \text{graph-trans}\,(\text{cons}\,(y,\,\mathit{graph}))),\\
& \qquad \text{f-or}\,(\text{f-equal}\,(x,\,\text{ncode}\,(y)),\\
& \qquad\qquad \text{isin}\,(x,\,\text{graph-trans}\,(\mathit{graph}))),\\
& \qquad \text{in-graph-trans-proof}\,(x,\,y,\,\mathit{graph}))))
\end{array}
$$

```
;;;;;;; |- "((x in [GRAPH]) -> (x in [(CONS Y GRAPH)]))".
```

THEOREM: in-graph-cdr-proof-proves

(subset (list (p (1, 2), fn (3, 1), fn (2, 2), fn (1, 2), fn (0, 0)), *symbols*)
 ∧ termp (x, 0, *symbols*))
→ proves (in-graph-cdr-proof (x, y, *graph*),
  f-implies (isin (x, graph-trans (*graph*)),
   isin (x, graph-trans (cons (y, *graph*)))),
  given,
  defns,
  symbols)

Event: Enable in-graph-cdr-proof; name this event ‘g0850’.

Definition:
a-get-defn-part-step-proof (y, u, v, *v1*, *pf*)
= tautconseq-proof (list (f-implies (v, *v1*)),
  f-implies (f-if (y, u, v), f-if (y, u, *v1*)),
  list (*pf*))

```
;;;;;;;If |- "(C -> C1)", then |- "((if A then B else C) -> (if A then B else C1))".
```

Theorem: a-get-defn-part-step-proof-proves
(formula (f-if (y, u, v), *symbols*)
 ∧ formula (*v1*, *symbols*)
 ∧ proves (*pf*, f-implies (v, *v1*), *given*, *defns*, *symbols*))
→ proves (a-get-defn-part-step-proof (y, u, v, *v1*, *pf*),
  f-implies (f-if (y, u, v), f-if (y, u, *v1*)),
  given,
  defns,
  symbols)

Event: Enable a-get-defn-part-step-proof; name this event ‘g0851’.

Definition:
inst-formals-proof (*formals*, *actuals*, *exp*, *pf*)
= chain-proof (f-equal (z-list (*formals*), z-list (*actuals*)),
  exp,
  list (list-f-and (list-f-equal (*formals*, *actuals*))),
  list (z-list-equal-imp-proof (*formals*, *actuals*),
   form-equal-case-proof (*formals*, *actuals*, *exp*, *pf*)))

Definition:
a-get-defn-ok-proof (*x1*, *y1*, *val1*, x, y)
= **if** $x \simeq 0$
 then graph-trans-merge-proof (cons (get (x, y), cons (x, y)),
  nil,

```
                z-list (list (val1, x1, y1)),
                a-get-defn-part (x1,
                                 y1,
                                 val1,
                                 graph-trans (get-graph (x,
                                                         y))),
                inst-formals-proof (list (val1, x1, y1),
                                    list (ncode (get (x, y)),
                                          ncode (x),
                                          ncode (y)),
                                    a-get-defn-part (x1,
                                                     y1,
                                                     val1,
                                                     graph-trans (get-graph (x,
                                                                             y))),
                                    a-get-defn-part-ok-proof1 (x,
                                                               y)),
                tautconseq-proof (list (f-not (isin (z-list (list (val1,
                                                                   x1,
                                                                   y1)),
                                                     PHI))),
                                  f-implies (isin (z-list (list (val1,
                                                                 x1,
                                                                 y1)),
                                                   PHI),
                                             a-get-defn-part (x1,
                                                              y1,
                                                              val1,
                                                              graph-trans (get-graph (x,
                                                                                      y)))),
                                  list (null-axiom-proof (z-list (list (val1,
                                                                        x1,
                                                                        y1))))))
else graph-trans-merge-proof (cons (get (x, y), cons (x, y)),
                              get-graph (x - 1, cdr (y)),
                              z-list (list (val1, x1, y1)),
                              a-get-defn-part (x1,
                                               y1,
                                               val1,
                                               graph-trans (get-graph (x,
                                                                       y))),
                              inst-formals-proof (list (val1, x1, y1),
                                                  list (ncode (get (x,
                                                                    y)),
```

```
                    ncode (x),
                    ncode (y)),
                a-get-defn-part (x1,
                                 y1,
                                 val1,
                                 graph-trans (get-graph (x,
                                                         y))),
                a-get-defn-part-ok-proof1 (x,
                                           y)),
chain-proof (isin (z-list (list (val1,
                                 x1,
                                 y1)),
                   graph-trans (get-graph (x - 1,
                                           cdr (y)))),
             a-get-defn-part (x1,
                              y1,
                              val1,
                              graph-trans (get-graph (x,
                                                      y))),
             list (a-get-defn-part (x1,
                                    y1,
                                    val1,
                                    graph-trans (get-graph (x - 1,
                                                            cdr (y))))),
             list (a-get-defn-ok-proof (x1,
                                        y1,
                                        val1,
                                        x - 1,
                                        cdr (y)),
                   a-get-defn-part-step-proof (f-equal (z-zerop (x1),
                                                        ncode (1)),
                                               f-equal (val1,
                                                        z-car (y1)),
                                               isin (z-list (list (val1,
                                                                   z-sub1 (x1),
                                                                   z-cdr (y1))),
                                                     graph-trans (get-graph (x - 1,
                                                                             cdr (y)))),
                                               isin (z-list (list (val1,
                                                                   z-sub1 (x1),
                                                                   z-cdr (y1))),
                                                     graph-trans (get-graph (x,
                                                                             y))),
                                               in-graph-cdr-proof (z-list (list (val1,
```

```
                                                                z-sub1 (x1),
                                                                z-cdr (y1))),
                                                         cons (get (x,
                                                                    y),
                                                              cons (x,
                                                                    y)),
                                                         get-graph (x - 1,
                                                                    cdr (y)))))) endif
```

```
;;;;;;;In the lemma below, we carry out the induction to prove that every element of
;;;;;;;the partial graph satisfies the analogue of the definition of GET, i.e.,
;;;;;;; |- "((<v1,x1,y1> in [(GET-GRAPH X Y)]) -> (A-GET-DEFN-PART x1 y1 v1 [(GET-GRAPH X Y)]))".
```

THEOREM: a-get-defn-ok-proof-proves

```
(subset (list (fn (10, 1),
               fn (9, 1),
               fn (5, 1),
               fn (4, 1),
               fn (3, 1),
               fn (2, 2),
               fn (1, 2),
               p (1, 2),
               p (2, 1),
               fn (0, 0)),
         symbols)
 ∧   z-succ-hyps (defns, symbols)
 ∧   z-sub1-hyps (defns, symbols)
 ∧   z-zerop-hyps (defns, symbols)
 ∧   z-car-hyps (defns, symbols)
 ∧   z-cdr-hyps (defns, symbols)
 ∧   z-int-hyps (defns, symbols)
 ∧   var-set (list (val1, x1, y1), 3))
→   proves (a-get-defn-ok-proof (x1, y1, val1, x, y),
            f-implies (isin (z-list (list (val1, x1, y1)),
                             graph-trans (get-graph (x, y))),
                       a-get-defn-part (x1,
                                        y1,
                                        val1,
                                        graph-trans (get-graph (x, y)))),
            given,
            defns,
            symbols)
```

EVENT: Enable a-get-defn-ok-proof; name this event ‘g0852’.

DEFINITION:
list-gen-proof (*vars*, *exp*, *pf*)
= **if** listp (*vars*)
then generalise-proof (car (*vars*),
list-all-quantify (cdr (*vars*), *exp*),
list-gen-proof (cdr (*vars*), *exp*, *pf*))
else *pf* **endif**

THEOREM: list-gen-proof-proves
(var-list (*vars*, length (*vars*)) ∧ proves (*pf*, *exp*, *given*, *defns*, *symbols*))
→ proves (list-gen-proof (*vars*, *exp*, *pf*),
list-all-quantify (*vars*, *exp*),
given,
defns,
symbols)

EVENT: Enable list-gen-proof; name this event 'g0853'.

EVENT: Enable a-get-defn-part; name this event 'g0854'.

```
;;;;;;;A-GET combines the first and second parts of the specification of "trace".
```

DEFINITION:
a-get (*x*, *y*, *val*, *x1*, *y1*, *val1*, *trace*)
= f-and (isin (z-opair (*val*, z-opair (*x*, *y*)), *trace*),
list-all-quantify (list (*x1*, *y1*, *val1*),
f-implies (isin (z-list (list (*val1*, *x1*, *y1*)), *trace*),
a-get-defn-part (*x1*, *y1*, *val1*, *trace*))))

DEFINITION:
a-get-ok-proof1 (*x*, *y*, *x1*, *y1*, *val1*)
= and-join-proof (isin (ncode (cons (get (*x*, *y*), cons (*x*, *y*))),
graph-trans (get-graph (*x*, *y*))),
list-all-quantify (list (*x1*, *y1*, *val1*),
f-implies (isin (z-list (list (*val1*,
x1,
y1)),
graph-trans (get-graph (*x*,
y))),
a-get-defn-part (*x1*,
y1,
val1,
graph-trans (get-graph (*x*,
y))))),

```
                    in-graph-proof (cons (get (x, y), cons (x, y)), get-graph (x, y)),
                    list-gen-proof (list (x1, y1, val1),
                                    f-implies (isin (z-list (list (val1, x1, y1)),
                                                     graph-trans (get-graph (x,
                                                                             y))),
                                               a-get-defn-part (x1,
                                                                y1,
                                                                val1,
                                                                graph-trans (get-graph (x,
                                                                                        y)))),
                                    a-get-defn-ok-proof (x1, y1, val1, x, y)))
```

```
;;;;;;;Finally we show that
;;;;;;;|- "[<X>/x, <Y>/y, <(GET X Y)>/val, [(GET-GRAPH X Y)]/trace]A-get".
;;;;;;;This completes the first part of the representability proof for GET.
;;;;;;;The second part involves showing that any "trace" satisfying the
;;;;;;;specification A-GET must encode only evaluations of GET.  By this we
;;;;;;;show that the representing formula is ‘GOOD’.
```

THEOREM: a-get-ok-proof-proves

```
 (subset (list (fn (10, 1),
                fn (9, 1),
                fn (5, 1),
                fn (4, 1),
                fn (3, 1),
                fn (2, 2),
                fn (1, 2),
                p (1, 2),
                p (2, 1),
                fn (0, 0)),
          symbols)
  ∧  z-succ-hyps (defns, symbols)
  ∧  z-zerop-hyps (defns, symbols)
  ∧  z-sub1-hyps (defns, symbols)
  ∧  z-car-hyps (defns, symbols)
  ∧  z-cdr-hyps (defns, symbols)
  ∧  z-int-hyps (defns, symbols)
  ∧  var-set (list (val1, x1, y1), 3)
  ∧  (x-bar = ncode (x))
  ∧  (y-bar = ncode (y))
  ∧  (val-bar = ncode (get (x, y)))
  ∧  (trace = graph-trans (get-graph (x, y))))
 →  proves (a-get-ok-proof1 (x, y, x1, y1, val1),
            a-get (x-bar, y-bar, val-bar, x1, y1, val1, trace),
```

given,
defns,
symbols)

EVENT: Enable a-get-ok-proof1; name this event 'g0856'.

EVENT: Disable a-get-defn-part; name this event 'g0857'.

DEFINITION:
equal-isin-imp-proof(x, y, z, pf)
= list-detach-proof(list (f-equal (x, y), f-equal (z, z)),
f-implies (isin (x, z), isin (y, z)),
list (pf, ident-axiom-proof (z)),
equal-axiom2-proof(p (1, 2), list (x, z), list (y, z)))

THEOREM: form-z-if-eval-proof-proves-again
(form-z-if (*exp*, *symbols*)
∧ termp (list (*lhs*, *val*), 1, *symbols*)
∧ proves-form-z-if-eval (*pf-tree*, *exp*, *lhs*, *val*, *given*, *defns*, *symbols*)
∧ (*form* = make-form-z-if (*exp*)))
→ proves (form-z-if-eval-proof (*pf-tree*, *exp*, *lhs*, *val*),
f-implies (*form*, f-equal (*lhs*, *val*)),
given,
defns,
symbols)

EVENT: Enable form-z-if-eval-proof-proves; name this event 'g0858'.

```
;;;;;;;PROVES-REC-FORM-Z-IF-EVAL is similar to PROVES-Z-IF, except that
;;;;;;;in the base case it checks if PF-TREE is a proof of "(EXP -> CONCL)".
```

DEFINITION:
proves-rec-form-z-if-eval (*pf-tree*, *exp*, *concl*, *given*, *defns*, *symbols*)
= **if** z-ifp (*exp*)
then if caar (*pf-tree*)
then proves (cadar (*pf-tree*), z-test (*exp*), *given*, *defns*, *symbols*)
∧ proves-rec-form-z-if-eval (cadr (*pf-tree*),
z-left (*exp*),
concl,
given,
defns,
symbols)
else proves (caddar (*pf-tree*),

```
                    f-not (z-test (exp)),
                    given,
                    defns,
                    symbols)
              ∧  proves-rec-form-z-if-eval (caddr (pf-tree),
                                            z-right (exp),
                                            concl,
                                            given,
                                            defns,
                                            symbols) endif
   else proves (pf-tree, f-implies (exp, concl), given, defns, symbols) endif
```

DEFINITION:

```
rec-form-z-if-eval-proof (pf-tree, exp, concl)
=  if z-ifp (exp)
   then if caar (pf-tree)
        then chain-proof (make-form-z-if (exp),
                          concl,
                          list (make-form-z-if (z-left (exp))),
                          list (f-if-elim-proof1 (z-test (exp),
                                                  make-form-z-if (z-left (exp)),
                                                  make-form-z-if (z-right (exp)),
                                                  cadar (pf-tree)),
                                rec-form-z-if-eval-proof (cadr (pf-tree),
                                                          z-left (exp),
                                                          concl)))
        else chain-proof (make-form-z-if (exp),
                          concl,
                          list (make-form-z-if (z-right (exp))),
                          list (f-if-elim-proof2 (z-test (exp),
                                                  make-form-z-if (z-left (exp)),
                                                  make-form-z-if (z-right (exp)),
                                                  caddar (pf-tree)),
                                rec-form-z-if-eval-proof (caddr (pf-tree),
                                                          z-right (exp),
                                                          concl))) endif
   else pf-tree endif
```

```
;;;;;;;The lemma below proves that if each of the leaf nodes of an form-if-tree
;;;;;;;implies a certain conclusion CONCL, then the if-expression represented by
;;;;;;;the if-tree implies CONCL.
```

THEOREM: rec-form-z-if-eval-proof-proves

```
 (form-z-if (exp, symbols)
  ∧  proves-rec-form-z-if-eval (pf-tree, exp, concl, given, defns, symbols))
```

$\rightarrow$ proves (rec-form-z-if-eval-proof (*pf-tree*, *exp*, *concl*),
f-implies (make-form-z-if (*exp*), *concl*),
given,
defns,
symbols)

EVENT: Enable rec-form-z-if-eval-proof; name this event 'g0743'.

THEOREM: rec-form-z-if-eval-proof-proves-again
(form-z-if (*exp*, *symbols*)
$\land$ proves-rec-form-z-if-eval (*pf-tree*, *exp*, *concl*, *given*, *defns*, *symbols*)
$\land$ (*form* = make-form-z-if (*exp*)))
$\rightarrow$ proves (rec-form-z-if-eval-proof (*pf-tree*, *exp*, *concl*),
f-implies (*form*, *concl*),
given,
defns,
symbols)

EVENT: Enable rec-form-z-if-eval-proof-proves; name this event 'g0745'.

THEOREM: termp-if
termp (**if** x **then** y
else z **endif**,
flg,
symbols)
= **if** x **then** termp (y, *flg*, *symbols*)
else termp (z, *flg*, *symbols*) **endif**

DEFINITION:
a-get-defn-part-good-proof1 (x, y, *val*, *trace*, *exp*, *pf*)
= rec-form-z-if-eval-proof (list (list ($x \simeq$ `0`,
z-zerop-ncode-proof (x),
trans-not-equal-proof (ncode (`1`),
z-zerop (ncode (x)),
ncode (`0`),
ncode-nequal-proof (`1`,
`0`),
z-zerop-ncode-proof (x))),
trans-implies-proof (f-equal (*val*,
z-car (ncode (y))),
f-equal (*val*,
ncode (car (y))),
f-implies (*exp*,
f-equal (*val*,

```
                  ncode (car (y)))),
list-detach-proof(list (f-equal (val,
                                 val),
                        f-equal (z-car (ncode (y)),
                                 ncode (car (y)))),
                  f-implies (f-equal (val,
                                      z-car (ncode (y))),
                             f-equal (val,
                                      ncode (car (y)))),
                  list (ident-axiom-proof (val),
                        z-car-ncode-proof (y)),
                  equal-axiom2-proof(p (0,
                                        2),
                                     list (val,
                                           z-car (ncode (y))),
                                     list (val,
                                           ncode (car (y))))),
tautconseq-proof (nil,
                  f-implies (f-equal (val,
                                      ncode (car (y))),
                             f-implies (exp,
                                        f-equal (val,
                                                 ncode (car (y))))),
                  nil)),
trans-implies-proof(isin (z-list (list (val,
                                        z-sub1 (ncode (x)),
                                        z-cdr (ncode (y)))),
                          trace),
                    isin (z-list (list (val,
                                        ncode (x − 1),
                                        ncode (cdr (y)))),
                          trace),
                    f-implies (exp,
                               f-equal (val,
                                        ncode (get (x,
                                                    y)))),
                    equal-isin-imp-proof(z-list (list (val,
                                                       z-sub1 (ncode (x)),
                                                       z-cdr (ncode (y)))),
                                         z-list (list (val,
                                                       ncode (x − 1),
                                                       ncode (cdr (y)))),
                                         trace,
                                         equal-z-list-proof(list (val,
```

```
                                                          z-sub1 (ncode (x)),
                                                          z-cdr (ncode (y))),
                                                      list (val,
                                                          ncode (x − 1),
                                                          ncode (cdr (y))),
                                                      list (ident-axiom-proof (val),
                                                          z-sub1-ncode-proof (x),
                                                          z-cdr-ncode-proof (y)))),
                         pf)),
              z-if (f-equal (z-zerop (ncode (x)), ncode (1)),
                   f-equal (val, z-car (ncode (y))),
                   isin (z-opair (val,
                                  z-opair (z-sub1 (ncode (x)),
                                           z-cdr (ncode (y)))),
                        trace)),
              f-implies (exp, f-equal (val, ncode (get (x, y)))))
```

```
;;;;;;;The second part involves showing that A-GET is 'GOOD', i.e., that
;;;;;;;any "trace" satisfying A-GET encodes a computation of GET.
;;;;;;;The proof is somewhat tricky but the essential idea is this:
;;;;;;;we want to prove "A-get(<X>, <Y>, val, trace) -> (val = <(GET X Y)>)".
;;;;;;;We start with the left-hand side of this implication, it implies that
;;;;;;; "(<<X>, <Y>, val> in trace)" and hence by the second part of "A-get"
;;;;;;;that "<<X>, <Y>, val>" satisfies the analogue of the definition of GET.
;;;;;;;If (ZEROP X), then "((Z-zerop <X>) = <1>)" and hence "(val = (Z-car <Y>))"
;;;;;;;which by the lemma about "Z-car" is "(val = <(CAR Y)>)" which is the
;;;;;;;desired conclusion.  In the case when (ZEROP X) is F, we have
;;;;;;;"(A-get(<X>, <Y>, val, trace) -> (<val, (Z-sub1 <X>), (Z-cdr <Y>)> in trace))"
;;;;;;;(call it [1]), the r.h.s. of which is "(<val, <(SUB1 X)>, <(CDR Y)>> in trace)"
;;;;;;;By the induction hypothesis,
;;;;;;;"(A-get(<(SUB1 X)>, <(CDR Y)>, val, trace) -> (val = <(GET (SUB1 X)(CDR Y))>))" [2].
;;;;;;;But "A-get(<X>, <Y>, val, trace)" and  "(<val, <(SUB1 X)>, <(CDR Y)>> in trace)" imply
;;;;;;;"A-get(<(SUB1 X)>, <(CDR Y)>, val, trace)".  By [1], [2], and the definition of GET,
;;;;;;;we have the desired conclusion.  The lemma below is one portion of the argument
;;;;;;;above where we prove
;;;;;;; |- "(A-get-defn-part(<X>, <Y>, val, trace) -> (EXP -> (val = <(GET X Y)>)))",
;;;;;;;provided when (ZEROP X) is F,
;;;;;;; |- "((<val, <(SUB1 X)>, <(CDR Y)>> in trace) -> (EXP -> (val = <(GET X Y)>)))".
;;;;;;;
```

THEOREM: a-get-defn-part-good-proof1-proves

```
 (subset (list (fn (10, 1),
                fn (9, 1),
                fn (5, 1),
```

```
              fn (4, 1),
              fn (3, 1),
              fn (2, 2),
              fn (1, 2),
              p (1, 2),
              p (2, 1),
              fn (0, 0)),
          symbols)
 ∧   z-succ-hyps (defns, symbols)
 ∧   z-zerop-hyps (defns, symbols)
 ∧   z-sub1-hyps (defns, symbols)
 ∧   z-car-hyps (defns, symbols)
 ∧   z-cdr-hyps (defns, symbols)
 ∧   z-int-hyps (defns, symbols)
 ∧   termp (val, 0, symbols)
 ∧   termp (trace, 0, symbols)
 ∧   formula (exp, symbols)
 ∧   (val-bar = ncode (get (x, y)))
 ∧   (form = a-get-defn-part (ncode (x), ncode (y), val, trace))
 ∧   if x ≃ 0 then t
     else proves (pf,
                  f-implies (isin (z-list (list (val,
                                                 ncode (x − 1),
                                                 ncode (cdr (y)))),
                                         trace),
                              f-implies (exp, f-equal (val, val-bar))),
                  given,
                  defns,
                  symbols) endif)
 →   proves (a-get-defn-part-good-proof1 (x, y, val, trace, exp, pf),
             f-implies (form, f-implies (exp, f-equal (val, val-bar))),
             given,
             defns,
             symbols)
```

Event: Enable a-get-defn-part-good-proof1; name this event 'g0615'.

Definition:

```
good-frame1-proof (a, b, c)
=   tautconseq-proof (nil,
                      f-iff (f-implies (f-and (a, b), c),
                             f-implies (b, f-implies (a, c))),
                      nil)
```

THEOREM: good-frame1-proof-proves
(formula (a, $symbols$) $\land$ formula (b, $symbols$) $\land$ formula (c, $symbols$))
$\rightarrow$ proves (good-frame1-proof (a, b, c),
f-iff (f-implies (f-and (a, b), c), f-implies (b, f-implies (a, c))),
$given$,
$defns$,
$symbols$)

EVENT: Enable good-frame1-proof; name this event ‘g0746’.

DEFINITION:
a-get-reduc-proof (x, y, val, $x1$, $y1$, $val1$, $trace$)
= f-iff-mp-proof2 (f-implies (a-get (x, y, val, $x1$, $y1$, $val1$, $trace$),
a-get-defn-part (x, y, val, $trace$)),
f-implies (list-all-quantify (list ($x1$, $y1$, $val1$),
f-implies (isin (z-list (list ($val1$,
$x1$,
$y1$)),
$trace$),
a-get-defn-part ($x1$,
$y1$,
$val1$,
$trace$))),
f-implies (isin (z-list (list (val, x, y)), $trace$),
a-get-defn-part (x, y, val, $trace$))),
good-frame1-proof (isin (z-list (list (val, x, y)), $trace$),
list-all-quantify (list ($x1$, $y1$, $val1$),
f-implies (isin (z-list (list ($val1$,
$x1$,
$y1$)),
$trace$),
a-get-defn-part ($x1$,
$y1$,
$val1$,
$trace$))),
a-get-defn-part (x, y, val, $trace$)),
subst-thm-b-proof (f-implies (isin (z-list (list ($val1$,
$x1$,
$y1$)),
$trace$),
a-get-defn-part ($x1$,
$y1$,
$val1$,
$trace$)),

list (*x1*, *y1*, *val1*),
list (*x*, *y*, *val*)))

```
;;;;;;; |- "(A-get(x, y, val, trace) -> A-get-defn-part(x, y, val, trace))".
```

THEOREM: a-get-reduc-proof-proves
(subset (list (fn (10, 1),
fn (9, 1),
fn (5, 1),
fn (4, 1),
fn (3, 1),
fn (2, 2),
fn (1, 2),
p (1, 2),
p (2, 1),
fn (0, 0)),
symbols)
∧ termp (*x*, 0, *symbols*)
∧ termp (*y*, 0, *symbols*)
∧ termp (*val*, 0, *symbols*)
∧ var-set (list (*x1*, *y1*, *val1*, *trace*), 4)
∧ (*concl* = f-implies (a-get (*x*, *y*, *val*, *x1*, *y1*, *val1*, *trace*),
a-get-defn-part (*x*, *y*, *val*, *trace*))))
→ proves (a-get-reduc-proof (*x*, *y*, *val*, *x1*, *y1*, *val1*, *trace*),
concl,
given,
defns,
symbols)

EVENT: Enable a-get-reduc-proof; name this event 'g0644'.

DEFINITION:
good-frame2-proof (*a*, *b*, *c*, *d*, *pf1*, *pf2*)
= tautconseq-proof (list (f-implies (f-and (*a*, *b*), *c*),
f-implies (*c*, f-implies (*b*, *d*))),
f-implies (f-and (*a*, *b*), *d*),
list (*pf1*, *pf2*))

THEOREM: good-frame2-proof-proves
(formula (*a*, *symbols*)
∧ formula (*b*, *symbols*)
∧ formula (*c*, *symbols*)
∧ formula (*d*, *symbols*)
∧ proves (*pf1*, f-implies (f-and (*a*, *b*), *c*), *given*, *defns*, *symbols*)

$\wedge$ proves (*pf2*, f-implies (*c*, f-implies (*b*, *d*)), *given*, *defns*, *symbols*))
$\rightarrow$ proves (good-frame2-proof (*a*, *b*, *c*, *d*, *pf1*, *pf2*),
f-implies (f-and (*a*, *b*), *d*),
given,
defns,
symbols)

EVENT: Enable good-frame2-proof; name this event 'g0747'.

DEFINITION:
good-frame3-proof (*a*, *b*, *c*, *pf*)
= tautconseq-proof (list (f-implies (f-and (*a*, *b*), *c*)),
f-implies (*a*, f-implies (*b*, *c*)),
list (*pf*))

THEOREM: good-frame3-proof-proves
proves (*pf*, f-implies (f-and (*a*, *b*), *c*), *given*, *defns*, *symbols*)
$\rightarrow$ proves (good-frame3-proof (*a*, *b*, *c*, *pf*),
f-implies (*a*, f-implies (*b*, *c*)),
given,
defns,
symbols)

EVENT: Enable good-frame3-proof; name this event 'g0748'.

DEFINITION:
a-get-good-proof (*x*, *y*, *val*, *x1*, *y1*, *val1*, *trace*)
= **if** $x \simeq 0$
then good-frame2-proof (isin (z-list (list (*val*, ncode (*x*), ncode (*y*))), *trace*),
list-all-quantify (list (*x1*, *y1*, *val1*),
f-implies (isin (z-list (list (*val1*,
x1,
y1)),
trace),
a-get-defn-part (*x1*,
y1,
val1,
trace))),
a-get-defn-part (ncode (*x*), ncode (*y*), *val*, *trace*),
f-equal (*val*, ncode (get (*x*, *y*))),
a-get-reduc-proof (ncode (*x*),
ncode (*y*),
val,
x1,

```
                                        y1,
                                        val1,
                                        trace),
                         a-get-defn-part-good-proof1 (x,
                                                      y,
                                                      val,
                                                      trace,
                                                      list-all-quantify (list (x1,
                                                                               y1,
                                                                               val1),
                                                                         f-implies (isin (z-list (list (val1,
                                                                                                        x1,
                                                                                                        y1)),
                                                                                          trace),
                                                                                    a-get-defn-part (x1,
                                                                                                     y1,
                                                                                                     val1,
                                                                                                     trace))),
                                                      nil))
else good-frame2-proof (isin (z-list (list (val, ncode (x), ncode (y))),
                              trace),
                        list-all-quantify (list (x1, y1, val1),
                                           f-implies (isin (z-list (list (val1,
                                                                          x1,
                                                                          y1)),
                                                            trace),
                                                      a-get-defn-part (x1,
                                                                       y1,
                                                                       val1,
                                                                       trace))),
                        a-get-defn-part (ncode (x),
                                         ncode (y),
                                         val,
                                         trace),
                        f-equal (val, ncode (get (x, y))),
                        a-get-reduc-proof (ncode (x),
                                           ncode (y),
                                           val,
                                           x1,
                                           y1,
                                           val1,
                                           trace),
                        a-get-defn-part-good-proof1 (x,
                                                     y,
```

```
val,
trace,
list-all-quantify (list (x1,
                         y1,
                         val1),
                   f-implies (isin (z-list (list (val1,
                                                  x1,
                                                  y1)),
                                   trace),
                              a-get-defn-part (x1,
                                               y1,
                                               val1,
                                               trace))),
good-frame3-proof (isin (z-list (list (val,
                                       ncode (x - 1),
                                       ncode (cdr (y)))),
                         trace),
                   list-all-quantify (list (x1,
                                            y1,
                                            val1),
                                      f-implies (isin (z-list (list (val1,
                                                                     x1,
                                                                     y1)),
                                                      trace),
                                                 a-get-defn-part (x1,
                                                                  y1,
                                                                  val1,
                                                                  trace))),
                   f-equal (val,
                            ncode (get (x,
                                        y))),
                   a-get-good-proof (x - 1,
                                     cdr (y),
                                     val,
                                     x1,
                                     y1,
                                     val1,
                                     trace)))) endif
```

```
;;;;;;;The desired ‘GOOD’ part:
;;;;;;; |- "(A-get(<X>, <Y>, val, trace) -> (val = <(GET X Y)>))".
```

THEOREM: a-get-good-proof-proves
 (subset (list (fn (10, 1),

fn (9, 1),
fn (5, 1),
fn (4, 1),
fn (3, 1),
fn (2, 2),
fn (1, 2),
p (1, 2),
p (2, 1),
fn (0, 0)),
symbols)
∧ z-succ-hyps (*defns*, *symbols*)
∧ z-zerop-hyps (*defns*, *symbols*)
∧ z-sub1-hyps (*defns*, *symbols*)
∧ z-car-hyps (*defns*, *symbols*)
∧ z-cdr-hyps (*defns*, *symbols*)
∧ z-int-hyps (*defns*, *symbols*)
∧ termp (*val*, 0, *symbols*)
∧ var-set (list (*x1*, *y1*, *val1*, *trace*), 4))
→ proves (a-get-good-proof (*x*, *y*, *val*, *x1*, *y1*, *val1*, *trace*),
f-implies (a-get (ncode (*x*), ncode (*y*), *val*, *x1*, *y1*, *val1*, *trace*),
f-equal (*val*, ncode (get (*x*, *y*)))),
given,
defns,
symbols)

EVENT: Enable a-get-good-proof; name this event 'g0749'.

EVENT: Enable ncode; name this event 'g0750'.

```
;;;;;;;SUBRP is a function used in the Lisp interpreter which checks if a given
;;;;;;;e-function symbol, i.e., a function symbol in an expression to be interpreted
;;;;;;;by the Lisp interpreter.  The events below introduce the Z2 analogue of this
;;;;;;;function.
```

DEFINITION: z-subrp (*fn*) = list (fn (16, 1), *fn*)

DEFINITION:
z-subrp-defn (*fn*, *y*)
= f-if (f-equal (*fn*, ncode (1)),
f-equal (*y*, ncode (1)),
f-if (f-equal (*fn*, ncode (2)),
f-equal (*y*, ncode (1)),
f-if (f-equal (*fn*, ncode (3)),
f-equal (*y*, ncode (1)),

```
f-if (f-equal (fn, ncode (4)),
     f-equal (y, ncode (1)),
     f-if (f-equal (fn, ncode (5)),
          f-equal (y, ncode (1)),
          f-if (f-equal (fn, ncode (6)),
               f-equal (y, ncode (1)),
               f-if (f-equal (fn, ncode (7)),
                    f-equal (y, ncode (1)),
                    f-if (f-equal (fn, ncode (8)),
                         f-equal (y, ncode (1)),
                         f-if (f-equal (fn, ncode (9)),
                              f-equal (y, ncode (1)),
                              f-equal (y, PHI)))))))))
```

DEFINITION:

```
z-subrp-expr (fn)
=  z-if (f-equal (fn, ncode (1)),
        ncode (1),
        z-if (f-equal (fn, ncode (2)),
             ncode (1),
             z-if (f-equal (fn, ncode (3)),
                  ncode (1),
                  z-if (f-equal (fn, ncode (4)),
                       ncode (1),
                       z-if (f-equal (fn, ncode (5)),
                            ncode (1),
                            z-if (f-equal (fn, ncode (6)),
                                 ncode (1),
                                 z-if (f-equal (fn, ncode (7)),
                                      ncode (1),
                                      z-if (f-equal (fn, ncode (8)),
                                           ncode (1),
                                           z-if (f-equal (fn, ncode (9)),
                                                ncode (1),
                                                PHI)))))))))
```

DEFINITION:

z-subrp-exis-proof (*fn*, *y*) = term-z-if-exis-proof (*y*, z-subrp-expr (*fn*))

DEFINITION:

z-subrp-unique-proof (*fn*, *y*, *y1*)
= term-z-if-unique-proof (*y*, *y1*, z-subrp-expr (*fn*))

DEFINITION:

z-subrp-defn-proof (*fn*)

$=$ instance-proof (z-subrp-defn (0, z-subrp (0)),
list (0),
list (*fn*),
fun-defn-proof (fn (16, 1),
list (0),
1,
2,
z-subrp-defn (0, 1),
z-subrp-exis-proof (0, 1),
z-subrp-unique-proof (0, 1, 2)))

DEFINITION:
z-subrp-hyps (*defns*, *symbols*)
$=$ ((assoc (fn (16, 1), *defns*) $=$ z-subrp-defn (0, z-subrp (0)))
$\wedge$ subset (list (fn (3, 1), fn (1, 2), fn (0, 0)),
rest-of (fn (16, 1), *symbols*)))

THEOREM: z-subrp-defn-proof-proves
(subset (list (fn (16, 1), fn (0, 0), fn (3, 1), fn (1, 2)), *symbols*)
$\wedge$ z-subrp-hyps (*defns*, *symbols*)
$\wedge$ termp (x, 0, *symbols*)
$\wedge$ (*concl* $=$ z-subrp-defn (x, z-subrp (x))))
$\rightarrow$ proves (z-subrp-defn-proof (x), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-subrp-defn-proof; name this event 'g0751'.

EVENT: Enable z-subrp-hyps; name this event 'g0752'.

DEFINITION:
subrp (*fn*)
$=$ ((*fn* $=$ 1)
$\vee$ (*fn* $=$ 2)
$\vee$ (*fn* $=$ 3)
$\vee$ (*fn* $=$ 4)
$\vee$ (*fn* $=$ 5)
$\vee$ (*fn* $=$ 6)
$\vee$ (*fn* $=$ 7)
$\vee$ (*fn* $=$ 8)
$\vee$ (*fn* $=$ 9))

```
(DEFN
 Z-SUBRP-NCODE-PROOF
 (FN)
```

```
(Z-IF-EVAL-PROOF1
 (LIST
  (LIST (EQUAL FN 1) (IDENT-AXIOM-PROOF (NCODE 1)) (NCODE-NEQUAL-PROOF FN 1))
  (IDENT-AXIOM-PROOF (NCODE 1))
  (LIST (LIST (EQUAL FN 2) (IDENT-AXIOM-PROOF (NCODE 2)) (NCODE-NEQUAL-PROOF FN 2))
        (IDENT-AXIOM-PROOF (NCODE 1))
        (LIST (LIST (EQUAL FN 3) (IDENT-AXIOM-PROOF (NCODE 3)) (NCODE-NEQUAL-PROOF FN 3))
              (IDENT-AXIOM-PROOF (NCODE 1))
              (LIST (LIST (EQUAL FN 4)
                          (IDENT-AXIOM-PROOF (NCODE 4))
                          (NCODE-NEQUAL-PROOF FN 4))
                    (IDENT-AXIOM-PROOF (NCODE 1))
                    (LIST (LIST (EQUAL FN 5)
                                (IDENT-AXIOM-PROOF (NCODE 5))
                                (NCODE-NEQUAL-PROOF FN 5))
                          (IDENT-AXIOM-PROOF (NCODE 1))
                          (LIST (LIST (EQUAL FN 6)
                                      (IDENT-AXIOM-PROOF (NCODE 6))
                                      (NCODE-NEQUAL-PROOF FN 6))
                                (IDENT-AXIOM-PROOF (NCODE 1))
                                (LIST (LIST (EQUAL FN 7)
                                            (IDENT-AXIOM-PROOF (NCODE 7))
                                            (NCODE-NEQUAL-PROOF FN 7))
                                      (IDENT-AXIOM-PROOF (NCODE 1))
                                      (LIST (LIST (EQUAL FN 8)
                                                  (IDENT-AXIOM-PROOF (NCODE 8))
                                                  (NCODE-NEQUAL-PROOF FN 8))
                                            (IDENT-AXIOM-PROOF (NCODE 1))
                                            (LIST (LIST (EQUAL FN 9)
                                                        (IDENT-AXIOM-PROOF (NCODE 9))
                                                        (NCODE-NEQUAL-PROOF FN 9))
                                                  (IDENT-AXIOM-PROOF (NCODE 1))
                                                  (IDENT-AXIOM-PROOF (PHI))))))))))
 (Z-SUBRP-EXPR (NCODE FN))
 (Z-SUBRP (NCODE FN))
 (NCODE (BOOL-FIX (SUBRP FN)))
 (Z-SUBRP-DEFN-PROOF (NCODE FN))))
```

THEOREM: z-subrp-ncode-proof-proves
(subset (list (fn (16, 1), fn (0, 0), fn (3, 1), fn (1, 2), fn (2, 2), p (2, 1), p (1, 2)),
symbols)
$\wedge$ z-subrp-hyps (*defns*, *symbols*)
$\wedge$ z-int-hyps (*defns*, *symbols*)
$\wedge$ z-succ-hyps (*defns*, *symbols*)

$\wedge$ (*concl* = f-equal (z-subrp (ncode (*fn*)), ncode (bool-fix (subrp (*fn*))))))
$\rightarrow$ proves (z-subrp-ncode-proof (*fn*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-subrp-ncode-proof; name this event 'g0753'.

EVENT: Enable subrp; name this event 'g0754'.

DEFINITION:
eg-proof (*exp*, *vars*, *terms*, *pf*)
= detach-proof (parallel-subst (*exp*, *vars*, *terms*, 0),
list-exis-quantify (*vars*, *exp*),
pf,
subst-thm-a-proof (*exp*, *vars*, *terms*))

```
;;;;;;; If |- "[a1/x1,..., an/xn]A", then |- "(forsome x1,...,xn A)".
```

THEOREM: eg-proof-proves
(formula (*exp*, *symbols*)
$\wedge$ termp (*terms*, 1, *symbols*)
$\wedge$ (length (*vars*) = length (*terms*))
$\wedge$ var-set (*vars*, length (*vars*))
$\wedge$ free-for-list (*exp*, *vars*, *terms*)
$\wedge$ proves (*pf*, parallel-subst (*exp*, *vars*, *terms*, 0), *given*, *defns*, *symbols*)
$\wedge$ (*concl* = list-exis-quantify (*vars*, *exp*)))
$\rightarrow$ proves (eg-proof (*exp*, *vars*, *terms*, *pf*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable eg-proof; name this event 'g0479'.

THEOREM: a-get-ok-proof-proves-again
(subset (list (fn (10, 1),
fn (9, 1),
fn (5, 1),
fn (4, 1),
fn (3, 1),
fn (2, 2),
fn (1, 2),
p (1, 2),
p (2, 1),
fn (0, 0)),
symbols)
$\wedge$ z-succ-hyps (*defns*, *symbols*)
$\wedge$ z-zerop-hyps (*defns*, *symbols*)
$\wedge$ z-sub1-hyps (*defns*, *symbols*)

$\wedge$ z-car-hyps (*defns*, *symbols*)
$\wedge$ z-cdr-hyps (*defns*, *symbols*)
$\wedge$ z-int-hyps (*defns*, *symbols*)
$\wedge$ var-set (list (*val1*, *x1*, *y1*), 3)
$\wedge$ (*concl* = a-get (ncode (x),
ncode (y),
ncode (get (x, y)),
x1,
y1,
val1,
graph-trans (get-graph (x, y)))))
$\rightarrow$ proves (a-get-ok-proof1 (x, y, *x1*, *y1*, *val1*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable a-get-ok-proof-proves; name this event ‘g0773’.

DEFINITION:
a-get1 (x, y, *val*, *x1*, *y1*, *val1*, *trace*)
= forsome (*trace*, a-get (x, y, *val*, *x1*, *y1*, *val1*, *trace*))

DEFINITION:
a-get1-ok-proof (x, y, *x1*, *y1*, *val1*, *trace*)
= eg-proof (a-get (ncode (x), ncode (y), ncode (get (x, y)), *x1*, *y1*, *val1*, *trace*),
list (*trace*),
list (graph-trans (get-graph (x, y))),
a-get-ok-proof1 (x, y, *x1*, *y1*, *val1*))

THEOREM: parallel-subst-ncode
parallel-subst (ncode (x), *vars*, *terms*, 0) = ncode (x)

THEOREM: subst-ncode
subst (ncode (x), *var*, *term*, 0) = ncode (x)

THEOREM: parallel-subst-graph-trans
parallel-subst (graph-trans (x), *vars*, *terms*, 0) = graph-trans (x)

THEOREM: subst-graph-trans
subst (graph-trans (x), *var*, *term*, 0) = graph-trans (x)

EVENT: Enable not-free-parallel-subst; name this event ‘g0755’.

EVENT: Enable not-free-del-pairs-parallel-subst; name this event ‘g0759’.

EVENT: Enable parallel-subst-nil; name this event ‘g0768’.

EVENT: Enable parallel-subst-not-free-var; name this event ‘g0771’.

EVENT: Enable not-member-del-pairs; name this event ‘g0772’.

THEOREM: free-for-nil
(collect-free (*term*, 0) = **nil**) → free-for (*exp*, *var*, *term*, *flg*)

DEFINITION:
nil-list-collect-free (*x*)
= **if** listp (*x*)
  then (collect-free (car (*x*), 0) = **nil**)
    ∧ nil-list-collect-free (cdr (*x*))
  else t endif

THEOREM: free-for-list-nil-list
nil-list-collect-free (*terms*) → free-for-list (*exp*, *vars*, *terms*)

EVENT: Enable free-for-list; name this event ‘g0774’.

THEOREM: listp-parallel-subst
listp (*exp*) → listp (parallel-subst (*exp*, *vars*, *terms*, *flg*))

THEOREM: parallel-subst-z-list
parallel-subst (z-list (*list*), *vars*, *terms*, 0)
= z-list (parallel-subst (*list*, *vars*, *terms*, 1))

EVENT: Enable listp-parallel-subst; name this event ‘g0775’.

THEOREM: a-get1-ok-proof-proves
(subset (list (fn (10, 1),
    fn (9, 1),
    fn (5, 1),
    fn (4, 1),
    fn (3, 1),
    fn (2, 2),
    fn (1, 2),
    p (1, 2),
    p (2, 1),
    fn (0, 0)),
  symbols)
∧ z-succ-hyps (*defns*, *symbols*)
∧ z-zerop-hyps (*defns*, *symbols*)
∧ z-sub1-hyps (*defns*, *symbols*)

$\wedge$ z-car-hyps (*defns*, *symbols*)
$\wedge$ z-cdr-hyps (*defns*, *symbols*)
$\wedge$ z-int-hyps (*defns*, *symbols*)
$\wedge$ var-set (list (*val1*, *x1*, *y1*, *trace*), `4`)
$\wedge$ (*concl* = a-get1 (ncode (x), ncode (y), ncode (get (x, y)), *x1*, *y1*, *val1*, *trace*)))
$\rightarrow$ proves (a-get1-ok-proof (x, y, *x1*, *y1*, *val1*, *trace*),
concl,
given,
defns,
symbols)

Event: Enable a-get1-ok-proof; name this event 'g0776'.

Theorem: a-get-good-proof-proves-again
(subset (list (fn (`10`, `1`),
fn (`9`, `1`),
fn (`5`, `1`),
fn (`4`, `1`),
fn (`3`, `1`),
fn (`2`, `2`),
fn (`1`, `2`),
p (`1`, `2`),
p (`2`, `1`),
fn (`0`, `0`)),
symbols)
$\wedge$ z-succ-hyps (*defns*, *symbols*)
$\wedge$ z-zerop-hyps (*defns*, *symbols*)
$\wedge$ z-sub1-hyps (*defns*, *symbols*)
$\wedge$ z-car-hyps (*defns*, *symbols*)
$\wedge$ z-cdr-hyps (*defns*, *symbols*)
$\wedge$ z-int-hyps (*defns*, *symbols*)
$\wedge$ termp (*val*, `0`, *symbols*)
$\wedge$ var-set (list (*x1*, *y1*, *val1*, *trace*), `4`)
$\wedge$ (*concl* = f-implies (a-get (ncode (x), ncode (y), *val*, *x1*, *y1*, *val1*, *trace*),
f-equal (*val*, ncode (get (x, y))))))
$\rightarrow$ proves (a-get-good-proof (x, y, *val*, *x1*, *y1*, *val1*, *trace*),
concl,
given,
defns,
symbols)

Event: Enable a-get-good-proof-proves; name this event 'g0777'.

Definition:

```
a-get1-good-proof (x, y, val, x1, y1, val1, trace)
=    forsome-intro-proof (trace,
                          body (a-get1 (ncode (x),
                                        ncode (y),
                                        val,
                                        x1,
                                        y1,
                                        val1,
                                        trace)),
                          f-equal (val, ncode (get (x, y))),
                          a-get-good-proof (x, y, val, x1, y1, val1, trace))
```

THEOREM: a-get1-good-proof-proves

```
 (subset (list (fn (10, 1),
                fn (9, 1),
                fn (5, 1),
                fn (4, 1),
                fn (3, 1),
                fn (2, 2),
                fn (1, 2),
                p (1, 2),
                p (2, 1),
                fn (0, 0)),
          symbols)
  ∧   z-succ-hyps (defns, symbols)
  ∧   z-zerop-hyps (defns, symbols)
  ∧   z-sub1-hyps (defns, symbols)
  ∧   z-car-hyps (defns, symbols)
  ∧   z-cdr-hyps (defns, symbols)
  ∧   z-int-hyps (defns, symbols)
  ∧   termp (val, 0, symbols)
  ∧   (trace ∉ collect-free (val, 0))
  ∧   var-set (list (x1, y1, val1, trace), 4)
  ∧   (concl = f-implies (a-get1 (ncode (x), ncode (y), val, x1, y1, val1, trace),
                          f-equal (val, ncode (get (x, y))))))
 →    proves (a-get1-good-proof (x, y, val, x1, y1, val1, trace),
              concl,
              given,
              defns,
              symbols)
```

EVENT: Enable a-get1-good-proof; name this event ‘g0778’.

```
;;;;;;;The next several pages cover the proof of the representability of the
```

```
;;;;;;;Lisp interpreter EV.  Apart from the e-expression to be evaluated,
;;;;;;;the e-variable bindings list, and the e-function symbol bindings list,
;;;;;;;the interpreter takes a counter N which is the maximum permissible
;;;;;;;depth of the evaluation stack.  If this bound is exceeded before the
;;;;;;;expression is fully evaluated, EV returns a bottom object (BTM).
;;;;;;;This object is added by as a shell by the command below.
```

EVENT: Add the shell *btm*, with recognizer function symbol *btmp* and no accessors.

DEFINITION:
equal-z-list-isin-proof(*list1*, *list2*, *exp*, *pflist*, *pf*)
= equal-isin-proof(z-list (*list1*),
z-list (*list2*),
exp,
equal-z-list-proof(*list1*, *list2*, *pflist*),
pf)

DEFINITION:
equal-z-list-isin-imp-proof(*list1*, *list2*, *exp*, *pflist*)
= equal-isin-imp-proof(z-list (*list1*),
z-list (*list2*),
exp,
equal-z-list-proof(*list1*, *list2*, *pflist*))

THEOREM: ncode-cons
ncode (cons (x, y)) = z-opair (ncode (x), ncode (y))

EVENT: Enable proves-is-formula-again; name this event 'g0781'.

EVENT: Enable termp-term-list; name this event 'g0782'.

EVENT: Enable term-list-var-list; name this event 'g0783'.

DEFINITION:
ev-equal-z-list-proof(*val*, *flg*, *exp*, *expr*, *va*, *fa*, *pf*)
= equal-z-list-proof(list (*val*, ncode (*flg*), ncode (*exp*), ncode (*va*), ncode (*fa*)),
list (*val*, ncode (*flg*), *expr*, ncode (*va*), ncode (*fa*)),
list (ident-axiom-proof(*val*),
ident-axiom-proof(ncode (*flg*)),
flip-equals-proof(*expr*, ncode (*exp*), *pf*),
ident-axiom-proof(ncode (*va*)),
ident-axiom-proof(ncode (*fa*))))

DEFINITION:
ev-equal-z-list-isin-proof(*val*, *flg*, *exp*, *expr*, *va*, *fa*, *trace*, *pf*, *in-pf*)
= equal-z-list-isin-proof(list (*val*,
ncode (*flg*),
ncode (*exp*),
ncode (*va*),
ncode (*fa*)),
list (*val*, ncode (*flg*), *expr*, ncode (*va*), ncode (*fa*)),
trace,
list (ident-axiom-proof (*val*),
ident-axiom-proof (ncode (*flg*)),
flip-equals-proof (*expr*, ncode (*exp*), *pf*),
ident-axiom-proof (ncode (*va*)),
ident-axiom-proof (ncode (*fa*))),
in-pf)

THEOREM: ev-equal-z-list-proof-proves
(subset (list (fn (`0`, `0`), fn (`1`, `2`), fn (`2`, `2`), fn (`3`, `1`)), *symbols*)
∧ termp (*val*, `0`, *symbols*)
∧ proves (*pf*, f-equal (*expr*, ncode (*exp*)), *given*, *defns*, *symbols*)
∧ (*concl* = f-equal (z-list (list (*val*,
ncode (*flg*),
ncode (*exp*),
ncode (*va*),
ncode (*fa*))),
z-list (list (*val*,
ncode (*flg*),
expr,
ncode (*va*),
ncode (*fa*))))))
→ proves (ev-equal-z-list-proof (*val*, *flg*, *exp*, *expr*, *va*, *fa*, *pf*),
concl,
given,
defns,
symbols)

EVENT: Enable ev-equal-z-list-proof; name this event 'g0789'.

THEOREM: ev-equal-z-list-isin-proof-proves
(subset (list (p (`1`, `2`), fn (`0`, `0`), fn (`1`, `2`), fn (`2`, `2`), fn (`3`, `1`)), *symbols*)
∧ termp (*val*, `0`, *symbols*)
∧ termp (*trace*, `0`, *symbols*)
∧ proves (*in-pf*,
isin (z-list (list (*val*,

```
                                  ncode (flg),
                                  ncode (exp),
                                  ncode (va),
                                  ncode (fa))),
                            trace),
                        given,
                        defns,
                        symbols)
 ∧   proves (pf, f-equal (expr, ncode (exp)), given, defns, symbols)
 ∧   (concl = isin (z-list (list (val, ncode (flg), expr, ncode (va), ncode (fa))),
                            trace)))
→    proves (ev-equal-z-list-isin-proof (val,
                                         flg,
                                         exp,
                                         expr,
                                         va,
                                         fa,
                                         trace,
                                         pf,
                                         in-pf),
                 concl,
                 given,
                 defns,
                 symbols)
```

EVENT: Enable ev-equal-z-list-isin-proof; name this event 'g0792'.

```
;;;;;;;The skeleton of the representability proof for EV is similar to that of GET, but
;;;;;;;the details are enormously more complicated since EV is an extremely complicated
;;;;;;;function.  In the first 'OK' part of the proof, we prove each section of EV to
;;;;;;;be 'OK' separately.  Below, the part of EV which evaluates Lisp if-expressions
;;;;;;;is proved 'OK'.
```

DEFINITION:

```
a-ev-defn-if1-ok-proof (exp, va, fa, val, test, trace, in-pf2, in-pf3)
=    form-z-if-ok-proof (list (list (ncode-equal (test, 0),
                                     ident-axiom-proof (ncode (test)),
                                     ncode-nequal-proof (test, 0)),
                               ev-equal-z-list-isin-proof (val,
                                                           0,
                                                           cadddr (exp),
                                                           z-car-cdr (ncode (exp),
                                                                      list (1,
                                                                            2,
```

```
                                                  2,
                                                  2)),
                                        va,
                                        fa,
                                        trace,
                                        z-car-cdr-proof(exp,
                                                        list(1,
                                                             2,
                                                             2,
                                                             2)),
                                        in-pf2),
            ev-equal-z-list-isin-proof(val,
                                        0,
                                        caddr(exp),
                                        z-car-cdr(ncode(exp),
                                             list(1,
                                                  2,
                                                  2)),
                                        va,
                                        fa,
                                        trace,
                                        z-car-cdr-proof(exp,
                                                        list(1,
                                                             2,
                                                             2)),
                                        in-pf3)),
        z-if(f-equal(ncode(test), ncode(0)),
            isin(z-list(list(val,
                             ncode(0),
                             z-car-cdr(ncode(exp),
                                       list(1, 2, 2, 2)),
                             ncode(va),
                             ncode(fa))),
                 trace),
            isin(z-list(list(val,
                             ncode(0),
                             z-car-cdr(ncode(exp),
                                       list(1, 2, 2)),
                             ncode(va),
                             ncode(fa))),
                 trace)))
```

EVENT: Enable termp-car-cdr; name this event 'g0797'.

THEOREM: a-ev-defn-if1-ok-proof-proves

```
(subset (list (fn (5, 1),
               fn (4, 1),
               fn (3, 1),
               fn (2, 2),
               fn (1, 2),
               p (1, 2),
               p (2, 1),
               fn (0, 0)),
          symbols)
 ∧   z-succ-hyps (defns, symbols)
 ∧   z-car-hyps (defns, symbols)
 ∧   z-cdr-hyps (defns, symbols)
 ∧   z-int-hyps (defns, symbols)
 ∧   termp (trace, 0, symbols)
 ∧   if ncode-equal (test, 0)
     then proves (in-pf2,
                  isin (z-list (list (ncode (val),
                                      ncode (0),
                                      ncode (cadddr (exp)),
                                      ncode (va),
                                      ncode (fa))),
                        trace),
                  given,
                  defns,
                  symbols)
     else proves (in-pf3,
                  isin (z-list (list (ncode (val),
                                      ncode (0),
                                      ncode (caddr (exp)),
                                      ncode (va),
                                      ncode (fa))),
                        trace),
                  given,
                  defns,
                  symbols) endif
 ∧   (concl = f-if (f-equal (ncode (test), ncode (0)),
                    isin (z-list (list (ncode (val),
                                        ncode (0),
                                        z-car-cdr (ncode (exp),
                                                   list (1, 2, 2, 2)),
                                        ncode (va),
                                        ncode (fa))),
                          trace),
```

```
                isin (z-list (list (ncode (val),
                                    ncode (0),
                                    z-car-cdr (ncode (exp), list (1, 2, 2)),
                                    ncode (va),
                                    ncode (fa))),
                      trace))))
 →  proves (a-ev-defn-if1-ok-proof (exp,
                                    va,
                                    fa,
                                    ncode (val),
                                    test,
                                    trace,
                                    in-pf2,
                                    in-pf3),
           concl,
           given,
           defns,
           symbols)
```

EVENT: Enable a-ev-defn-if1-ok-proof; name this event 'g0798'.

DEFINITION:

```
a-ev-if-part (exp-bar, va-bar, fa-bar, val-bar, trace, rvar1)
=   forsome (rvar1,
          f-and (isin (z-list (list (rvar1,
                                     ncode (0),
                                     z-car (z-cdr (exp-bar)),
                                     va-bar,
                                     fa-bar)),
                       trace),
                 f-if (f-equal (rvar1, ncode (0)),
                       isin (z-list (list (val-bar,
                                           ncode (0),
                                           z-car-cdr (exp-bar, list (1, 2, 2, 2)),
                                           va-bar,
                                           fa-bar)),
                             trace),
                       isin (z-list (list (val-bar,
                                           ncode (0),
                                           z-car-cdr (exp-bar, list (1, 2, 2)),
                                           va-bar,
                                           fa-bar)),
                             trace))))
```

DEFINITION:

```
a-ev-defn-if-ok-proof (exp, va, fa, val, test, rvar1, graph, in-pf1, in-pf2, in-pf3)
=   eg-proof (f-and (isin (z-list (list (rvar1,
                                         ncode (0),
                                         z-car (z-cdr (ncode (exp))),
                                         ncode (va),
                                         ncode (fa))),
                                graph-trans (graph)),
                          f-if (f-equal (rvar1, ncode (0)),
                                isin (z-list (list (ncode (val),
                                                    ncode (0),
                                                    z-car-cdr (ncode (exp),
                                                               list (1, 2, 2, 2)),
                                                    ncode (va),
                                                    ncode (fa))),
                                      graph-trans (graph)),
                                isin (z-list (list (ncode (val),
                                                    ncode (0),
                                                    z-car-cdr (ncode (exp),
                                                               list (1, 2, 2)),
                                                    ncode (va),
                                                    ncode (fa))),
                                      graph-trans (graph)))),
              list (rvar1),
              list (ncode (test)),
              and-join-proof (isin (z-list (list (ncode (test),
                                                  ncode (0),
                                                  z-car (z-cdr (ncode (exp))),
                                                  ncode (va),
                                                  ncode (fa))),
                                    graph-trans (graph)),
                              f-if (f-equal (ncode (test), ncode (0)),
                                    isin (z-list (list (ncode (val),
                                                        ncode (0),
                                                        z-car-cdr (ncode (exp),
                                                                   list (1,
                                                                         2,
                                                                         2,
                                                                         2)),
                                                        ncode (va),
                                                        ncode (fa))),
                                          graph-trans (graph)),
                                    isin (z-list (list (ncode (val),
                                                        ncode (0),
```

```
                                  z-car-cdr (ncode (exp),
                                             list (1, 2, 2)),
                                  ncode (va),
                                  ncode (fa))),
                         graph-trans (graph))),
                 ev-equal-z-list-isin-proof(ncode (test),
                                            0,
                                            cadr (exp),
                                            z-car (z-cdr (ncode (exp))),
                                            va,
                                            fa,
                                            graph-trans (graph),
                                            z-car-cdr-proof (exp,
                                                             list (1,
                                                                   2)),
                                            in-pf1),
                 a-ev-defn-if1-ok-proof (exp,
                                         va,
                                         fa,
                                         ncode (val),
                                         test,
                                         graph-trans (graph),
                                         in-pf2,
                                         in-pf3)))
```

Theorem: a-ev-defn-if-ok-proof-proves

```
 (subset (list (fn (5, 1),
                fn (4, 1),
                fn (3, 1),
                fn (2, 2),
                fn (1, 2),
                p (1, 2),
                p (2, 1),
                fn (0, 0)),
         symbols)
  ∧   z-succ-hyps (defns, symbols)
  ∧   z-car-hyps (defns, symbols)
  ∧   z-cdr-hyps (defns, symbols)
  ∧   z-int-hyps (defns, symbols)
  ∧   variable (rvar1)
  ∧   proves (in-pf1,
              isin (z-list (list (ncode (test),
                                  ncode (0),
                                  ncode (cadr (exp)),
```

```
                                    ncode (va),
                                    ncode (fa))),
                            graph-trans (graph)),
                     given,
                     defns,
                     symbols)
∧   if ncode-equal (test, 0)
    then proves (in-pf2,
                  isin (z-list (list (ncode (val),
                                      ncode (0),
                                      ncode (cadddr (exp)),
                                      ncode (va),
                                      ncode (fa))),
                        graph-trans (graph)),
                  given,
                  defns,
                  symbols)
    else proves (in-pf3,
                  isin (z-list (list (ncode (val),
                                      ncode (0),
                                      ncode (caddr (exp)),
                                      ncode (va),
                                      ncode (fa))),
                        graph-trans (graph)),
                  given,
                  defns,
                  symbols) endif
∧   (concl = a-ev-if-part (ncode (exp),
                           ncode (va),
                           ncode (fa),
                           ncode (val),
                           graph-trans (graph),
                           rvar1)))
→   proves (a-ev-defn-if-ok-proof (exp,
                                   va,
                                   fa,
                                   val,
                                   test,
                                   rvar1,
                                   graph,
                                   in-pf1,
                                   in-pf2,
                                   in-pf3),
             concl,
```

```
              given,
              defns,
              symbols)
```

EVENT: Enable a-ev-defn-if-ok-proof; name this event 'g0799'.

```
;;;;;;;The Z2 representation of the part of EV which evaluates Lisp primitives is proved
;;;;;;;'OK' below.
```

DEFINITION:
```
a-ev-subr-part (exp, va, fa, val, rvar, trace)
=   forsome (rvar,
             f-and (isin (z-list (list (rvar,
                                        ncode (1),
                                        z-cdr (ncode (exp)),
                                        ncode (va),
                                        ncode (fa))),
                          trace),
                    f-equal (val, z-apply-subr (z-car (ncode (exp)), rvar))))
```

DEFINITION:
```
a-ev-defn-subr-ok-proof (exp, va, fa, val, rval, rvar, graph, in-pf)
=   eg-proof (f-and (isin (z-list (list (rvar,
                                         ncode (1),
                                         z-cdr (ncode (exp)),
                                         ncode (va),
                                         ncode (fa))),
                           graph-trans (graph)),
                     f-equal (val, z-apply-subr (z-car (ncode (exp)), rvar))),
              list (rvar),
              list (ncode (rval)),
              and-join-proof (isin (z-list (list (ncode (rval),
                                                  ncode (1),
                                                  z-cdr (ncode (exp)),
                                                  ncode (va),
                                                  ncode (fa))),
                                    graph-trans (graph)),
                              f-equal (val,
                                       z-apply-subr (z-car (ncode (exp)),
                                                     ncode (rval))),
                              ev-equal-z-list-isin-proof (ncode (rval),
                                                          1,
                                                          cdr (exp),
                                                          z-cdr (ncode (exp)),
```

```
                                              va,
                                              fa,
                                              graph-trans (graph),
                                              z-cdr-ncode-proof (exp),
                                              in-pf ),
                   flip-equals-proof (z-apply-subr (z-car (ncode (exp)),
                                                   ncode (rval)),
                                      val,
                                      compose-proof (fn (15, 2),
                                                     val,
                                                     list (z-car (ncode (exp)),
                                                           ncode (rval)),
                                                     list (ncode (car (exp)),
                                                           ncode (rval)),
                                                     list (z-car-ncode-proof (exp),
                                                           ident-axiom-proof (ncode (rval))),
                                                     z-apply-subr-ncode-proof (car (exp),
                                                                               rval)))))
```

THEOREM: a-ev-defn-subr-ok-proof-proves

```
(subset (list (fn (15, 2),
               fn (14, 2),
               fn (13, 2),
               fn (12, 2),
               fn (0, 0),
               fn (3, 1),
               fn (4, 1),
               fn (5, 1),
               fn (6, 1),
               fn (7, 1),
               fn (8, 1),
               fn (9, 1),
               fn (10, 1),
               fn (11, 2),
               fn (1, 2),
               fn (2, 2),
               p (2, 1),
               p (1, 2)),
         symbols)
 ∧  z-apply-subr-all-hyps (given, defns, symbols)
 ∧  variable (rvar)
 ∧  proves (in-pf ,
           isin (z-list (list (ncode (rval),
                               ncode (1),
```

$$
\begin{array}{ll}
 & \qquad\qquad\qquad\qquad \text{ncode}\,(\text{cdr}\,(exp)), \\
 & \qquad\qquad\qquad\qquad \text{ncode}\,(va), \\
 & \qquad\qquad\qquad\qquad \text{ncode}\,(fa))), \\
 & \qquad\qquad\qquad \text{graph-trans}\,(graph)), \\
 & \qquad\quad given, \\
 & \qquad\quad defns, \\
 & \qquad\quad symbols) \\
\wedge & (val = \text{ncode}\,(\text{apply-subr}\,(\text{car}\,(exp),\ rval))) \\
\wedge & (concl = \text{a-ev-subr-part}\,(exp,\ va,\ fa,\ val,\ rvar,\ \text{graph-trans}\,(graph)))) \\
\rightarrow & \text{proves}\,(\text{a-ev-defn-subr-ok-proof}\,(exp,\ va,\ fa,\ val,\ rval,\ rvar,\ graph,\ in\text{-}pf), \\
 & \qquad\quad concl, \\
 & \qquad\quad given, \\
 & \qquad\quad defns, \\
 & \qquad\quad symbols)
\end{array}
$$

EVENT: Enable a-ev-defn-subr-ok-proof; name this event ‘g0802’.

EVENT: Enable get; name this event ‘g0803’.

EVENT: Enable pairlist; name this event ‘g0809’.

EVENT: Enable not-free-covering; name this event ‘g0816’.

```
;;;;;;;The e-functions symbols 0 is used for the Lisp function IF, the symbols 1-9
;;;;;;;are used for the Lisp primitives, 10 is used for the QUOTE function.  The defined
;;;;;;;e-function symbols therefore start from 11.  Since the first definition in the
;;;;;;;function definition list corresponds to 11, we have to subtract 11 from the symbol
;;;;;;;before we apply GET to the definition list.  the function SUB1-N subtracts N
;;;;;;;from EXP.  It is represented in Z2 by an iterated application of Z-SUB1.
```

DEFINITION:
z-sub1-n $(exp,\ n)$
$=$ **if** $n \simeq 0$ **then** exp
else z-sub1 (z-sub1-n $(exp,\ n-1)$) **endif**

DEFINITION:
sub1-n $(exp,\ n)$
$=$ **if** $n \simeq 0$ **then** exp
else sub1-n $(exp,\ n-1) - 1$ **endif**

DEFINITION:
z-sub1-n-ncode-proof $(exp,\ n,\ val,\ pf)$
$=$ **if** $n \simeq 0$ **then** pf

else compose-proof (fn (10, 1),
    ncode (sub1-n (*val*, *n*)),
    list (z-sub1-n (*exp*, $n-1$)),
    list (ncode (sub1-n (*val*, $n-1$))),
    list (z-sub1-n-ncode-proof (*exp*, $n-1$, *val*, *pf*)),
    z-sub1-ncode-proof (sub1-n (*val*, $n-1$))) **endif**

THEOREM: term-z-sub1-n
((fn (10, 1) $\in$ *symbols*) $\wedge$ termp (*exp*, 0, *symbols*))
$\rightarrow$ termp (z-sub1-n (*exp*, *n*), 0, *symbols*)

THEOREM: z-sub1-n-ncode-proof-proves
(subset (list (fn (10, 1), fn (3, 1), p (2, 1), p (1, 2), fn (2, 2), fn (1, 2), fn (0, 0)),
symbols)
$\wedge$ z-succ-hyps (*defns*, *symbols*)
$\wedge$ z-sub1-hyps (*defns*, *symbols*)
$\wedge$ z-int-hyps (*defns*, *symbols*)
$\wedge$ proves (*pf*, f-equal (*exp*, ncode (*val*)), *given*, *defns*, *symbols*)
$\wedge$ (*concl* = f-equal (z-sub1-n (*exp*, *n*), ncode (sub1-n (*val*, *n*)))))
$\rightarrow$ proves (z-sub1-n-ncode-proof (*exp*, *n*, *val*, *pf*), *concl*, *given*, *defns*, *symbols*)

EVENT: Enable z-sub1-n-ncode-proof; name this event 'g0817'.

EVENT: Enable sub1-n; name this event 'g0818'.

```
;;;;;;;Next we prove the representation of that part of EV which evaluates defined Lisp
;;;;;;;functions, to be 'OK'.
```

DEFINITION:
a-ev-fun-part (*exp*, *va*, *fa*, *val*, *trace*)
= forsome (11,
    list-all-quantify (list (15, 16),
        f-and (isin (z-list (list (11,
                    ncode (1),
                    z-cdr (ncode (*exp*)),
                    ncode (*va*),
                    ncode (*fa*))),
                trace),
            f-implies (a-get (z-sub1-n (z-car (ncode (*exp*)),
                        11),
                    ncode (*fa*),
                    15,
                    11,
                    13,

```
                                          14,
                                          16),
                                   isin (z-list (list (val,
                                                       ncode (0),
                                                       15,
                                                       11,
                                                       ncode (fa))),
                                         trace)))))
```

THEOREM: formula-list-all-quantify

```
 (var-list (vars, length (vars)) ∧ formula (exp, symbols))
 →   formula (list-all-quantify (vars, exp), symbols)
```

THEOREM: list-gen-proof-proves-again

```
 (var-list (vars, length (vars))
  ∧   proves (pf, exp, given, defns, symbols)
  ∧   (concl = list-all-quantify (vars, exp)))
 →   proves (list-gen-proof (vars, exp, pf), concl, given, defns, symbols)
```

DEFINITION:

```
a-get-ev-reduc-proof (exp, fa, getval, n, x1, y1, val1, tr1)
=    tautconseq-proof (list (f-implies (isin (z-list (list (getval,
                                                            z-sub1-n (z-car (ncode (exp)),
                                                                      n),
                                                            ncode (fa))),
                                                      tr1),
                                                isin (z-list (list (getval,
                                                            ncode (sub1-n (car (exp),
                                                                           n)),
                                                            ncode (fa))),
                                                      tr1)),
                                     f-implies (a-get (ncode (sub1-n (car (exp), n)),
                                                       ncode (fa),
                                                       getval,
                                                       x1,
                                                       y1,
                                                       val1,
                                                       tr1),
                                                f-equal (getval,
                                                         ncode (get (sub1-n (car (exp), n),
                                                                     fa))))),
                             f-implies (a-get (z-sub1-n (z-car (ncode (exp)), n),
                                               ncode (fa),
                                               getval,
                                               x1,
```

```
                                        y1,
                                        val1,
                                        tr1),
                              f-equal (getval,
                                       ncode (get (sub1-n (car (exp), n), fa)))),
                 list (equal-z-list-isin-imp-proof (list (getval,
                                                          z-sub1-n (z-car (ncode (exp)),
                                                                    n),
                                                          ncode (fa)),
                                                     list (getval,
                                                          ncode (sub1-n (car (exp),
                                                                         n)),
                                                          ncode (fa)),
                                                     tr1,
                                                     list (ident-axiom-proof (getval),
                                                          z-sub1-n-ncode-proof (z-car (ncode (exp)),
                                                                                n,
                                                                                car (exp),
                                                                                z-car-ncode-proof (exp)),
                                                          ident-axiom-proof (ncode (fa)))),
                       a-get-good-proof (sub1-n (car (exp), n),
                                         fa,
                                         getval,
                                         x1,
                                         y1,
                                         val1,
                                         tr1)))
```

DEFINITION:

```
a-ev-defn-fun-ok-proof (exp, va, fa, val, rval, graph, in-pf1, in-pf2)
=    eg-proof (list-all-quantify (list (15, 16),
                                  f-and (isin (z-list (list (11,
                                                             ncode (1),
                                                             z-cdr (ncode (exp)),
                                                             ncode (va),
                                                             ncode (fa))),
                                               graph-trans (graph)),
                                         f-implies (a-get (z-sub1-n (z-car (ncode (exp)),
                                                                      11),
                                                            ncode (fa),
                                                            15,
                                                            11,
                                                            13,
                                                            14,
```

```
                                                  16),
                                          isin (z-list (list (val,
                                                             ncode (0),
                                                             15,
                                                             11,
                                                             ncode (fa))),
                                                graph-trans (graph))))),
list (11),
list (ncode (rval)),
list-gen-proof (list (15, 16),
                f-and (isin (z-list (list (ncode (rval),
                                           ncode (1),
                                           z-cdr (ncode (exp)),
                                           ncode (va),
                                           ncode (fa))),
                             graph-trans (graph)),
                       f-implies (a-get (z-sub1-n (z-car (ncode (exp)),
                                                          11),
                                         ncode (fa),
                                         15,
                                         11,
                                         13,
                                         14,
                                         16),
                                  isin (z-list (list (val,
                                                      ncode (0),
                                                      15,
                                                      ncode (rval),
                                                      ncode (fa))),
                                        graph-trans (graph)))),
                and-join-proof (isin (z-list (list (ncode (rval),
                                                    ncode (1),
                                                    z-cdr (ncode (exp)),
                                                    ncode (va),
                                                    ncode (fa))),
                                      graph-trans (graph)),
                                f-implies (a-get (z-sub1-n (z-car (ncode (exp)),
                                                                   11),
                                                  ncode (fa),
                                                  15,
                                                  11,
                                                  13,
                                                  14,
                                                  16),
```

```
isin (z-list (list (val,
                    ncode (0),
                    15,
                    ncode (rval),
                    ncode (fa))),
          graph-trans (graph))),
ev-equal-z-list-isin-proof(ncode (rval),
                           1,
                           cdr (exp),
                           z-cdr (ncode (exp)),
                           va,
                           fa,
                           graph-trans (graph),
                           z-cdr-ncode-proof (exp),
                           in-pf1 ),
chain-proof (a-get (z-sub1-n (z-car (ncode (exp)),
                              11),
                    ncode (fa),
                    15,
                    11,
                    13,
                    14,
                    16),
             isin (z-list (list (val,
                                 ncode (0),
                                 15,
                                 ncode (rval),
                                 ncode (fa))),
                   graph-trans (graph)),
             list (f-equal (15,
                            ncode (get (sub1-n (car (exp),
                                                11),
                                        fa)))),
             list (a-get-ev-reduc-proof (exp,
                                         fa,
                                         15,
                                         11,
                                         11,
                                         13,
                                         14,
                                         16),
                   form-equal-case-proof (list (15),
                                          list (ncode (get (sub1-n (car (exp),
                                                                    11),
```

fa))),
isin (z-list (list (*val*,
ncode (0),
15,
ncode (*rval*),
ncode (*fa*))),
graph-trans (*graph*)),
in-pf2))))))

THEOREM: parallel-subst-z-sub1-n
parallel-subst (z-sub1-n (*exp*, *n*), *vars*, *terms*, 0)
= z-sub1-n (parallel-subst (*exp*, *vars*, *terms*, 0), *n*)

THEOREM: a-ev-defn-fun-ok-proof-proves
(subset (list (fn (10, 1),
fn (9, 1),
fn (5, 1),
fn (4, 1),
fn (3, 1),
fn (2, 2),
fn (1, 2),
p (1, 2),
p (2, 1),
fn (0, 0)),
symbols)
∧ z-succ-hyps (*defns*, *symbols*)
∧ z-car-hyps (*defns*, *symbols*)
∧ z-cdr-hyps (*defns*, *symbols*)
∧ z-zerop-hyps (*defns*, *symbols*)
∧ z-sub1-hyps (*defns*, *symbols*)
∧ z-int-hyps (*defns*, *symbols*)
∧ proves (*in-pf1*,
isin (z-list (list (ncode (*rval*),
ncode (1),
ncode (cdr (*exp*)),
ncode (*va*),
ncode (*fa*))),
graph-trans (*graph*)),
given,
defns,
symbols)
∧ proves (*in-pf2*,
isin (z-list (list (ncode (*val*),
ncode (0),

```
                        ncode (get (sub1-n (car (exp), 11), fa)),
                        ncode (rval),
                        ncode (fa))),
                graph-trans (graph)),
         given,
         defns,
         symbols)
 ∧   (concl = a-ev-fun-part (exp, va, fa, ncode (val), graph-trans (graph))))
 →   proves (a-ev-defn-fun-ok-proof (exp,
                                     va,
                                     fa,
                                     ncode (val),
                                     rval,
                                     graph,
                                     in-pf1,
                                     in-pf2),
         concl,
         given,
         defns,
         symbols)
```

EVENT: Enable a-ev-defn-fun-ok-proof; name this event 'g0819'.

```
;;;;;;;Finally we prove the representation of the part of EV which evaluates lists
;;;;;;;of e-expressions to be 'OK'.
```

DEFINITION:

```
a-ev-defn-list-ok-proof (exp,
                         va,
                         fa,
                         rval1,
                         rval2,
                         rvar1,
                         rvar2,
                         graph,
                         in-pf1,
                         in-pf2)
=   eg-proof (f-and (isin (z-list (list (rvar1,
                                         ncode (0),
                                         z-car (ncode (exp)),
                                         ncode (va),
                                         ncode (fa))),
                          graph-trans (graph)),
                    f-and (isin (z-list (list (rvar2,
```

```
                        ncode (1),
                        z-cdr (ncode (exp)),
                        ncode (va),
                        ncode (fa))),
                graph-trans (graph)),
            f-equal (ncode (cons (rval1, rval2)),
                     z-opair (rvar1, rvar2)))),
list (rvar1, rvar2),
list (ncode (rval1), ncode (rval2)),
and-join-proof (isin (z-list (list (ncode (rval1),
                                    ncode (0),
                                    z-car (ncode (exp)),
                                    ncode (va),
                                    ncode (fa))),
                     graph-trans (graph)),
                f-and (isin (z-list (list (ncode (rval2),
                                           ncode (1),
                                           z-cdr (ncode (exp)),
                                           ncode (va),
                                           ncode (fa))),
                            graph-trans (graph)),
                      f-equal (ncode (cons (rval1, rval2)),
                               z-opair (ncode (rval1),
                                        ncode (rval2))))),
                ev-equal-z-list-isin-proof(ncode (rval1),
                                           0,
                                           car (exp),
                                           z-car (ncode (exp)),
                                           va,
                                           fa,
                                           graph-trans (graph),
                                           z-car-ncode-proof (exp),
                                           in-pf1),
                and-join-proof (isin (z-list (list (ncode (rval2),
                                                    ncode (1),
                                                    z-cdr (ncode (exp)),
                                                    ncode (va),
                                                    ncode (fa))),
                                     graph-trans (graph)),
                                f-equal (ncode (cons (rval1, rval2)),
                                         z-opair (ncode (rval1),
                                                  ncode (rval2))),
                                ev-equal-z-list-isin-proof(ncode (rval2),
                                                           1,
```

```
                                        cdr (exp),
                                        z-cdr (ncode (exp)),
                                        va,
                                        fa,
                                        graph-trans (graph),
                                        z-cdr-ncode-proof (exp),
                                        in-pf2),
                          ident-axiom-proof(z-opair (ncode (rval1),
                                                     ncode (rval2))))))
```

DEFINITION:

```
a-ev-list-part (exp, va, fa, val, rvar1, rvar2, trace)
=   forsome (rvar1,
        forsome (rvar2,
            f-and (isin (z-list (list (rvar1,
                                       ncode (0),
                                       z-car (ncode (exp)),
                                       ncode (va),
                                       ncode (fa))),
                         trace),
                   f-and (isin (z-list (list (rvar2,
                                              ncode (1),
                                              z-cdr (ncode (exp)),
                                              ncode (va),
                                              ncode (fa))),
                                trace),
                          f-equal (val, z-opair (rvar1, rvar2))))))
```

THEOREM: a-ev-defn-list-ok-proof-proves

```
 (subset (list (fn (5, 1),
                fn (4, 1),
                fn (3, 1),
                fn (2, 2),
                fn (1, 2),
                p (1, 2),
                p (2, 1),
                fn (0, 0)),
          symbols)
  ∧   z-succ-hyps (defns, symbols)
  ∧   z-car-hyps (defns, symbols)
  ∧   z-cdr-hyps (defns, symbols)
  ∧   z-int-hyps (defns, symbols)
  ∧   var-set (list (rvar1, rvar2), 2)
  ∧   proves (in-pf1,
```

```
              isin (z-list (list (ncode (rval1),
                                  ncode (0),
                                  ncode (car (exp)),
                                  ncode (va),
                                  ncode (fa))),
                    graph-trans (graph)),
              given,
              defns,
              symbols)
  ∧   proves (in-pf2,
              isin (z-list (list (ncode (rval2),
                                  ncode (1),
                                  ncode (cdr (exp)),
                                  ncode (va),
                                  ncode (fa))),
                    graph-trans (graph)),
              given,
              defns,
              symbols)
  ∧   (concl = a-ev-list-part (exp,
                               va,
                               fa,
                               ncode (cons (rval1, rval2)),
                               rvar1,
                               rvar2,
                               graph-trans (graph))))
 →    proves (a-ev-defn-list-ok-proof (exp,
                                       va,
                                       fa,
                                       rval1,
                                       rval2,
                                       rvar1,
                                       rvar2,
                                       graph,
                                       in-pf1,
                                       in-pf2),
              concl,
              given,
              defns,
              symbols)
```

EVENT: Enable a-ev-defn-list-ok-proof; name this event 'g0823'.

```
;;;;;;;EV is defined below.  It takes a FLG to control the recursion, an e-expression EXP,
```

```
;;;;;;;an e-variable bindings list VA, a e-function symbols definition list FA, and a stack
;;;;;;;bound N.  If the FLG is 1, it is evaluating a list of e-expressions - if any of them
;;;;;;;evaluates to (BTM), the whole list evaluates to (BTM).  Otherwise, it returns the
;;;;;;;list of values.  If the FLG is not 1, then if EXP is not a list, it must be an
;;;;;;;e-variable and EV returns (GET EXP VA).  Otherwise, (CAR EXP) is an e-function
;;;;;;;symbol.  If the symbol is 0, then the e-expression is evaluated as a Lisp
;;;;;;;IF-expression.  If the symbol is 10, then the e-expression is evaluated as a
;;;;;;;Lisp QUOTE-expression.  If the symbol is between 1 and 9, then the appropriate
;;;;;;;Lisp primitive is applied to the recursively evaluated arguments.  In the remaining
;;;;;;;case, the e-function symbol is a defined one, and the corresponding definition is
;;;;;;;evaluated with a new e-variable-bindings list consisting of the recursively evaluated
;;;;;;;arguments.  Since there is no guarantee in the last case that the size of the
;;;;;;;definition is going to be smaller than the e-expression being evaluated, the counter
;;;;;;;N is decremented by one.  If either the counter is exhausted or one of the
;;;;;;;evaluated sub-expressions evaluates to (BTM), then EV returns (BTM).
```

DEFINITION:
ev (*flg*, *exp*, *va*, *fa*, *n*)
= **if** *flg* = 1
  then if listp (*exp*)
    then if btmp (ev (0, car (*exp*), *va*, *fa*, *n*))
        ∨ btmp (ev (1, cdr (*exp*), *va*, *fa*, *n*)) **then** BTM
      else cons (ev (0, car (*exp*), *va*, *fa*, *n*),
          ev (1, cdr (*exp*), *va*, *fa*, *n*)) **endif**
    else nil endif
  elseif listp (*exp*)
  then if car (*exp*) ≃ 0
    then if btmp (ev (0, cadr (*exp*), *va*, *fa*, *n*)) **then** BTM
      elseif ncode-equal (ev (0, cadr (*exp*), *va*, *fa*, *n*), 0)
      then ev (0, cadddr (*exp*), *va*, *fa*, *n*)
      else ev (0, caddr (*exp*), *va*, *fa*, *n*) **endif**
    elseif car (*exp*) = 10 **then** cadr (*exp*)
    elseif btmp (ev (1, cdr (*exp*), *va*, *fa*, *n*)) **then** BTM
    elseif subrp (car (*exp*))
    then apply-subr (car (*exp*), ev (1, cdr (*exp*), *va*, *fa*, *n*))
    elseif $n \simeq 0$ **then** BTM
    else ev (0,
        get (sub1-n (car (*exp*), 11), *fa*),
        ev (1, cdr (*exp*), *va*, *fa*, *n*),
        fa,
        $n - 1$) **endif**
  else get (*exp*, *va*) **endif**

THEOREM: ev-if-reduc

$((flg \neq 1)$
$\wedge$ listp (exp)
$\wedge$ (car $(exp) \simeq 0)$
$\wedge$ $(\neg$ btmp (ev $(flg,\ exp,\ va,\ fa,\ n))))$
$\rightarrow$ (ev $(flg,\ exp,\ va,\ fa,\ n)$
= **if** ncode-equal (ev (0, cadr $(exp),\ va,\ fa,\ n)$, 0)
then ev (0, cadddr $(exp),\ va,\ fa,\ n)$
else ev (0, caddr $(exp),\ va,\ fa,\ n)$ **endif**)

THEOREM: ev-subr-reduc
$((flg \neq 1)$
$\wedge$ listp (exp)
$\wedge$ (car $(exp) \not\simeq 0)$
$\wedge$ (car $(exp) \neq 10)$
$\wedge$ subrp (car (exp)))
$\wedge$ $(\neg$ btmp (ev $(flg,\ exp,\ va,\ fa,\ n))))$
$\rightarrow$ (ev $(flg,\ exp,\ va,\ fa,\ n)$ = apply-subr (car (exp), ev (1, cdr $(exp),\ va,\ fa,\ n)))$

THEOREM: ev-fun-reduc
$((flg \neq 1)$
$\wedge$ listp (exp)
$\wedge$ (car $(exp) \not\simeq 0)$
$\wedge$ (car $(exp) \neq 10)$
$\wedge$ $(\neg$ subrp (car (exp))))
$\wedge$ $(\neg$ btmp (ev $(flg,\ exp,\ va,\ fa,\ n))))$
$\rightarrow$ (ev $(flg,\ exp,\ va,\ fa,\ n)$
= ev (0,
get (sub1-n (car (exp), 11), fa),
ev (1, cdr $(exp),\ va,\ fa,\ n)$,
fa,
$n - 1$))

THEOREM: ev-list-reduc
$((flg = 1) \wedge$ listp $(exp) \wedge (\neg$ btmp (ev $(flg,\ exp,\ va,\ fa,\ n))))$
$\rightarrow$ (ev $(flg,\ exp,\ va,\ fa,\ n)$
= cons (ev (0, car $(exp),\ va,\ fa,\ n)$, ev (1, cdr $(exp),\ va,\ fa,\ n)))$

```
;;;;;;;The list of value/argument pairs generated in a computation of EV is constructed
;;;;;;;by EV-GRAPH below.
```

DEFINITION:
ev-graph $(flg,\ exp,\ va,\ fa,\ n)$
= **if** $flg = 1$
then if listp (exp)
then cons (cons (ev $(flg,\ exp,\ va,\ fa,\ n)$,

```
                        cons (flg, cons (exp, cons (va, fa)))),
                   append (ev-graph (0, car (exp), va, fa, n),
                           ev-graph (1, cdr (exp), va, fa, n)))
        else list (cons (ev (flg, exp, va, fa, n),
                         cons (flg, cons (exp, cons (va, fa))))) endif
    elseif listp (exp)
    then if car (exp) ≃ 0
        then if ncode-equal (ev (0, cadr (exp), va, fa, n), 0)
            then cons (cons (ev (flg, exp, va, fa, n),
                             cons (flg, cons (exp, cons (va, fa)))),
                       append (ev-graph (0, cadr (exp), va, fa, n),
                               ev-graph (0, cadddr (exp), va, fa, n)))
            else cons (cons (ev (flg, exp, va, fa, n),
                             cons (flg, cons (exp, cons (va, fa)))),
                       append (ev-graph (0, cadr (exp), va, fa, n),
                               ev-graph (0, caddr (exp), va, fa, n))) endif
        elseif car (exp) = 10
        then list (cons (ev (flg, exp, va, fa, n),
                         cons (flg, cons (exp, cons (va, fa)))))
        elseif subrp (car (exp))
        then cons (cons (ev (flg, exp, va, fa, n),
                         cons (flg, cons (exp, cons (va, fa)))),
                   ev-graph (1, cdr (exp), va, fa, n))
        elseif n ≃ 0
        then list (cons (ev (flg, exp, va, fa, n),
                         cons (flg, cons (exp, cons (va, fa)))))
        else cons (cons (ev (flg, exp, va, fa, n),
                         cons (flg, cons (exp, cons (va, fa)))),
                   append (ev-graph (0,
                                     get (sub1-n (car (exp), 11), fa),
                                     ev (1, cdr (exp), va, fa, n),
                                     fa,
                                     n − 1),
                           ev-graph (1, cdr (exp), va, fa, n))) endif
    else list (cons (get (exp, va), cons (flg, cons (exp, cons (va, fa))))) endif
```

THEOREM: member-ev-graph1

```
cons (ev (flg, exp, va, fa, n), cons (flg, cons (exp, cons (va, fa))))
∈  ev-graph (flg, exp, va, fa, n)
```

THEOREM: subr-member-graph

```
((flg ≠ 1)
 ∧  listp (exp)
 ∧  (car (exp) ≄ 0)
```

$$
\begin{array}{ll}
\land & (\mathrm{car}\,(exp) \neq \mathtt{10}) \\
\land & \mathrm{subrp}\,(\mathrm{car}\,(exp)) \\
\land & (x = \mathrm{cons}\,(\mathrm{ev}\,(\mathtt{1},\ \mathrm{cdr}\,(exp),\ va,\ fa,\ n), \\
& \qquad\qquad \mathrm{cons}\,(\mathtt{1},\ \mathrm{cons}\,(\mathrm{cdr}\,(exp),\ \mathrm{cons}\,(va,\ fa)))))) \\
\to & (x \in \text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n))
\end{array}
$$

THEOREM: fun-ev-graph

$$
\begin{array}{ll}
\multicolumn{2}{l}{((flg \neq \mathtt{1})} \\
\land & \mathrm{listp}\,(exp) \\
\land & (\mathrm{car}\,(exp) \not\simeq \mathtt{0}) \\
\land & (\mathrm{car}\,(exp) \neq \mathtt{10}) \\
\land & (\neg\ \mathrm{subrp}\,(\mathrm{car}\,(exp))) \\
\land & (\neg\ \mathrm{btmp}\,(\mathrm{ev}\,(flg,\ exp,\ va,\ fa,\ n)))) \\
\to & (\text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n) \\
& \ = \ \mathrm{cons}\,(\mathrm{cons}\,(\mathrm{ev}\,(flg,\ exp,\ va,\ fa,\ n),\ \mathrm{cons}\,(flg,\ \mathrm{cons}\,(exp,\ \mathrm{cons}\,(va,\ fa)))), \\
& \qquad\qquad \mathrm{append}\,(\text{ev-graph}\,(\mathtt{0}, \\
& \qquad\qquad\qquad\qquad\qquad \mathrm{get}\,(\text{sub1-n}\,(\mathrm{car}\,(exp),\ \mathtt{11}),\ fa), \\
& \qquad\qquad\qquad\qquad\qquad \mathrm{ev}\,(\mathtt{1},\ \mathrm{cdr}\,(exp),\ va,\ fa,\ n), \\
& \qquad\qquad\qquad\qquad\qquad fa, \\
& \qquad\qquad\qquad\qquad\qquad n - 1), \\
& \qquad\qquad\qquad \text{ev-graph}\,(\mathtt{1},\ \mathrm{cdr}\,(exp),\ va,\ fa,\ n))))
\end{array}
$$

THEOREM: member-fun-graph

$$
\begin{array}{ll}
\multicolumn{2}{l}{((flg \neq \mathtt{1})} \\
\land & \mathrm{listp}\,(exp) \\
\land & (\mathrm{car}\,(exp) \not\simeq \mathtt{0}) \\
\land & (\mathrm{car}\,(exp) \neq \mathtt{10}) \\
\land & (\neg\ \mathrm{subrp}\,(\mathrm{car}\,(exp))) \\
\land & (x = \mathrm{cons}\,(\mathrm{ev}\,(\mathtt{1},\ \mathrm{cdr}\,(exp),\ va,\ fa,\ n), \\
& \qquad\qquad \mathrm{cons}\,(\mathtt{1},\ \mathrm{cons}\,(\mathrm{cdr}\,(exp),\ \mathrm{cons}\,(va,\ fa))))) \\
\land & (y = \mathrm{cons}\,(\mathrm{ev}\,(\mathtt{0}, \\
& \qquad\qquad\qquad \mathrm{get}\,(\text{sub1-n}\,(\mathrm{car}\,(exp),\ \mathtt{11}),\ fa), \\
& \qquad\qquad\qquad \mathrm{ev}\,(\mathtt{1},\ \mathrm{cdr}\,(exp),\ va,\ fa,\ n), \\
& \qquad\qquad\qquad fa, \\
& \qquad\qquad\qquad n - 1), \\
& \qquad\qquad \mathrm{cons}\,(\mathtt{0}, \\
& \qquad\qquad\qquad \mathrm{cons}\,(\mathrm{get}\,(\text{sub1-n}\,(\mathrm{car}\,(exp),\ \mathtt{11}),\ fa), \\
& \qquad\qquad\qquad\qquad \mathrm{cons}\,(\mathrm{ev}\,(\mathtt{1},\ \mathrm{cdr}\,(exp),\ va,\ fa,\ n),\ fa))))) \\
\land & (\neg\ \mathrm{btmp}\,(\mathrm{ev}\,(flg,\ exp,\ va,\ fa,\ n)))) \\
\to & ((x \in \text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n)) \\
& \ \land \ (y \in \text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n)))
\end{array}
$$

THEOREM: if-ev-graph1

$$
\begin{array}{ll}
\multicolumn{2}{l}{((flg \neq \mathtt{1})} \\
\land & \mathrm{listp}\,(exp)
\end{array}
$$

$\wedge \quad (\mathrm{car}\,(exp) \simeq \mathtt{0})$
$\wedge \quad \text{ncode-equal}\,(\mathrm{ev}\,(\mathtt{0},\ \mathrm{cadr}\,(exp),\ va,\ fa,\ n),\ \mathtt{0}))$
$\rightarrow \quad (\text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n)$
$\quad = \quad \mathrm{cons}\,(\mathrm{cons}\,(\mathrm{ev}\,(flg,\ exp,\ va,\ fa,\ n),\ \mathrm{cons}\,(flg,\ \mathrm{cons}\,(exp,\ \mathrm{cons}\,(va,\ fa)))),$
$\qquad \mathrm{append}\,(\text{ev-graph}\,(\mathtt{0},\ \mathrm{cadr}\,(exp),\ va,\ fa,\ n),$
$\qquad\quad \text{ev-graph}\,(\mathtt{0},\ \mathrm{cadddr}\,(exp),\ va,\ fa,\ n))))$

THEOREM: if-ev-graph2
$((flg \neq \mathtt{1})$
$\wedge \quad \mathrm{listp}\,(exp)$
$\wedge \quad (\mathrm{car}\,(exp) \simeq \mathtt{0})$
$\wedge \quad (\neg\ \text{ncode-equal}\,(\mathrm{ev}\,(\mathtt{0},\ \mathrm{cadr}\,(exp),\ va,\ fa,\ n),\ \mathtt{0})))$
$\rightarrow \quad (\text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n)$
$\quad = \quad \mathrm{cons}\,(\mathrm{cons}\,(\mathrm{ev}\,(flg,\ exp,\ va,\ fa,\ n),\ \mathrm{cons}\,(flg,\ \mathrm{cons}\,(exp,\ \mathrm{cons}\,(va,\ fa)))),$
$\qquad \mathrm{append}\,(\text{ev-graph}\,(\mathtt{0},\ \mathrm{cadr}\,(exp),\ va,\ fa,\ n),$
$\qquad\quad \text{ev-graph}\,(\mathtt{0},\ \mathrm{caddr}\,(exp),\ va,\ fa,\ n))))$

THEOREM: member-if-graph1
$((flg \neq \mathtt{1})$
$\wedge \quad \mathrm{listp}\,(exp)$
$\wedge \quad (\mathrm{car}\,(exp) \simeq \mathtt{0})$
$\wedge \quad \text{ncode-equal}\,(\mathrm{ev}\,(\mathtt{0},\ \mathrm{cadr}\,(exp),\ va,\ fa,\ n),\ \mathtt{0})$
$\wedge \quad (x = \mathrm{cons}\,(\mathrm{ev}\,(\mathtt{0},\ \mathrm{cadr}\,(exp),\ va,\ fa,\ n),$
$\qquad \mathrm{cons}\,(\mathtt{0},\ \mathrm{cons}\,(\mathrm{cadr}\,(exp),\ \mathrm{cons}\,(va,\ fa)))))$
$\wedge \quad (y = \mathrm{cons}\,(\mathrm{ev}\,(\mathtt{0},\ \mathrm{cadddr}\,(exp),\ va,\ fa,\ n),$
$\qquad \mathrm{cons}\,(\mathtt{0},\ \mathrm{cons}\,(\mathrm{cadddr}\,(exp),\ \mathrm{cons}\,(va,\ fa))))))$
$\rightarrow \quad ((x \in \text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n))$
$\quad \wedge \quad (y \in \text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n)))$

THEOREM: member-if-graph2
$((flg \neq \mathtt{1})$
$\wedge \quad \mathrm{listp}\,(exp)$
$\wedge \quad (\mathrm{car}\,(exp) \simeq \mathtt{0})$
$\wedge \quad (\neg\ \text{ncode-equal}\,(\mathrm{ev}\,(\mathtt{0},\ \mathrm{cadr}\,(exp),\ va,\ fa,\ n),\ \mathtt{0}))$
$\wedge \quad (x = \mathrm{cons}\,(\mathrm{ev}\,(\mathtt{0},\ \mathrm{cadr}\,(exp),\ va,\ fa,\ n),$
$\qquad \mathrm{cons}\,(\mathtt{0},\ \mathrm{cons}\,(\mathrm{cadr}\,(exp),\ \mathrm{cons}\,(va,\ fa)))))$
$\wedge \quad (y = \mathrm{cons}\,(\mathrm{ev}\,(\mathtt{0},\ \mathrm{caddr}\,(exp),\ va,\ fa,\ n),$
$\qquad \mathrm{cons}\,(\mathtt{0},\ \mathrm{cons}\,(\mathrm{caddr}\,(exp),\ \mathrm{cons}\,(va,\ fa))))))$
$\rightarrow \quad ((x \in \text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n))$
$\quad \wedge \quad (y \in \text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n)))$

THEOREM: member-list-graph
$(\mathrm{listp}\,(exp)$
$\wedge \quad (flg = \mathtt{1})$
$\wedge \quad (x = \mathrm{cons}\,(\mathrm{ev}\,(\mathtt{0},\ \mathrm{car}\,(exp),\ va,\ fa,\ n),$

$$
\begin{array}{ll}
& \qquad\qquad \text{cons}\,(0,\ \text{cons}\,(\text{car}\,(\mathit{exp}),\ \text{cons}\,(\mathit{va},\ \mathit{fa})))))\\
\wedge & (y = \text{cons}\,(\text{ev}\,(1,\ \text{cdr}\,(\mathit{exp}),\ \mathit{va},\ \mathit{fa},\ n),\\
& \qquad\qquad \text{cons}\,(1,\ \text{cons}\,(\text{cdr}\,(\mathit{exp}),\ \text{cons}\,(\mathit{va},\ \mathit{fa})))))\\
\rightarrow & ((x \in \text{ev-graph}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n))\\
& \ \wedge\ \ (y \in \text{ev-graph}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))
\end{array}
$$

Event: Enable fun-ev-graph; name this event ‘g0824’.

Event: Enable if-ev-graph1; name this event ‘g0844’.

Event: Enable if-ev-graph2; name this event ‘g0848’.

Event: Disable proves-is-formula-again; name this event ‘g0849’.

Event: Enable nlistp-ncode; name this event ‘g0855’.

```
;;;;;;;Below is the Z2 analogue of the definition of EV.
```

Definition:
z-ev-defn-part (*flg*, *exp*, *va*, *fa*, *val*, *trace*)

$$
\begin{array}{ll}
= & \text{z-if}\,(\text{f-equal}\,(\text{ncode}\,(\mathit{flg}),\ \text{ncode}\,(1)),\\
& \quad \text{z-if}\,(\text{f-equal}\,(\text{z-listp}\,(\text{ncode}\,(\mathit{exp})),\ \text{ncode}\,(1)),\\
& \quad\quad \text{a-ev-list-part}\,(\mathit{exp},\ \mathit{va},\ \mathit{fa},\ \mathit{val},\ 11,\ 12,\ \mathit{trace}),\\
& \quad\quad \text{f-equal}\,(\mathit{val},\ \text{ncode}\,(\mathbf{nil}))),\\
& \quad \text{z-if}\,(\text{f-equal}\,(\text{z-listp}\,(\text{ncode}\,(\mathit{exp})),\ \text{ncode}\,(1)),\\
& \quad\quad \text{z-if}\,(\text{f-equal}\,(\text{z-zerop}\,(\text{z-car}\,(\text{ncode}\,(\mathit{exp}))),\ \text{ncode}\,(1)),\\
& \quad\quad\quad \text{a-ev-if-part}\,(\text{ncode}\,(\mathit{exp}),\\
& \quad\quad\quad\qquad\qquad \text{ncode}\,(\mathit{va}),\\
& \quad\quad\quad\qquad\qquad \text{ncode}\,(\mathit{fa}),\\
& \quad\quad\quad\qquad\qquad \mathit{val},\\
& \quad\quad\quad\qquad\qquad \mathit{trace},\\
& \quad\quad\quad\qquad\qquad 11),\\
& \quad\quad\quad \text{z-if}\,(\text{f-equal}\,(\text{z-car}\,(\text{ncode}\,(\mathit{exp})),\ \text{ncode}\,(10)),\\
& \quad\quad\quad\quad \text{f-equal}\,(\mathit{val},\ \text{z-car-cdr}\,(\text{ncode}\,(\mathit{exp}),\ \text{list}\,(1,\ 2))),\\
& \quad\quad\quad\quad \text{z-if}\,(\text{f-equal}\,(\text{z-subrp}\,(\text{z-car}\,(\text{ncode}\,(\mathit{exp}))),\ \text{ncode}\,(1)),\\
& \quad\quad\quad\quad\quad \text{a-ev-subr-part}\,(\mathit{exp},\ \mathit{va},\ \mathit{fa},\ \mathit{val},\ 11,\ \mathit{trace}),\\
& \quad\quad\quad\quad\quad \text{a-ev-fun-part}\,(\mathit{exp},\ \mathit{va},\ \mathit{fa},\ \mathit{val},\ \mathit{trace})))),\\
& \quad\quad \text{a-get1}\,(\text{ncode}\,(\mathit{exp}),\ \text{ncode}\,(\mathit{va}),\ \mathit{val},\ 11,\ 13,\ 14,\ 16)))
\end{array}
$$

```
(DEFN
 A-EV-DEFN-PART-OK-PFTREE
```

```
(FLG EXP VA FA N)
(LIST
 (LIST (EQUAL FLG 1) (IDENT-AXIOM-PROOF (NCODE 1)) (NCODE-NEQUAL-PROOF FLG 1))
 (LIST (LIST (LISTP EXP)
             (Z-LISTP-NCODE-PROOF EXP)
             (TRANS-NOT-EQUAL-PROOF (NCODE 1)
                                    (Z-LISTP (NCODE EXP))
                                    (NCODE 0)
                                    (NCODE-NEQUAL-PROOF 1 0)
                                    (Z-LISTP-NCODE-PROOF EXP)))
       (A-EV-DEFN-LIST-OK-PROOF EXP
                                VA
                                FA
                                (EV 0 (CAR EXP) VA FA N)
                                (EV 1 (CDR EXP) VA FA N)
                                11
                                12
                                (EV-GRAPH FLG EXP VA FA N)
                                (IN-GRAPH-PROOF (CONS (EV 0 (CAR EXP) VA FA N)
                                                      (CONS 0 (CONS (CAR EXP) (CONS VA FA))))
                                                (EV-GRAPH FLG EXP VA FA N))
                                (IN-GRAPH-PROOF (CONS (EV 1 (CDR EXP) VA FA N)
                                                      (CONS 1 (CONS (CDR EXP) (CONS VA FA))))
                                                (EV-GRAPH FLG EXP VA FA N)))
       (IDENT-AXIOM-PROOF (NCODE NIL)))
 (LIST
  (LIST (LISTP EXP)
        (Z-LISTP-NCODE-PROOF EXP)
        (TRANS-NOT-EQUAL-PROOF (NCODE 1)
                               (Z-LISTP (NCODE EXP))
                               (NCODE 0)
                               (NCODE-NEQUAL-PROOF 1 0)
                               (Z-LISTP-NCODE-PROOF EXP)))
  (LIST
   (LIST (ZEROP (CAR EXP))
         (COMPOSE-CAR-PROOF (FN 9 1) (NCODE 1) EXP (Z-ZEROP-NCODE-PROOF (CAR EXP)))
         (TRANS-NOT-EQUAL-PROOF (NCODE 1)
                                (Z-ZEROP (Z-CAR (NCODE EXP)))
                                (NCODE 0)
                                (NCODE-NEQUAL-PROOF 1 0)
                                (COMPOSE-CAR-PROOF (FN 9 1)
                                                   (NCODE 0)
                                                   EXP
                                                   (Z-ZEROP-NCODE-PROOF (CAR EXP)))))
```

```
(A-EV-DEFN-IF-OK-PROOF EXP
                       VA
                       FA
                       (EV FLG EXP VA FA N)
                       (EV 0 (CADR EXP) VA FA N)
                       11
                       (EV-GRAPH FLG EXP VA FA N)
                       (IN-GRAPH-PROOF (CONS (EV 0 (CADR EXP) VA FA N)
                                             (CONS 0 (CONS (CADR EXP) (CONS VA FA))))
                                       (EV-GRAPH FLG EXP VA FA N))
                       (IN-GRAPH-PROOF (CONS (EV 0 (CADDDR EXP) VA FA N)
                                             (CONS 0 (CONS (CADDDR EXP) (CONS VA FA))))
                                       (EV-GRAPH FLG EXP VA FA N))
                       (IN-GRAPH-PROOF (CONS (EV 0 (CADDR EXP) VA FA N)
                                             (CONS 0 (CONS (CADDR EXP) (CONS VA FA))))
                                       (EV-GRAPH FLG EXP VA FA N)))
(LIST
 (LIST (EQUAL (CAR EXP) 10)
       (Z-CAR-NCODE-PROOF EXP)
       (TRANS-NOT-EQUAL-PROOF (NCODE 10)
                              (Z-CAR (NCODE EXP))
                              (NCODE (CAR EXP))
                              (NCODE-NEQUAL-PROOF 10 (CAR EXP))
                              (Z-CAR-NCODE-PROOF EXP)))
 (FLIP-EQUALS-PROOF (Z-CAR-CDR (NCODE EXP) (LIST 1 2))
                    (NCODE (CADR EXP))
                    (Z-CAR-CDR-PROOF EXP (LIST 1 2)))
 (LIST (LIST (SUBRP (CAR EXP))
             (COMPOSE-CAR-PROOF (FN 16 1) (NCODE 1) EXP (Z-SUBRP-NCODE-PROOF (CAR EXP)))
             (TRANS-NOT-EQUAL-PROOF (NCODE 1)
                                    (Z-SUBRP (Z-CAR (NCODE EXP)))
                                    (NCODE 0)
                                    (NCODE-NEQUAL-PROOF 1 0)
                                    (COMPOSE-CAR-PROOF (FN 16 1)
                                                       (NCODE 0)
                                                       EXP
                                                       (Z-SUBRP-NCODE-PROOF (CAR EXP)))))
       (A-EV-DEFN-SUBR-OK-PROOF EXP
                                VA
                                FA
                                (NCODE (EV FLG EXP VA FA N))
                                (EV 1 (CDR EXP) VA FA N)
                                11
                                (EV-GRAPH FLG EXP VA FA N)
```

```
                (IN-GRAPH-PROOF (CONS (EV 1 (CDR EXP) VA FA N)
                                      (CONS 1
                                            (CONS (CDR EXP)
                                                  (CONS VA FA))))
                                (EV-GRAPH FLG EXP VA FA N)))
    (A-EV-DEFN-FUN-OK-PROOF EXP
                            VA
                            FA
                            (NCODE (EV FLG EXP VA FA N))
                            (EV 1 (CDR EXP) VA FA N)
                            (EV-GRAPH FLG EXP VA FA N)
                            (IN-GRAPH-PROOF (CONS (EV 1 (CDR EXP) VA FA N)
                                                  (CONS 1
                                                        (CONS (CDR EXP) (CONS VA FA))))
                                            (EV-GRAPH FLG EXP VA FA N))
                            (IN-GRAPH-PROOF (CONS (EV FLG EXP VA FA N)
                                                  (CONS 0
                                                        (CONS (GET (SUB1-N (CAR EXP)
                                                                           11)
                                                                   FA)
                                                              (CONS (EV 1
                                                                        (CDR EXP)
                                                                        VA
                                                                        FA
                                                                        N)
                                                                    FA))))
                                            (EV-GRAPH FLG EXP VA FA N))))))
(A-GET1-OK-PROOF EXP VA 11 13 14 16))))
```

Definition:

a-ev-defn-part-ok-proof (*flg*, *exp*, *va*, *fa*, *n*)
= form-z-if-ok-proof (a-ev-defn-part-ok-pftree (*flg*, *exp*, *va*, *fa*, *n*),
z-ev-defn-part (*flg*,
exp,
va,
fa,
ncode (ev (*flg*, *exp*, *va*, *fa*, *n*)),
graph-trans (ev-graph (*flg*,
exp,
va,
fa,
n))))

Theorem: form-z-if-z-ev-defn-part

```
subset (list (fn (16, 1),
              fn (15, 2),
              fn (14, 2),
              fn (13, 2),
              fn (12, 2),
              fn (0, 0),
              fn (3, 1),
              fn (4, 1),
              fn (5, 1),
              fn (6, 1),
              fn (7, 1),
              fn (8, 1),
              fn (9, 1),
              fn (10, 1),
              fn (11, 2),
              fn (1, 2),
              fn (2, 2),
              p (2, 1),
              p (1, 2)),
         symbols)
→   form-z-if (z-ev-defn-part (flg,
                               exp,
                               va,
                               fa,
                               ncode (ev (flg, exp, va, fa, n)),
                               graph-trans (ev-graph (flg, exp, va, fa, n))),
               symbols)
```

DEFINITION:

```
a-ev-defn-part (flg, exp, va, fa, val, trace)
=   f-if (f-equal (ncode (flg), ncode (1)),
        f-if (f-equal (z-listp (ncode (exp)), ncode (1)),
            a-ev-list-part (exp, va, fa, val, 11, 12, trace),
            f-equal (val, ncode (nil))),
        f-if (f-equal (z-listp (ncode (exp)), ncode (1)),
            f-if (f-equal (z-zerop (z-car (ncode (exp))), ncode (1)),
                a-ev-if-part (ncode (exp),
                              ncode (va),
                              ncode (fa),
                              val,
                              trace,
                              11),
                f-if (f-equal (z-car (ncode (exp)), ncode (10)),
                    f-equal (val, z-car-cdr (ncode (exp), list (1, 2))),
```

$$
\begin{array}{l}
\qquad\qquad\qquad\quad \text{f-if}\,(\text{f-equal}\,(\text{z-subrp}\,(\text{z-car}\,(\text{ncode}\,(\mathit{exp}))),\ \text{ncode}\,(\mathtt{1})),\\
\qquad\qquad\qquad\qquad\quad \text{a-ev-subr-part}\,(\mathit{exp},\ \mathit{va},\ \mathit{fa},\ \mathit{val},\ \mathtt{11},\ \mathit{trace}),\\
\qquad\qquad\qquad\qquad\quad \text{a-ev-fun-part}\,(\mathit{exp},\ \mathit{va},\ \mathit{fa},\ \mathit{val},\ \mathit{trace})))),\\
\qquad\qquad \text{a-get1}\,(\text{ncode}\,(\mathit{exp}),\ \text{ncode}\,(\mathit{va}),\ \mathit{val},\ \mathtt{11},\ \mathtt{13},\ \mathtt{14},\ \mathtt{16})))
\end{array}
$$

DEFINITION:

$$
\begin{array}{l}
\text{ev-hyps}\,(\mathit{given},\ \mathit{defns},\ \mathit{symbols})\\
= \quad (\text{subset}\,(\text{list}\,(\text{fn}\,(\mathtt{16},\ \mathtt{1}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{15},\ \mathtt{2}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{14},\ \mathtt{2}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{13},\ \mathtt{2}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{12},\ \mathtt{2}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{0},\ \mathtt{0}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{3},\ \mathtt{1}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{4},\ \mathtt{1}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{5},\ \mathtt{1}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{6},\ \mathtt{1}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{7},\ \mathtt{1}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{8},\ \mathtt{1}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{9},\ \mathtt{1}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{10},\ \mathtt{1}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{11},\ \mathtt{2}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{1},\ \mathtt{2}),\\
\qquad\qquad\qquad \text{fn}\,(\mathtt{2},\ \mathtt{2}),\\
\qquad\qquad\qquad \text{p}\,(\mathtt{2},\ \mathtt{1}),\\
\qquad\qquad\qquad \text{p}\,(\mathtt{1},\ \mathtt{2})),\\
\qquad\qquad \mathit{symbols})\\
\qquad \land \quad \text{z-apply-subr-all-hyps}\,(\mathit{given},\ \mathit{defns},\ \mathit{symbols})\\
\qquad \land \quad \text{z-subrp-hyps}\,(\mathit{defns},\ \mathit{symbols}))
\end{array}
$$

THEOREM: a-ev-defn-part-ok-proof-proves-quote-case

$$
\begin{array}{l}
((\mathit{flg} \neq \mathtt{1})\\
\ \land \quad \text{listp}\,(\mathit{exp})\\
\ \land \quad (\text{car}\,(\mathit{exp}) = \mathtt{10})\\
\ \land \quad \text{ev-hyps}\,(\mathit{given},\ \mathit{defns},\ \mathit{symbols})\\
\ \land \quad (\mathit{concl} = \text{a-ev-defn-part}\,(\mathit{flg},\\
\qquad\qquad\qquad\qquad\qquad \mathit{exp},\\
\qquad\qquad\qquad\qquad\qquad \mathit{va},\\
\qquad\qquad\qquad\qquad\qquad \mathit{fa},\\
\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\text{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)),\\
\qquad\qquad\qquad\qquad\qquad \text{graph-trans}\,(\text{ev-graph}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))))\\
\rightarrow \quad \text{proves}\,(\text{a-ev-defn-part-ok-proof}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n),\\
\qquad\qquad\quad \mathit{concl},\\
\qquad\qquad\quad \mathit{given},
\end{array}
$$

$defns,$
$symbols)$

THEOREM: form-z-if-z-ev-defn-part-again

$$
\begin{array}{l}
(\text{subset}\,(\text{list}\,(\text{fn}\,(\texttt{16},\ \texttt{1}),\\
\qquad\text{fn}\,(\texttt{15},\ \texttt{2}),\\
\qquad\text{fn}\,(\texttt{14},\ \texttt{2}),\\
\qquad\text{fn}\,(\texttt{13},\ \texttt{2}),\\
\qquad\text{fn}\,(\texttt{12},\ \texttt{2}),\\
\qquad\text{fn}\,(\texttt{0},\ \texttt{0}),\\
\qquad\text{fn}\,(\texttt{3},\ \texttt{1}),\\
\qquad\text{fn}\,(\texttt{4},\ \texttt{1}),\\
\qquad\text{fn}\,(\texttt{5},\ \texttt{1}),\\
\qquad\text{fn}\,(\texttt{6},\ \texttt{1}),\\
\qquad\text{fn}\,(\texttt{7},\ \texttt{1}),\\
\qquad\text{fn}\,(\texttt{8},\ \texttt{1}),\\
\qquad\text{fn}\,(\texttt{9},\ \texttt{1}),\\
\qquad\text{fn}\,(\texttt{10},\ \texttt{1}),\\
\qquad\text{fn}\,(\texttt{11},\ \texttt{2}),\\
\qquad\text{fn}\,(\texttt{1},\ \texttt{2}),\\
\qquad\text{fn}\,(\texttt{2},\ \texttt{2}),\\
\qquad\text{p}\,(\texttt{2},\ \texttt{1}),\\
\qquad\text{p}\,(\texttt{1},\ \texttt{2})),\\
\quad symbols)\\
\wedge\quad (concl = \text{z-ev-defn-part}\,(flg,\ exp,\ va,\ fa,\ \text{ncode}\,(\text{ev}\,(flg,\ exp,\ va,\ fa,\ n)),\ \text{graph-trans}\,(\text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n)))))\\
\rightarrow\quad \text{form-z-if}\,(concl,\ symbols)
\end{array}
$$

THEOREM: a-ev-defn-part-ok-proof-proves-list-case

$$
\begin{array}{l}
((flg = \texttt{1})\\
\wedge\quad \text{listp}\,(exp)\\
\wedge\quad \text{ev-hyps}\,(given,\ defns,\ symbols)\\
\wedge\quad \textbf{if } flg = \texttt{1} \textbf{ then } \neg\,\text{btmp}\,(\text{ev}\,(\texttt{1},\ exp,\ va,\ fa,\ n))\\
\qquad \textbf{else } \neg\,\text{btmp}\,(\text{ev}\,(flg,\ exp,\ va,\ fa,\ n))\ \textbf{endif}\\
\wedge\quad (concl = \text{a-ev-defn-part}\,(flg,\ exp,\ va,\ fa,\ \text{ncode}\,(\text{ev}\,(flg,\ exp,\ va,\ fa,\ n)),\ \text{graph-trans}\,(\text{ev-graph}\,(flg,\ exp,\ va,\ fa,\ n)))))
\end{array}
$$

$\rightarrow$ proves (a-ev-defn-part-ok-proof (flg, exp, va, fa, n),
$concl$,
$given$,
$defns$,
$symbols$)

THEOREM: a-ev-defn-part-ok-proof-proves-nlist-case
((flg = 1)
$\wedge$ ($exp \simeq$ **nil**)
$\wedge$ ev-hyps ($given$, $defns$, $symbols$)
$\wedge$ **if** flg = 1 **then** $\neg$ btmp (ev (1, exp, va, fa, n))
else $\neg$ btmp (ev (flg, exp, va, fa, n)) **endif**
$\wedge$ ($concl$ = a-ev-defn-part (flg,
exp,
va,
fa,
ncode (ev (flg, exp, va, fa, n)),
graph-trans (ev-graph (flg, exp, va, fa, n)))))
$\rightarrow$ proves (a-ev-defn-part-ok-proof (flg, exp, va, fa, n),
$concl$,
$given$,
$defns$,
$symbols$)

THEOREM: ncode-nequal-proof-proves-case
((0 < y)
$\wedge$ ($x \neq y$)
$\wedge$ subset (list (fn (3, 1), p (2, 1), p (1, 2), fn (2, 2), fn (1, 2), fn (0, 0)),
$symbols$)
$\wedge$ z-succ-hyps ($defns$, $symbols$)
$\wedge$ z-int-hyps ($defns$, $symbols$)
$\wedge$ ($x\text{-}bar$ = ncode (x))
$\wedge$ ($y\text{-}bar$ = ncode (y)))
$\rightarrow$ proves (ncode-nequal-proof (x, y),
f-not (f-equal ($x\text{-}bar$, $y\text{-}bar$)),
$given$,
$defns$,
$symbols$)

THEOREM: a-ev-defn-part-ok-proof-proves-if-case1
(($flg \neq$ 1)
$\wedge$ listp (exp)
$\wedge$ (car (exp) $\simeq$ 0)
$\wedge$ ncode-equal (ev (0, cadr (exp), va, fa, n), 0)
$\wedge$ ev-hyps ($given$, $defns$, $symbols$)

$$
\begin{array}{ll}
\land & (\neg\ \mathrm{btmp}\,(\mathrm{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))\\
\land & (\mathit{concl} = \text{a-ev-defn-part}\,(\mathit{flg},\\
& \qquad\qquad\qquad\qquad \mathit{exp},\\
& \qquad\qquad\qquad\qquad \mathit{va},\\
& \qquad\qquad\qquad\qquad \mathit{fa},\\
& \qquad\qquad\qquad\qquad \mathrm{ncode}\,(\mathrm{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)),\\
& \qquad\qquad\qquad\qquad \text{graph-trans}\,(\text{ev-graph}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))))\\
\to & \mathrm{proves}\,(\text{a-ev-defn-part-ok-proof}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n),\\
& \qquad \mathit{concl},\\
& \qquad \mathit{given},\\
& \qquad \mathit{defns},\\
& \qquad \mathit{symbols})
\end{array}
$$

Theorem: a-ev-defn-part-ok-proof-proves-if-case2

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\mathit{flg} \neq \mathtt{1})}\\
\land & \mathrm{listp}\,(\mathit{exp})\\
\land & (\mathrm{car}\,(\mathit{exp}) \simeq \mathtt{0})\\
\land & (\neg\ \text{ncode-equal}\,(\mathrm{ev}\,(\mathtt{0},\ \mathrm{cadr}\,(\mathit{exp}),\ \mathit{va},\ \mathit{fa},\ n),\ \mathtt{0}))\\
\land & \text{ev-hyps}\,(\mathit{given},\ \mathit{defns},\ \mathit{symbols})\\
\land & (\neg\ \mathrm{btmp}\,(\mathrm{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))\\
\land & (\mathit{concl} = \text{a-ev-defn-part}\,(\mathit{flg},\\
& \qquad\qquad\qquad\qquad \mathit{exp},\\
& \qquad\qquad\qquad\qquad \mathit{va},\\
& \qquad\qquad\qquad\qquad \mathit{fa},\\
& \qquad\qquad\qquad\qquad \mathrm{ncode}\,(\mathrm{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)),\\
& \qquad\qquad\qquad\qquad \text{graph-trans}\,(\text{ev-graph}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))))\\
\to & \mathrm{proves}\,(\text{a-ev-defn-part-ok-proof}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n),\\
& \qquad \mathit{concl},\\
& \qquad \mathit{given},\\
& \qquad \mathit{defns},\\
& \qquad \mathit{symbols})
\end{array}
$$

Theorem: a-ev-defn-part-ok-proof-proves-subr-case

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\mathit{flg} \neq \mathtt{1})}\\
\land & \mathrm{listp}\,(\mathit{exp})\\
\land & (\mathrm{car}\,(\mathit{exp}) \not\simeq \mathtt{0})\\
\land & (\mathrm{car}\,(\mathit{exp}) \neq \mathtt{10})\\
\land & \mathrm{subrp}\,(\mathrm{car}\,(\mathit{exp}))\\
\land & \text{ev-hyps}\,(\mathit{given},\ \mathit{defns},\ \mathit{symbols})\\
\land & (\neg\ \mathrm{btmp}\,(\mathrm{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))\\
\land & (\mathit{concl} = \text{a-ev-defn-part}\,(\mathit{flg},\\
& \qquad\qquad\qquad\qquad \mathit{exp},\\
& \qquad\qquad\qquad\qquad \mathit{va},\\
& \qquad\qquad\qquad\qquad \mathit{fa},
\end{array}
$$

$$\begin{array}{ll}
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{ncode}\,(\text{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)), \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{graph-trans}\,(\text{ev-graph}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n))))) \\
\rightarrow & \text{proves}\,(\text{a-ev-defn-part-ok-proof}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n), \\
& \quad\quad \mathit{concl}, \\
& \quad\quad \mathit{given}, \\
& \quad\quad \mathit{defns}, \\
& \quad\quad \mathit{symbols})
\end{array}$$

THEOREM: a-ev-defn-part-ok-proof-proves-fun-case

$$\begin{array}{ll}
\multicolumn{2}{l}{((\mathit{flg} \neq \mathtt{1})} \\
\wedge & \text{listp}\,(\mathit{exp}) \\
\wedge & (\text{car}\,(\mathit{exp}) \not\simeq \mathtt{0}) \\
\wedge & (\text{car}\,(\mathit{exp}) \neq \mathtt{10}) \\
\wedge & (\neg\ \text{subrp}\,(\text{car}\,(\mathit{exp}))) \\
\wedge & \text{ev-hyps}\,(\mathit{given},\ \mathit{defns},\ \mathit{symbols}) \\
\wedge & (\neg\ \text{btmp}\,(\text{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n))) \\
\wedge & (\mathit{concl} = \text{a-ev-defn-part}\,(\mathit{flg}, \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{exp}, \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{va}, \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{fa}, \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{ncode}\,(\text{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)), \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{graph-trans}\,(\text{ev-graph}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n))))) \\
\rightarrow & \text{proves}\,(\text{a-ev-defn-part-ok-proof}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n), \\
& \quad\quad \mathit{concl}, \\
& \quad\quad \mathit{given}, \\
& \quad\quad \mathit{defns}, \\
& \quad\quad \mathit{symbols})
\end{array}$$

THEOREM: a-ev-defn-part-ok-proof-proves-var-case

$$\begin{array}{ll}
\multicolumn{2}{l}{((\mathit{flg} \neq \mathtt{1})} \\
\wedge & (\mathit{exp} \simeq \textbf{nil}) \\
\wedge & \text{ev-hyps}\,(\mathit{given},\ \mathit{defns},\ \mathit{symbols}) \\
\wedge & \textbf{if}\ \mathit{flg} = \mathtt{1}\ \ \textbf{then}\ \neg\ \text{btmp}\,(\text{ev}\,(\mathtt{1},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)) \\
& \textbf{else}\ \neg\ \text{btmp}\,(\text{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n))\ \textbf{endif} \\
\wedge & (\mathit{concl} = \text{a-ev-defn-part}\,(\mathit{flg}, \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{exp}, \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{va}, \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{fa}, \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{ncode}\,(\text{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)), \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{graph-trans}\,(\text{ev-graph}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n))))) \\
\rightarrow & \text{proves}\,(\text{a-ev-defn-part-ok-proof}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n), \\
& \quad\quad \mathit{concl}, \\
& \quad\quad \mathit{given}, \\
& \quad\quad \mathit{defns}, \\
& \quad\quad \mathit{symbols})
\end{array}$$

THEOREM: collect-free-var-list
var-list (*vlist*, length (*vlist*))
$\rightarrow$ (collect-free (*vlist*, `1`) = append (*vlist*, **nil**))

DEFINITION:
graph-trans-append-proof (*x*, *y*, *z*, *exp*, *pf1*, *pf2*)
= **if** listp (*y*)
then graph-trans-merge-proof (car (*y*),
append (cdr (*y*), *z*),
x,
exp,
chain-proof (f-equal (*x*, ncode (car (*y*))),
exp,
list (isin (*x*, graph-trans (*y*))),
list (tautconseq-proof (list (f-implies (f-equal (*x*,
ncode (car (*y*))),
f-equal (ncode (car (*y*)),
x)),
f-equal (graph-trans (*y*),
graph-trans (*y*)),
isin (ncode (car (*y*)),
graph-trans (*y*)),
f-implies (f-equal (ncode (car (*y*)),
x),
f-implies (f-equal (graph-trans (*y*),
graph-trans (*y*)),
f-implies (isin (ncode (car (*y*)),
graph-trans (*y*)),
isin (*x*,
graph-trans (*y*)))))),
f-implies (f-equal (*x*,
ncode (car (*y*))),
isin (*x*,
graph-trans (*y*))),
list (symmetry-proof1 (*x*,
ncode (car (*y*))),
ident-axiom-proof (graph-trans (*y*)),
in-graph-proof (car (*y*),
y),
equal-axiom2-proof (p (`1`,
`2`),
list (ncode (car (*y*)),
graph-trans (*y*)),
list (*x*,

```
                                                                                                graph-trans (y))))),
                                                   pf1)),
                                  graph-trans-append-proof (x,
                                                            cdr (y),
                                                            z,
                                                            exp,
                                                            chain-proof (isin (x,
                                                                             graph-trans (cdr (y))),
                                                                        exp,
                                                                        list (isin (x,
                                                                                 graph-trans (y))),
                                                                        list (tautconseq-proof (list (f-implies (f-or (f-equal (x,
                                                                                                                        ncode (car (y))),
                                                                                                                isin (x,
                                                                                                                     graph-trans (cdr (y)))),
                                                                                                           isin (x,
                                                                                                                graph-trans (y)))),
                                                                                              f-implies (isin (x,
                                                                                                            graph-trans (cdr (y))),
                                                                                                       isin (x,
                                                                                                            graph-trans (y))),
                                                                                              list (iff-split-proof2 (isin (x,
                                                                                                                      graph-trans (y)),
                                                                                                                 f-or (f-equal (x,
                                                                                                                             ncode (car (y))),
                                                                                                                      isin (x,
                                                                                                                           graph-trans (cdr (y)))),
                                                                                                                 in-graph-trans-proof (x,
                                                                                                                                      car (y),
                                                                                                                                      cdr (y)))))),
                                                                             pf1)),
                                                            pf2))
    else pf2 endif
```

THEOREM: graph-trans-append-proof-proves

```
 (subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), symbols)
  ∧    termp (x, 0, symbols)
  ∧    formula (exp, symbols)
  ∧    proves (pf1, f-implies (isin (x, graph-trans (y)), exp), given, defns, symbols)
  ∧    proves (pf2, f-implies (isin (x, graph-trans (z)), exp), given, defns, symbols)
  ∧    (trace = graph-trans (append (y, z))))
 →    proves (graph-trans-append-proof (x, y, z, exp, pf1, pf2),
              f-implies (isin (x, trace), exp),
              given,
```

defns,
symbols)

EVENT: Enable graph-trans-append-proof; name this event ‘g0888’.

```
;;;;;;;Since we will be manipulating large if-trees with expressions containing
;;;;;;; "in", "forall", "^", "->", and so on, so we introduce symbols (shells) for them.
```

EVENT: Add the shell *s-isin*, with recognizer function symbol *s-isinp* and 2 accessors: *s-elem*, with type restriction (none-of) and default value zero; *s-set*, with type restriction (none-of) and default value zero.

EVENT: Add the shell *s-and*, with recognizer function symbol *s-andp* and 2 accessors: *s-conj1*, with type restriction (none-of) and default value zero; *s-conj2*, with type restriction (none-of) and default value zero.

EVENT: Add the shell *s-implies*, with recognizer function symbol *s-impliesp* and 2 accessors: *s-prem*, with type restriction (none-of) and default value zero; *s-conc*, with type restriction (none-of) and default value zero.

EVENT: Add the shell *s-forall*, with recognizer function symbol *s-forallp* and 2 accessors: *s-bind*, with type restriction (one-of numberp) and default value zero; *s-body*, with type restriction (none-of) and default value zero.

DEFINITION:
weaken-isin (*form1*, *form2*, *gr1*, *gr2*)
= **if** z-ifp (*form1*) ∧ z-ifp (*form2*)
  then (z-test (*form1*) = z-test (*form2*))
    ∧ weaken-isin (z-left (*form1*), z-left (*form2*), *gr1*, *gr2*)
    ∧ weaken-isin (z-right (*form1*), z-right (*form2*), *gr1*, *gr2*)
  elseif forsomep (*form1*) ∧ forsomep (*form2*)
  then (bind (*form1*) = bind (*form2*))
    ∧ weaken-isin (body (*form1*), body (*form2*), *gr1*, *gr2*)
  elseif s-forallp (*form1*) ∧ s-forallp (*form2*)
  then (s-bind (*form1*) = s-bind (*form2*))
    ∧ weaken-isin (s-body (*form1*), s-body (*form2*), *gr1*, *gr2*)
  elseif s-andp (*form1*) ∧ s-andp (*form2*)
  then weaken-isin (s-conj1 (*form1*), s-conj1 (*form2*), *gr1*, *gr2*)
    ∧ weaken-isin (s-conj2 (*form1*), s-conj2 (*form2*), *gr1*, *gr2*)
  elseif s-impliesp (*form1*) ∧ s-impliesp (*form2*)
  then (s-prem (*form1*) = s-prem (*form2*))
    ∧ weaken-isin (s-conc (*form1*), s-conc (*form2*), *gr1*, *gr2*)

elseif s-isinp (*form1*) $\wedge$ s-isinp (*form2*)
then (s-elem (*form1*) = s-elem (*form2*))
 $\wedge$ (s-set (*form1*) = graph-trans (*gr2*))
 $\wedge$ (s-set (*form2*) = graph-trans (cons (*gr1*, *gr2*)))
else *form1* = *form2* **endif**

DEFINITION:
make-form-symb (*exp*)
= **if** z-ifp (*exp*)
then f-if (z-test (*exp*),
 make-form-symb (z-left (*exp*)),
 make-form-symb (z-right (*exp*)))
elseif forsomep (*exp*)
then forsome (bind (*exp*), make-form-symb (body (*exp*)))
elseif s-forallp (*exp*) **then** $\forall$ *s-bind*, *exp* make-form-symb (s-body (*exp*))
elseif s-andp (*exp*)
then f-and (make-form-symb (s-conj1 (*exp*)), make-form-symb (s-conj2 (*exp*)))
elseif s-impliesp (*exp*)
then f-implies (s-prem (*exp*), make-form-symb (s-conc (*exp*)))
elseif s-isinp (*exp*) **then** isin (s-elem (*exp*), s-set (*exp*))
else *exp* **endif**

DEFINITION:
weaken-isin-proof (*form1*, *form2*, *gr1*, *gr2*)
= **if** z-ifp (*form1*) $\wedge$ z-ifp (*form2*)
then tautconseq-proof (list (f-implies (make-form-symb (z-left (*form1*)),
 make-form-symb (z-left (*form2*))),
 f-implies (make-form-symb (z-right (*form1*)),
 make-form-symb (z-right (*form2*)))),
 f-implies (make-form-symb (*form1*),
 make-form-symb (*form2*)),
 list (weaken-isin-proof (z-left (*form1*),
 z-left (*form2*),
 gr1,
 gr2),
 weaken-isin-proof (z-right (*form1*),
 z-right (*form2*),
 gr1,
 gr2)))
elseif forsomep (*form1*) $\wedge$ forsomep (*form2*)
then distrib-rule-proof-a (bind (*form2*),
 make-form-symb (body (*form1*)),
 make-form-symb (body (*form2*)),
 weaken-isin-proof (body (*form1*),

```
                                                         body (form2),
                                                         gr1,
                                                         gr2))
elseif s-forallp (form1) ∧ s-forallp (form2)
then distrib-rule-proof-b (s-bind (form2),
                           make-form-symb (s-body (form1)),
                           make-form-symb (s-body (form2)),
                           weaken-isin-proof (s-body (form1),
                                              s-body (form2),
                                              gr1,
                                              gr2))
elseif s-andp (form1) ∧ s-andp (form2)
then tautconseq-proof (list (f-implies (make-form-symb (s-conj1 (form1)),
                                        make-form-symb (s-conj1 (form2))),
                             f-implies (make-form-symb (s-conj2 (form1)),
                                        make-form-symb (s-conj2 (form2)))),
                       f-implies (make-form-symb (form1),
                                  make-form-symb (form2)),
                       list (weaken-isin-proof (s-conj1 (form1),
                                                s-conj1 (form2),
                                                gr1,
                                                gr2),
                             weaken-isin-proof (s-conj2 (form1),
                                                s-conj2 (form2),
                                                gr1,
                                                gr2)))
elseif s-impliesp (form1) ∧ s-impliesp (form2)
then tautconseq-proof (list (f-implies (make-form-symb (s-conc (form1)),
                                        make-form-symb (s-conc (form2)))),
                       f-implies (make-form-symb (form1),
                                  make-form-symb (form2)),
                       list (weaken-isin-proof (s-conc (form1),
                                                s-conc (form2),
                                                gr1,
                                                gr2)))
elseif s-isinp (form1) ∧ s-isinp (form2)
then in-graph-cdr-proof (s-elem (form2), gr1, gr2)
else tautconseq-proof (nil, f-implies (form1, form2), nil) endif
```

Definition:

```
form-symb1 (exp, symbols)
=   if z-ifp (exp)
    then formula (z-test (exp), symbols)
         ∧  form-symb1 (z-left (exp), symbols)
```

```
            ∧   form-symb1 (z-right (exp), symbols)
     elseif forsomep (exp)  then form-symb1 (body (exp), symbols)
     elseif s-forallp (exp)  then form-symb1 (s-body (exp), symbols)
     elseif s-andp (exp)
     then form-symb1 (s-conj1 (exp), symbols)
            ∧   form-symb1 (s-conj2 (exp), symbols)
     elseif s-impliesp (exp)
     then formula (s-prem (exp), symbols)
            ∧   form-symb1 (s-conc (exp), symbols)
     elseif s-isinp (exp)
     then (p (1, 2) ∈ symbols)
            ∧   termp (s-elem (exp), 0, symbols)
            ∧   termp (s-set (exp), 0, symbols)
     else formula (exp, symbols) endif
```

THEOREM: formula-make-form-symb

```
form-symb1 (exp, symbols) → formula (make-form-symb (exp), symbols)
```

THEOREM: weaken-isin-proof-proves

```
 (subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), symbols)
  ∧   form-symb1 (form1, symbols)
  ∧   form-symb1 (form2, symbols)
  ∧   weaken-isin (form1, form2, gr1, gr2)
  ∧   (concl = f-implies (make-form-symb (form1), make-form-symb (form2))))
 →    proves (weaken-isin-proof (form1, form2, gr1, gr2),
              concl,
              given,
              defns,
              symbols)
```

EVENT: Enable weaken-isin-proof; name this event ‘g0859’.

DEFINITION:

```
in-append-graph-proof1 (x, gr1, gr2)
=   if listp (gr1)
    then tautconseq-proof (list (f-implies (isin (x, graph-trans (gr1)),
                                            f-or (f-equal (x, ncode (car (gr1))),
                                                  isin (x,
                                                        graph-trans (cdr (gr1))))),
                                 f-implies (f-equal (x, ncode (car (gr1))),
                                            isin (x,
                                                  graph-trans (append (gr1, gr2)))),
                                 f-implies (isin (x, graph-trans (cdr (gr1))),
                                            isin (x,
```

```
                    graph-trans (append (cdr (gr1),
                                             gr2)))),
   f-implies (isin (x,
                    graph-trans (append (cdr (gr1),
                                             gr2))),
              isin (x,
                    graph-trans (append (gr1, gr2))))),
f-implies (isin (x, graph-trans (gr1)),
           isin (x, graph-trans (append (gr1, gr2)))),
list (iff-split-proof1 (isin (x, graph-trans (gr1)),
                        f-or (f-equal (x,
                                       ncode (car (gr1))),
                              isin (x,
                                    graph-trans (cdr (gr1)))),
                        in-graph-trans-proof (x,
                                              car (gr1),
                                              cdr (gr1))),
   tautconseq-proof (list (f-implies (f-or (f-equal (x,
                                                     ncode (car (gr1))),
                                            isin (x,
                                                  graph-trans (append (cdr (gr1),
                                                                       gr2)))),
                                      isin (x,
                                            graph-trans (cons (car (gr1),
                                                               append (cdr (gr1),
                                                                       gr2)))))),
                           f-implies (f-equal (x,
                                               ncode (car (gr1))),
                                      isin (x,
                                            graph-trans (cons (car (gr1),
                                                               append (cdr (gr1),
                                                                       gr2))))),
                           list (iff-split-proof2 (isin (x,
                                                         graph-trans (cons (car (gr1),
                                                                            append (cdr (gr1),
                                                                                    gr2)))),
                                                   f-or (f-equal (x,
                                                                  ncode (car (gr1))),
                                                         isin (x,
                                                               graph-trans (append (cdr (gr1),
                                                                                    gr2)))),
                                                   in-graph-trans-proof (x,
                                                                         car (gr1),
                                                                         append (cdr (gr1),
```

```
                                                                                gr2))))),
                          in-append-graph-proof1 (x, cdr (gr1), gr2),
                          in-graph-cdr-proof (x,
                                              car (gr1),
                                              append (cdr (gr1), gr2))))
   else tautconseq-proof (list (f-not (isin (x, graph-trans (gr1)))),
                      f-implies (isin (x, graph-trans (gr1)),
                                isin (x,
                                     graph-trans (append (gr1, gr2)))),
                      list (null-axiom-proof (x))) endif
```

```
;;;;;;;Let "(X|Y)" denote (APPEND X Y).  Then |- "((x in Y) -> (x in (Y|Z)))".
```

THEOREM: in-append-graph-proof1-proves

```
 (subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), symbols)
  ∧   termp (x, 0, symbols))
 →   proves (in-append-graph-proof1 (x, gr1, gr2),
            f-implies (isin (x, graph-trans (gr1)),
                      isin (x, graph-trans (append (gr1, gr2)))),
            given,
            defns,
            symbols)
```

EVENT: Enable in-append-graph-proof1; name this event 'g0860'.

DEFINITION:

```
in-append-graph-proof2 (x, gr1, gr2)
=   if listp (gr1)
    then tautconseq-proof (list (f-implies (isin (x, graph-trans (gr2)),
                                         isin (x,
                                              graph-trans (append (cdr (gr1),
                                                                   gr2)))),
                          f-implies (isin (x,
                                          graph-trans (append (cdr (gr1),
                                                               gr2))),
                                    isin (x,
                                         graph-trans (append (gr1, gr2))))),
                      f-implies (isin (x, graph-trans (gr2)),
                                isin (x, graph-trans (append (gr1, gr2)))),
                      list (in-append-graph-proof2 (x, cdr (gr1), gr2),
                          in-graph-cdr-proof (x,
                                              car (gr1),
                                              append (cdr (gr1), gr2))))
    else tautconseq-proof (nil,
```

f-implies (isin (x, graph-trans (*gr2*)),
isin (x,
graph-trans (append (*gr1*, *gr2*)))),
nil) **endif**

```
;;;;;;; |- "((x in Z) -> (x in (Y|Z)))".
```

THEOREM: in-append-graph-proof2-proves
(subset (list (p (`1`, `2`), fn (`0`, `0`), fn (`1`, `2`), fn (`2`, `2`), fn (`3`, `1`)), *symbols*)
$\wedge$ termp (x, `0`, *symbols*))
$\rightarrow$ proves (in-append-graph-proof2 (x, *gr1*, *gr2*),
f-implies (isin (x, graph-trans (*gr2*)),
isin (x, graph-trans (append (*gr1*, *gr2*)))),
given,
defns,
symbols)

EVENT: Enable in-append-graph-proof2; name this event 'g0861'.

DEFINITION:
weaken-append1 (*form1*, *form2*, y, *gr1*, *gr2*)
= **if** z-ifp (*form1*) $\wedge$ z-ifp (*form2*)
then (z-test (*form1*) = z-test (*form2*))
$\wedge$ (z-test (*form2*) = z-test (*form2*))
$\wedge$ weaken-append1 (z-left (*form1*), z-left (*form2*), y, *gr1*, *gr2*)
$\wedge$ weaken-append1 (z-right (*form1*), z-right (*form2*), y, *gr1*, *gr2*)
elseif forsomep (*form1*) $\wedge$ forsomep (*form2*)
then (bind (*form1*) = bind (*form2*))
$\wedge$ weaken-append1 (body (*form1*), body (*form2*), y, *gr1*, *gr2*)
elseif s-forallp (*form1*) $\wedge$ s-forallp (*form2*)
then (s-bind (*form1*) = s-bind (*form2*))
$\wedge$ weaken-append1 (s-body (*form1*), s-body (*form2*), y, *gr1*, *gr2*)
elseif s-andp (*form1*) $\wedge$ s-andp (*form2*)
then weaken-append1 (s-conj1 (*form1*), s-conj1 (*form2*), y, *gr1*, *gr2*)
$\wedge$ weaken-append1 (s-conj2 (*form1*), s-conj2 (*form2*), y, *gr1*, *gr2*)
elseif s-impliesp (*form1*) $\wedge$ s-impliesp (*form2*)
then (s-prem (*form1*) = s-prem (*form2*))
$\wedge$ weaken-append1 (s-conc (*form1*), s-conc (*form2*), y, *gr1*, *gr2*)
elseif s-isinp (*form1*) $\wedge$ s-isinp (*form2*)
then (s-elem (*form1*) = s-elem (*form2*))
$\wedge$ (s-set (*form1*) = graph-trans (*gr1*))
$\wedge$ (s-set (*form2*) = graph-trans (cons (y, append (*gr1*, *gr2*))))
else *form1* = *form2* **endif**

DEFINITION:

```
weaken-append2 (form1, form2, y, gr1, gr2)
=   if z-ifp (form1) ∧ z-ifp (form2)
    then (z-test (form1) = z-test (form2))
           ∧  (z-test (form2) = z-test (form2))
           ∧  weaken-append2 (z-left (form1), z-left (form2), y, gr1, gr2)
           ∧  weaken-append2 (z-right (form1), z-right (form2), y, gr1, gr2)
    elseif forsomep (form1) ∧ forsomep (form2)
    then (bind (form1) = bind (form2))
           ∧  weaken-append2 (body (form1), body (form2), y, gr1, gr2)
    elseif s-forallp (form1) ∧ s-forallp (form2)
    then (s-bind (form1) = s-bind (form2))
           ∧  weaken-append2 (s-body (form1), s-body (form2), y, gr1, gr2)
    elseif s-andp (form1) ∧ s-andp (form2)
    then weaken-append2 (s-conj1 (form1), s-conj1 (form2), y, gr1, gr2)
           ∧  weaken-append2 (s-conj2 (form1), s-conj2 (form2), y, gr1, gr2)
    elseif s-impliesp (form1) ∧ s-impliesp (form2)
    then (s-prem (form1) = s-prem (form2))
           ∧  weaken-append2 (s-conc (form1), s-conc (form2), y, gr1, gr2)
    elseif s-isinp (form1) ∧ s-isinp (form2)
    then (s-elem (form1) = s-elem (form2))
           ∧  (s-set (form1) = graph-trans (gr2))
           ∧  (s-set (form2) = graph-trans (cons (y, append (gr1, gr2))))
    else form1 = form2 endif
```

DEFINITION:

```
weaken-append1-proof (form1, form2, y, gr1, gr2)
=   if z-ifp (form1) ∧ z-ifp (form2)
    then tautconseq-proof (list (f-implies (make-form-symb (z-left (form1)),
                                            make-form-symb (z-left (form2))),
                                 f-implies (make-form-symb (z-right (form1)),
                                            make-form-symb (z-right (form2)))),
                           f-implies (make-form-symb (form1),
                                      make-form-symb (form2)),
                           list (weaken-append1-proof (z-left (form1),
                                                       z-left (form2),
                                                       y,
                                                       gr1,
                                                       gr2),
                                 weaken-append1-proof (z-right (form1),
                                                       z-right (form2),
                                                       y,
                                                       gr1,
                                                       gr2)))
```

```
elseif forsomep (form1) ∧ forsomep (form2)
then distrib-rule-proof-a (bind (form2),
                           make-form-symb (body (form1)),
                           make-form-symb (body (form2)),
                           weaken-append1-proof (body (form1),
                                                 body (form2),
                                                 y,
                                                 gr1,
                                                 gr2))
elseif s-forallp (form1) ∧ s-forallp (form2)
then distrib-rule-proof-b (s-bind (form2),
                           make-form-symb (s-body (form1)),
                           make-form-symb (s-body (form2)),
                           weaken-append1-proof (s-body (form1),
                                                 s-body (form2),
                                                 y,
                                                 gr1,
                                                 gr2))
elseif s-andp (form1) ∧ s-andp (form2)
then tautconseq-proof (list (f-implies (make-form-symb (s-conj1 (form1)),
                                        make-form-symb (s-conj1 (form2))),
                             f-implies (make-form-symb (s-conj2 (form1)),
                                        make-form-symb (s-conj2 (form2)))),
                       f-implies (make-form-symb (form1),
                                  make-form-symb (form2)),
                       list (weaken-append1-proof (s-conj1 (form1),
                                                   s-conj1 (form2),
                                                   y,
                                                   gr1,
                                                   gr2),
                             weaken-append1-proof (s-conj2 (form1),
                                                   s-conj2 (form2),
                                                   y,
                                                   gr1,
                                                   gr2)))
elseif s-impliesp (form1) ∧ s-impliesp (form2)
then tautconseq-proof (list (f-implies (make-form-symb (s-conc (form1)),
                                        make-form-symb (s-conc (form2)))),
                       f-implies (make-form-symb (form1),
                                  make-form-symb (form2)),
                       list (weaken-append1-proof (s-conc (form1),
                                                   s-conc (form2),
                                                   y,
                                                   gr1,
```

```
                                                     gr2)))
    elseif s-isinp (form1) ∧ s-isinp (form2)
    then chain-proof (isin (s-elem (form2), graph-trans (gr1)),
                      isin (s-elem (form2),
                           graph-trans (cons (y, append (gr1, gr2)))),
                      list (isin (s-elem (form2), graph-trans (append (gr1, gr2)))),
                      list (in-append-graph-proof1 (s-elem (form2), gr1, gr2),
                           in-graph-cdr-proof (s-elem (form2),
                                               y,
                                               append (gr1, gr2))))
    else tautconseq-proof (nil, f-implies (form1, form2), nil) endif
```

Definition:

```
weaken-append2-proof (form1, form2, y, gr1, gr2)
=   if z-ifp (form1) ∧ z-ifp (form2)
    then tautconseq-proof (list (f-implies (make-form-symb (z-left (form1)),
                                            make-form-symb (z-left (form2))),
                                 f-implies (make-form-symb (z-right (form1)),
                                            make-form-symb (z-right (form2)))),
                           f-implies (make-form-symb (form1),
                                      make-form-symb (form2)),
                           list (weaken-append2-proof (z-left (form1),
                                                       z-left (form2),
                                                       y,
                                                       gr1,
                                                       gr2),
                                 weaken-append2-proof (z-right (form1),
                                                       z-right (form2),
                                                       y,
                                                       gr1,
                                                       gr2)))
    elseif forsomep (form1) ∧ forsomep (form2)
    then distrib-rule-proof-a (bind (form2),
                               make-form-symb (body (form1)),
                               make-form-symb (body (form2)),
                               weaken-append2-proof (body (form1),
                                                     body (form2),
                                                     y,
                                                     gr1,
                                                     gr2))
    elseif s-forallp (form1) ∧ s-forallp (form2)
    then distrib-rule-proof-b (s-bind (form2),
                               make-form-symb (s-body (form1)),
                               make-form-symb (s-body (form2)),
```

```
                              weaken-append2-proof (s-body (form1),
                                                    s-body (form2),
                                                    y,
                                                    gr1,
                                                    gr2))
    elseif s-andp (form1) ∧ s-andp (form2)
    then tautconseq-proof (list (f-implies (make-form-symb (s-conj1 (form1)),
                                            make-form-symb (s-conj1 (form2))),
                                 f-implies (make-form-symb (s-conj2 (form1)),
                                            make-form-symb (s-conj2 (form2)))),
                           f-implies (make-form-symb (form1),
                                      make-form-symb (form2)),
                           list (weaken-append2-proof (s-conj1 (form1),
                                                       s-conj1 (form2),
                                                       y,
                                                       gr1,
                                                       gr2),
                                 weaken-append2-proof (s-conj2 (form1),
                                                       s-conj2 (form2),
                                                       y,
                                                       gr1,
                                                       gr2)))
    elseif s-impliesp (form1) ∧ s-impliesp (form2)
    then tautconseq-proof (list (f-implies (make-form-symb (s-conc (form1)),
                                            make-form-symb (s-conc (form2)))),
                           f-implies (make-form-symb (form1),
                                      make-form-symb (form2)),
                           list (weaken-append2-proof (s-conc (form1),
                                                       s-conc (form2),
                                                       y,
                                                       gr1,
                                                       gr2)))
    elseif s-isinp (form1) ∧ s-isinp (form2)
    then chain-proof (isin (s-elem (form2), graph-trans (gr2)),
                      isin (s-elem (form2),
                            graph-trans (cons (y, append (gr1, gr2)))),
                      list (isin (s-elem (form2), graph-trans (append (gr1, gr2)))),
                      list (in-append-graph-proof2 (s-elem (form2), gr1, gr2),
                            in-graph-cdr-proof (s-elem (form2),
                                                y,
                                                append (gr1, gr2))))
    else tautconseq-proof (nil, f-implies (form1, form2), nil) endif
```

THEOREM: weaken-append1-proof-proves

(subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), *symbols*)
∧ form-symb1 (*form1*, *symbols*)
∧ form-symb1 (*form2*, *symbols*)
∧ weaken-append1 (*form1*, *form2*, *y*, *gr1*, *gr2*)
∧ (*concl* = f-implies (make-form-symb (*form1*), make-form-symb (*form2*))))
→ proves (weaken-append1-proof (*form1*, *form2*, *y*, *gr1*, *gr2*),
concl,
given,
defns,
symbols)

EVENT: Enable weaken-append1-proof; name this event 'g0862'.

THEOREM: weaken-append2-proof-proves
(subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), *symbols*)
∧ form-symb1 (*form1*, *symbols*)
∧ form-symb1 (*form2*, *symbols*)
∧ weaken-append2 (*form1*, *form2*, *y*, *gr1*, *gr2*)
∧ (*concl* = f-implies (make-form-symb (*form1*), make-form-symb (*form2*))))
→ proves (weaken-append2-proof (*form1*, *form2*, *y*, *gr1*, *gr2*),
concl,
given,
defns,
symbols)

EVENT: Enable weaken-append2-proof; name this event 'g0863'.

DEFINITION:
a-ev-defn-ok-frame-proof2 (*formals*,
y,
gr1,
gr2,
defn-part,
defn-part-form,
rec1-defn-part,
rec1-form,
rec2-defn-part,
rec2-form,
defn-pf,
recpf1,
recpf2)
= graph-trans-merge-proof (*y*,
append (*gr1*, *gr2*),

```
formals,
defn-part,
defn-pf,
graph-trans-append-proof(formals,
                         gr1,
                         gr2,
                         defn-part,
                         chain-proof(isin(formals,
                                          graph-trans(gr1)),
                                     defn-part,
                                     list(rec1-defn-part),
                                     list(recpf1,
                                          weaken-append1-proof(rec1-form,
                                                               defn-part-form,
                                                               y,
                                                               gr1,
                                                               gr2))),
                         chain-proof(isin(formals,
                                          graph-trans(gr2)),
                                     defn-part,
                                     list(rec2-defn-part),
                                     list(recpf2,
                                          weaken-append2-proof(rec2-form,
                                                               defn-part-form,
                                                               y,
                                                               gr1,
                                                               gr2)))))
```

THEOREM: a-ev-defn-ok-frame-proof2-proves

(subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), *symbols*)
$\land$ (*graph* = cons (*y*, append (*gr1*, *gr2*)))
$\land$ form-symb1 (*defn-part-form*, *symbols*)
$\land$ form-symb1 (*rec1-form*, *symbols*)
$\land$ form-symb1 (*rec2-form*, *symbols*)
$\land$ (*defn-part* = make-form-symb (*defn-part-form*))
$\land$ (*rec1-defn-part* = make-form-symb (*rec1-form*))
$\land$ (*rec2-defn-part* = make-form-symb (*rec2-form*))
$\land$ weaken-append1 (*rec1-form*, *defn-part-form*, *y*, *gr1*, *gr2*)
$\land$ weaken-append2 (*rec2-form*, *defn-part-form*, *y*, *gr1*, *gr2*)
$\land$ proves (*defn-pf*,
  f-implies (f-equal (*formals*, ncode (*y*)), *defn-part*),
  given,
  defns,
  symbols)

$\wedge$ proves (*recpf1*,
  f-implies (isin (*formals*, graph-trans (*gr1*)), *rec1-defn-part*),
  given,
  defns,
  symbols)
$\wedge$ proves (*recpf2*,
  f-implies (isin (*formals*, graph-trans (*gr2*)), *rec2-defn-part*),
  given,
  defns,
  symbols))
$\rightarrow$ proves (a-ev-defn-ok-frame-proof2 (*formals*,
  y,
  gr1,
  gr2,
  defn-part,
  defn-part-form,
  rec1-defn-part,
  rec1-form,
  rec2-defn-part,
  rec2-form,
  defn-pf,
  recpf1,
  recpf2),
  f-implies (isin (*formals*, graph-trans (*graph*)), *defn-part*),
  given,
  defns,
  symbols)

EVENT: Enable a-ev-defn-ok-frame-proof2; name this event 'g0864'.

```
(DEFN
 A-EV-SYMB-FORM
 (FLG EXP VA FA VAL TRACE)
 (Z-IF
  (F-EQUAL FLG (NCODE 1))
  (Z-IF (F-EQUAL (Z-LISTP EXP) (NCODE 1))
        (FORSOME 11
                 (FORSOME 12
                          (S-AND (S-ISIN (Z-LIST (LIST 11 (NCODE 0) (Z-CAR EXP) VA FA)) TRACE)
                                 (S-AND (S-ISIN (Z-LIST (LIST 12 (NCODE 1) (Z-CDR EXP) VA FA))
                                                TRACE)
                                        (F-EQUAL VAL (Z-OPAIR 11 12))))))
        (F-EQUAL VAL (NCODE NIL)))
```

```
(Z-IF
 (F-EQUAL (Z-LISTP EXP) (NCODE 1))
 (Z-IF
  (F-EQUAL (Z-ZEROP (Z-CAR EXP)) (NCODE 1))
  (FORSOME 11
           (S-AND (S-ISIN (Z-LIST (LIST 11 (NCODE 0) (Z-CAR (Z-CDR EXP)) VA FA)) TRACE)
                  (Z-IF (F-EQUAL 11 (NCODE 0))
                        (S-ISIN (Z-LIST (LIST VAL
                                              (NCODE 0)
                                              (Z-CAR-CDR EXP (LIST 1 2 2 2))
                                              VA
                                              FA))
                                TRACE)
                        (S-ISIN (Z-LIST (LIST VAL
                                              (NCODE 0)
                                              (Z-CAR-CDR EXP (LIST 1 2 2))
                                              VA
                                              FA))
                                TRACE))))
  (Z-IF
   (F-EQUAL (Z-CAR EXP) (NCODE 10))
   (F-EQUAL VAL (Z-CAR-CDR EXP (LIST 1 2)))
   (Z-IF
    (F-EQUAL (Z-SUBRP (Z-CAR EXP)) (NCODE 1))
    (FORSOME 11
             (S-AND (S-ISIN (Z-LIST (LIST 11 (NCODE 1) (Z-CDR EXP) VA FA)) TRACE)
                    (F-EQUAL VAL (Z-APPLY-SUBR (Z-CAR EXP) 11))))
    (FORSOME 11
             (S-FORALL 15
                       (S-FORALL 16
                                 (S-AND (S-ISIN (Z-LIST (LIST 11
                                                              (NCODE 1)
                                                              (Z-CDR EXP)
                                                              VA
                                                              FA))
                                                TRACE)
                                        (S-IMPLIES (A-GET (Z-SUB1-N (Z-CAR EXP) 11)
                                                          FA
                                                          15
                                                          11
                                                          13
                                                          14
                                                          16)
                                                   (S-ISIN (Z-LIST (LIST VAL
```

```
                                        (NCODE 0)
                                        15
                                        11
                                        FA))
                              TRACE))))))))
  (A-GET1 EXP VA VAL 11 13 14 16))))
```

DEFINITION:
a-ev-defn-part1 (*flg*, *exp*, *va*, *fa*, *val*, *trace*)

```
=   f-if (f-equal (flg, ncode (1)),
        f-if (f-equal (z-listp (exp), ncode (1)),
            forsome (11,
                forsome (12,
                    f-and (isin (z-list (list (11,
                                               ncode (0),
                                               z-car (exp),
                                               va,
                                               fa)),
                                 trace),
                           f-and (isin (z-list (list (12,
                                                      ncode (1),
                                                      z-cdr (exp),
                                                      va,
                                                      fa)),
                                        trace),
                                  f-equal (val, z-opair (11, 12)))))),
            f-equal (val, ncode (nil))),
        f-if (f-equal (z-listp (exp), ncode (1)),
            f-if (f-equal (z-zerop (z-car (exp)), ncode (1)),
                forsome (11,
                    f-and (isin (z-list (list (11,
                                               ncode (0),
                                               z-car (z-cdr (exp)),
                                               va,
                                               fa)),
                                 trace),
                           f-if (f-equal (11, ncode (0)),
                                 isin (z-list (list (val,
                                                     ncode (0),
                                                     z-car-cdr (exp,
                                                                list (1,
                                                                      2,
                                                                      2,
                                                                      2)),
```

```
                                        va,
                                        fa)),
                               trace),
                          isin (z-list (list (val,
                                        ncode (0),
                                        z-car-cdr (exp,
                                                   list (1,
                                                         2,
                                                         2)),
                                        va,
                                        fa)),
                               trace)))),
f-if (f-equal (z-car (exp), ncode (10)),
    f-equal (val, z-car-cdr (exp, list (1, 2))),
    f-if (f-equal (z-subrp (z-car (exp)), ncode (1)),
        forsome (11,
                 f-and (isin (z-list (list (11,
                                            ncode (1),
                                            z-cdr (exp),
                                            va,
                                            fa)),
                              trace),
                        f-equal (val,
                                 z-apply-subr (z-car (exp),
                                               11)))),
        forsome (11,
                 ∀ 15 ∀ 16 f-and (isin (z-list (list (11,
                                                      ncode (1),
                                                      z-cdr (exp),
                                                      va,
                                                      fa)),
                                        trace),
                                  f-implies (a-get (z-sub1-n (z-car (exp),
                                                              11),
                                                    fa,
                                                    15,
                                                    11,
                                                    13,
                                                    14,
                                                    16),
                                             isin (z-list (list (val,
                                                                 ncode (0),
                                                                 15,
                                                                 11,
```

```
                                                        fa)),
                                                  trace))))))),
      a-get1 (exp, va, val, 11, 13, 14, 16)))
```

THEOREM: a-ev-symb-form-a-ev-defn-part

```
 make-form-symb (a-ev-symb-form (ncode (flg),
                                 ncode (exp),
                                 ncode (va),
                                 ncode (fa),
                                 val,
                                 trace))
 =   a-ev-defn-part (flg, exp, va, fa, val, trace)
```

THEOREM: weaken-append2-a-ev-defn-part

```
 weaken-append2 (a-ev-symb-form (flg, exp, va, fa, val, graph-trans (gr2)),
                 a-ev-symb-form (flg,
                                 exp,
                                 va,
                                 fa,
                                 val,
                                 graph-trans (cons (y, append (gr1, gr2)))),
                 y,
                 gr1,
                 gr2)
```

THEOREM: weaken-append1-a-ev-defn-part

```
 weaken-append1 (a-ev-symb-form (flg, exp, va, fa, val, graph-trans (gr1)),
                 a-ev-symb-form (flg,
                                 exp,
                                 va,
                                 fa,
                                 val,
                                 graph-trans (cons (y, append (gr1, gr2)))),
                 y,
                 gr1,
                 gr2)
```

DEFINITION:

```
a-ev-inst-formals-proof (flg, exp, va, fa, n)
=   inst-formals-proof (list (10, 6, 7, 8, 9),
                        list (ncode (ev (flg, exp, va, fa, n)),
                              ncode (flg),
                              ncode (exp),
                              ncode (va),
                              ncode (fa)),
```

```
                    a-ev-defn-part1 (6,
                                     7,
                                     8,
                                     9,
                                     10,
                                     graph-trans (ev-graph (flg,
                                                            exp,
                                                            va,
                                                            fa,
                                                            n))),
                    a-ev-defn-part-ok-proof (flg, exp, va, fa, n))
```

THEOREM: form-symb1-a-ev-symb-form

```
 (termp (list (flg, exp, va, fa, val, trace), 1, symbols)
  ∧   subset (list (p (1, 2),
                    fn (0, 0),
                    fn (1, 2),
                    fn (2, 2),
                    fn (3, 1),
                    fn (6, 1),
                    fn (10, 1),
                    fn (16, 1),
                    fn (15, 2),
                    fn (9, 1),
                    fn (4, 1),
                    fn (5, 1)),
              symbols))
 →   form-symb1 (a-ev-symb-form (flg, exp, va, fa, val, trace), symbols)
```

THEOREM: formula-a-ev-defn-part1

```
 (subset (list (p (1, 2),
                fn (0, 0),
                fn (1, 2),
                fn (2, 2),
                fn (3, 1),
                fn (6, 1),
                fn (10, 1),
                fn (16, 1),
                fn (15, 2),
                fn (9, 1),
                fn (4, 1),
                fn (5, 1)),
          symbols)
  ∧   termp (list (flg, exp, va, fa, val, trace), 1, symbols))
 →   formula (a-ev-defn-part1 (flg, exp, va, fa, val, trace), symbols)
```

THEOREM: formula-a-ev-defn-part
(subset (list (p (1, 2),
fn (0, 0),
fn (1, 2),
fn (2, 2),
fn (3, 1),
fn (6, 1),
fn (10, 1),
fn (16, 1),
fn (15, 2),
fn (9, 1),
fn (4, 1),
fn (5, 1)),
symbols)
$\land$ termp (list (*val*, *trace*), 1, *symbols*))
$\rightarrow$ formula (a-ev-defn-part (*flg*, *exp*, *va*, *fa*, *val*, *trace*), *symbols*)

THEOREM: parallel-subst-a-ev-defn-part1
parallel-subst (a-ev-defn-part1 (6, 7, 8, 9, 10, graph-trans (*graph*)),
list (10, 6, 7, 8, 9),
list (*val*, ncode (*flg*), ncode (*exp*), ncode (*va*), ncode (*fa*)),
0)
= a-ev-defn-part (*flg*, *exp*, *va*, *fa*, *val*, graph-trans (*graph*))

EVENT: Enable a-ev-defn-part1; name this event 'g0865'.

EVENT: Enable a-ev-defn-part; name this event 'g0866'.

THEOREM: termp-z-opair-case
subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), *symbols*)
$\rightarrow$ (termp (z-opair (x, y), 0, *symbols*)
= (termp (x, 0, *symbols*) $\land$ termp (y, 0, *symbols*)))

THEOREM: graph-trans-merge-again
(subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), *symbols*)
$\land$ proves (*pf1*,
f-implies (f-equal (*exp1*, ncode (x)), *exp2*),
given,
defns,
symbols)
$\land$ proves (*pf2*,
f-implies (isin (*exp1*, graph-trans (y)), *exp2*),
given,
defns,

$symbols$))
$\rightarrow$ proves (graph-trans-merge-proof ($x,\ y,\ exp1,\ exp2,\ pf1,\ pf2$),
f-implies (isin ($exp1$, graph-trans (cons ($x,\ y$))), $exp2$),
$given$,
$defns$,
$symbols$)

EVENT: Enable graph-trans-merge; name this event 'g0867'.

DEFINITION:
a-ev-defn-ok-frame-proof3 ($formals$,
y,
gr,
$defn\text{-}part$,
$defn\text{-}part\text{-}form$,
$rec\text{-}defn\text{-}part$,
$rec\text{-}form$,
$defn\text{-}pf$,
$recpf$)
= graph-trans-merge-proof (y,
gr,
$formals$,
$defn\text{-}part$,
$defn\text{-}pf$,
chain-proof (isin ($formals$, graph-trans (gr)),
$defn\text{-}part$,
list ($rec\text{-}defn\text{-}part$),
list ($recpf$,
weaken-isin-proof ($rec\text{-}form$,
$defn\text{-}part\text{-}form$,
y,
gr))))

THEOREM: a-ev-defn-ok-frame-proof3-proves
(subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), $symbols$)
$\wedge$ ($graph$ = cons (y, gr))
$\wedge$ form-symb1 ($defn\text{-}part\text{-}form$, $symbols$)
$\wedge$ form-symb1 ($rec\text{-}form$, $symbols$)
$\wedge$ ($defn\text{-}part$ = make-form-symb ($defn\text{-}part\text{-}form$))
$\wedge$ ($rec\text{-}defn\text{-}part$ = make-form-symb ($rec\text{-}form$))
$\wedge$ weaken-isin ($rec\text{-}form$, $defn\text{-}part\text{-}form$, y, gr)
$\wedge$ proves ($defn\text{-}pf$,
f-implies (f-equal ($formals$, ncode (y)), $defn\text{-}part$),
$given$,

$\quad\quad$ *defns*,
$\quad\quad$ *symbols*)
$\land$ $\quad$ proves (*recpf*,
$\quad\quad$ f-implies (isin (*formals*, graph-trans (*gr*)), *rec-defn-part*),
$\quad\quad$ *given*,
$\quad\quad$ *defns*,
$\quad\quad$ *symbols*))
$\rightarrow$ $\quad$ proves (a-ev-defn-ok-frame-proof3 (*formals*,
$\quad\quad\quad\quad$ *y*,
$\quad\quad\quad\quad$ *gr*,
$\quad\quad\quad\quad$ *defn-part*,
$\quad\quad\quad\quad$ *defn-part-form*,
$\quad\quad\quad\quad$ *rec-defn-part*,
$\quad\quad\quad\quad$ *rec-form*,
$\quad\quad\quad\quad$ *defn-pf*,
$\quad\quad\quad\quad$ *recpf*),
$\quad\quad$ f-implies (isin (*formals*, graph-trans (*graph*)), *defn-part*),
$\quad\quad$ *given*,
$\quad\quad$ *defns*,
$\quad\quad$ *symbols*)

EVENT: Enable a-ev-defn-ok-frame-proof3; name this event ‘g0869’.

THEOREM: weaken-isin-a-ev-defn-part
weaken-isin (a-ev-symb-form (*flg*, *exp*, *va*, *fa*, *val*, graph-trans (*gr*)),
$\quad\quad$ a-ev-symb-form (*flg*, *exp*, *va*, *fa*, *val*, graph-trans (cons (*y*, *gr*))),
$\quad\quad$ *y*,
$\quad\quad$ *gr*)

THEOREM: term-ncode-case
subset (list (p (`1`, `2`), fn (`0`, `0`), fn (`1`, `2`), fn (`2`, `2`), fn (`3`, `1`)), *symbols*)
$\rightarrow$ $\quad$ termp (ncode (*x*), `0`, *symbols*)

THEOREM: term-graph-trans-case
subset (list (p (`1`, `2`), fn (`0`, `0`), fn (`1`, `2`), fn (`2`, `2`), fn (`3`, `1`)), *symbols*)
$\rightarrow$ $\quad$ termp (graph-trans (*x*), `0`, *symbols*)

THEOREM: a-ev-symb-form-a-ev-defn-part1
make-form-symb (a-ev-symb-form (*flg*, *exp*, *va*, *fa*, *val*, *trace*))
$=$ $\quad$ a-ev-defn-part1 (*flg*, *exp*, *va*, *fa*, *val*, *trace*)

EVENT: Enable a-ev-symb-form; name this event ‘g0890’.

EVENT: Enable a-ev-defn-part-ok-proof; name this event ‘g0891’.

DEFINITION:
a-ev-defn-ok-proof (*flg*, *exp*, *va*, *fa*, *n*, *flg1*, *exp1*, *va1*, *fa1*, *val1*)

```
=   if flg = 1
    then if listp (exp)
         then a-ev-defn-ok-frame-proof2 (z-list (list (val1,
                                                        flg1,
                                                        exp1,
                                                        va1,
                                                        fa1)),
                                          cons (ev (flg, exp, va, fa, n),
                                                cons (flg,
                                                      cons (exp, cons (va, fa)))),
                                          ev-graph (0, car (exp), va, fa, n),
                                          ev-graph (1, cdr (exp), va, fa, n),
                                          a-ev-defn-part1 (flg1,
                                                           exp1,
                                                           va1,
                                                           fa1,
                                                           val1,
                                                           graph-trans (ev-graph (flg,
                                                                                  exp,
                                                                                  va,
                                                                                  fa,
                                                                                  n))),
                                          a-ev-symb-form (flg1,
                                                          exp1,
                                                          va1,
                                                          fa1,
                                                          val1,
                                                          graph-trans (ev-graph (flg,
                                                                                 exp,
                                                                                 va,
                                                                                 fa,
                                                                                 n))),
                                          a-ev-defn-part1 (flg1,
                                                           exp1,
                                                           va1,
                                                           fa1,
                                                           val1,
                                                           graph-trans (ev-graph (0,
                                                                                  car (exp),
                                                                                  va,
                                                                                  fa,
                                                                                  n))),
```

```
a-ev-symb-form (flg1,
                exp1,
                va1,
                fa1,
                val1,
                graph-trans (ev-graph (0,
                                       car (exp),
                                       va,
                                       fa,
                                       n))),
a-ev-defn-part1 (flg1,
                 exp1,
                 va1,
                 fa1,
                 val1,
                 graph-trans (ev-graph (1,
                                        cdr (exp),
                                        va,
                                        fa,
                                        n))),
a-ev-symb-form (flg1,
                exp1,
                va1,
                fa1,
                val1,
                graph-trans (ev-graph (1,
                                       cdr (exp),
                                       va,
                                       fa,
                                       n))),
a-ev-inst-formals-proof (flg,
                         exp,
                         va,
                         fa,
                         n),
a-ev-defn-ok-proof (0,
                    car (exp),
                    va,
                    fa,
                    n,
                    flg1,
                    exp1,
                    va1,
                    fa1,
```

```
                                             val1),
                    a-ev-defn-ok-proof(1,
                                       cdr(exp),
                                       va,
                                       fa,
                                       n,
                                       flg1,
                                       exp1,
                                       va1,
                                       fa1,
                                       val1))
else graph-trans-merge-proof(cons(ev(flg, exp, va, fa, n),
                                  cons(flg,
                                       cons(exp,
                                            cons(va, fa)))),
                             nil,
                             z-list(list(val1,
                                         flg1,
                                         exp1,
                                         va1,
                                         fa1)),
                             a-ev-defn-part1(flg1,
                                             exp1,
                                             va1,
                                             fa1,
                                             val1,
                                             graph-trans(ev-graph(flg,
                                                                  exp,
                                                                  va,
                                                                  fa,
                                                                  n))),
                             a-ev-inst-formals-proof(flg,
                                                     exp,
                                                     va,
                                                     fa,
                                                     n),
                             tautconseq-proof(list(f-not(isin(z-list(list(val1,
                                                                          flg1,
                                                                          exp1,
                                                                          va1,
                                                                          fa1)),
                                                              graph-trans(nil)))),
                                                   f-implies(isin(z-list(list(val1,
                                                                              flg1,
```

```
                                                                      exp1,
                                                                      va1,
                                                                      fa1)),
                                                                graph-trans (nil)),
                                                             a-ev-defn-part1 (flg1,
                                                                              exp1,
                                                                              va1,
                                                                              fa1,
                                                                              val1,
                                                                              graph-trans (ev-graph (flg,
                                                                                                     exp,
                                                                                                     va,
                                                                                                     fa,
                                                                                                     n)))),
                                                       list (null-axiom-proof (z-list (list (val1,
                                                                                             flg1,
                                                                                             exp1,
                                                                                             va1,
                                                                                             fa1)))))) endif
elseif listp (exp)
then if car (exp) ≃ 0
     then if ncode-equal (ev (0, cadr (exp), va, fa, n), 0)
          then a-ev-defn-ok-frame-proof2 (z-list (list (val1,
                                                        flg1,
                                                        exp1,
                                                        va1,
                                                        fa1)),
                                          cons (ev (flg, exp, va, fa, n),
                                                cons (flg,
                                                      cons (exp,
                                                            cons (va, fa)))),
                                          ev-graph (0,
                                                    cadr (exp),
                                                    va,
                                                    fa,
                                                    n),
                                          ev-graph (0,
                                                    cadddr (exp),
                                                    va,
                                                    fa,
                                                    n),
                                          a-ev-defn-part1 (flg1,
                                                           exp1,
                                                           va1,
```

```
                    fa1,
                    val1,
                    graph-trans (ev-graph (flg,
                                           exp,
                                           va,
                                           fa,
                                           n))),
a-ev-symb-form (flg1,
                exp1,
                va1,
                fa1,
                val1,
                graph-trans (ev-graph (flg,
                                       exp,
                                       va,
                                       fa,
                                       n))),
a-ev-defn-part1 (flg1,
                 exp1,
                 va1,
                 fa1,
                 val1,
                 graph-trans (ev-graph (0,
                                        cadr (exp),
                                        va,
                                        fa,
                                        n))),
a-ev-symb-form (flg1,
                exp1,
                va1,
                fa1,
                val1,
                graph-trans (ev-graph (0,
                                       cadr (exp),
                                       va,
                                       fa,
                                       n))),
a-ev-defn-part1 (flg1,
                 exp1,
                 va1,
                 fa1,
                 val1,
                 graph-trans (ev-graph (0,
                                        cadddr (exp),
```

```
                                                   va,
                                                   fa,
                                                   n))),
                   a-ev-symb-form (flg1,
                                   exp1,
                                   va1,
                                   fa1,
                                   val1,
                                   graph-trans (ev-graph (0,
                                                   cadddr (exp),
                                                   va,
                                                   fa,
                                                   n))),
                   a-ev-inst-formals-proof (flg,
                                   exp,
                                   va,
                                   fa,
                                   n),
                   a-ev-defn-ok-proof (0,
                                   cadr (exp),
                                   va,
                                   fa,
                                   n,
                                   flg1,
                                   exp1,
                                   va1,
                                   fa1,
                                   val1),
                   a-ev-defn-ok-proof (0,
                                   cadddr (exp),
                                   va,
                                   fa,
                                   n,
                                   flg1,
                                   exp1,
                                   va1,
                                   fa1,
                                   val1))
else a-ev-defn-ok-frame-proof2 (z-list (list (val1,
                                   flg1,
                                   exp1,
                                   va1,
                                   fa1)),
                   cons (ev (flg,
```

```
              exp,
              va,
              fa,
              n),
         cons (flg,
                cons (exp,
                       cons (va,
                              fa)))),
ev-graph (0,
              cadr (exp),
              va,
              fa,
              n),
ev-graph (0,
              caddr (exp),
              va,
              fa,
              n),
a-ev-defn-part1 (flg1,
                        exp1,
                        va1,
                        fa1,
                        val1,
                        graph-trans (ev-graph (flg,
                                                           exp,
                                                           va,
                                                           fa,
                                                           n))),
a-ev-symb-form (flg1,
                        exp1,
                        va1,
                        fa1,
                        val1,
                        graph-trans (ev-graph (flg,
                                                           exp,
                                                           va,
                                                           fa,
                                                           n))),
a-ev-defn-part1 (flg1,
                        exp1,
                        va1,
                        fa1,
                        val1,
                        graph-trans (ev-graph (0,
```

```
                                                  cadr (exp),
                                                  va,
                                                  fa,
                                                  n))),
a-ev-symb-form (flg1,
                exp1,
                va1,
                fa1,
                val1,
                graph-trans (ev-graph (0,
                                       cadr (exp),
                                       va,
                                       fa,
                                       n))),
a-ev-defn-part1 (flg1,
                 exp1,
                 va1,
                 fa1,
                 val1,
                 graph-trans (ev-graph (0,
                                        caddr (exp),
                                        va,
                                        fa,
                                        n))),
a-ev-symb-form (flg1,
                exp1,
                va1,
                fa1,
                val1,
                graph-trans (ev-graph (0,
                                       caddr (exp),
                                       va,
                                       fa,
                                       n))),
a-ev-inst-formals-proof (flg,
                         exp,
                         va,
                         fa,
                         n),
a-ev-defn-ok-proof (0,
                    cadr (exp),
                    va,
                    fa,
                    n,
```

```
                                              flg1,
                                              exp1,
                                              va1,
                                              fa1,
                                              val1),
                         a-ev-defn-ok-proof(0,
                                              caddr (exp),
                                              va,
                                              fa,
                                              n,
                                              flg1,
                                              exp1,
                                              va1,
                                              fa1,
                                              val1)) endif
elseif car (exp) = 10
then graph-trans-merge-proof(cons (ev (flg, exp, va, fa, n),
                                   cons (flg, cons (exp, cons (va, fa)))),
                             nil,
                             z-list (list (val1, flg1, exp1, va1, fa1)),
                             a-ev-defn-part1 (flg1,
                                              exp1,
                                              va1,
                                              fa1,
                                              val1,
                                              graph-trans (ev-graph (flg,
                                                                     exp,
                                                                     va,
                                                                     fa,
                                                                     n))),
                             a-ev-inst-formals-proof(flg,
                                                     exp,
                                                     va,
                                                     fa,
                                                     n),
                             tautconseq-proof (list (f-not (isin (z-list (list (val1,
                                                                               flg1,
                                                                               exp1,
                                                                               va1,
                                                                               fa1)),
                                                                  graph-trans (nil)))),
                                                     f-implies (isin (z-list (list (val1,
                                                                                    flg1,
                                                                                    exp1,
```

```
                                                    va1,
                                                    fa1)),
                                               graph-trans (nil)),
                                          a-ev-defn-part1 (flg1,
                                                           exp1,
                                                           va1,
                                                           fa1,
                                                           val1,
                                                           graph-trans (ev-graph (flg,
                                                                                  exp,
                                                                                  va,
                                                                                  fa,
                                                                                  n)))),
                                     list (null-axiom-proof (z-list (list (val1,
                                                                           flg1,
                                                                           exp1,
                                                                           va1,
                                                                           fa1))))))
elseif subrp (car (exp))
then a-ev-defn-ok-frame-proof3 (z-list (list (val1,
                                              flg1,
                                              exp1,
                                              va1,
                                              fa1)),
                                cons (ev (flg, exp, va, fa, n),
                                      cons (flg,
                                            cons (exp, cons (va, fa)))),
                                ev-graph (1, cdr (exp), va, fa, n),
                                a-ev-defn-part1 (flg1,
                                                 exp1,
                                                 va1,
                                                 fa1,
                                                 val1,
                                                 graph-trans (ev-graph (flg,
                                                                        exp,
                                                                        va,
                                                                        fa,
                                                                        n))),
                                a-ev-symb-form (flg1,
                                                exp1,
                                                va1,
                                                fa1,
                                                val1,
                                                graph-trans (ev-graph (flg,
```

```
                                                           exp,
                                                           va,
                                                           fa,
                                                           n))),
                                    a-ev-defn-part1 (flg1,
                                                     exp1,
                                                     va1,
                                                     fa1,
                                                     val1,
                                                     graph-trans (ev-graph (1,
                                                                          cdr (exp),
                                                                          va,
                                                                          fa,
                                                                          n))),
                                    a-ev-symb-form (flg1,
                                                    exp1,
                                                    va1,
                                                    fa1,
                                                    val1,
                                                    graph-trans (ev-graph (1,
                                                                         cdr (exp),
                                                                         va,
                                                                         fa,
                                                                         n))),
                                    a-ev-inst-formals-proof (flg,
                                                             exp,
                                                             va,
                                                             fa,
                                                             n),
                                    a-ev-defn-ok-proof (1,
                                                        cdr (exp),
                                                        va,
                                                        fa,
                                                        n,
                                                        flg1,
                                                        exp1,
                                                        va1,
                                                        fa1,
                                                        val1))
elseif n ≃ 0 then nil
else a-ev-defn-ok-frame-proof2 (z-list (list (val1,
                                              flg1,
                                              exp1,
                                              va1,
```

```
            fa1)),
cons (ev (flg, exp, va, fa, n),
      cons (flg,
            cons (exp,
                  cons (va, fa)))),
ev-graph (0,
          get (sub1-n (car (exp),
                       11),
               fa),
          ev (1,
              cdr (exp),
              va,
              fa,
              n),
          fa,
          n − 1),
ev-graph (1,
          cdr (exp),
          va,
          fa,
          n),
a-ev-defn-part1 (flg1,
                 exp1,
                 va1,
                 fa1,
                 val1,
                 graph-trans (ev-graph (flg,
                                        exp,
                                        va,
                                        fa,
                                        n))),
a-ev-symb-form (flg1,
                exp1,
                va1,
                fa1,
                val1,
                graph-trans (ev-graph (flg,
                                       exp,
                                       va,
                                       fa,
                                       n))),
a-ev-defn-part1 (flg1,
                 exp1,
                 va1,
```

```
                fa1,
                val1,
                graph-trans (ev-graph (0,
                                      get (sub1-n (car (exp),
                                                   11),
                                           fa),
                                      ev (1,
                                          cdr (exp),
                                          va,
                                          fa,
                                          n),
                                      fa,
                                      n − 1))),
a-ev-symb-form (flg1,
                exp1,
                va1,
                fa1,
                val1,
                graph-trans (ev-graph (0,
                                      get (sub1-n (car (exp),
                                                   11),
                                           fa),
                                      ev (1,
                                          cdr (exp),
                                          va,
                                          fa,
                                          n),
                                      fa,
                                      n − 1))),
a-ev-defn-part1 (flg1,
                exp1,
                va1,
                fa1,
                val1,
                graph-trans (ev-graph (1,
                                      cdr (exp),
                                      va,
                                      fa,
                                      n))),
a-ev-symb-form (flg1,
                exp1,
                va1,
                fa1,
                val1,
```

```
                              graph-trans (ev-graph (1,
                                                     cdr (exp),
                                                     va,
                                                     fa,
                                                     n))),
                   a-ev-inst-formals-proof (flg,
                                            exp,
                                            va,
                                            fa,
                                            n),
                   a-ev-defn-ok-proof (0,
                                       get (sub1-n (car (exp),
                                                    11),
                                            fa),
                                       ev (1,
                                           cdr (exp),
                                           va,
                                           fa,
                                           n),
                                       fa,
                                       n − 1,
                                       flg1,
                                       exp1,
                                       va1,
                                       fa1,
                                       val1),
                   a-ev-defn-ok-proof (1,
                                       cdr (exp),
                                       va,
                                       fa,
                                       n,
                                       flg1,
                                       exp1,
                                       va1,
                                       fa1,
                                       val1)) endif
else graph-trans-merge-proof (cons (ev (flg, exp, va, fa, n),
                                    cons (flg,
                                          cons (exp, cons (va, fa)))),
                              nil,
                              z-list (list (val1, flg1, exp1, va1, fa1)),
                              a-ev-defn-part1 (flg1,
                                               exp1,
                                               va1,
```

```
                                          fa1,
                                          val1,
                                          graph-trans (ev-graph (flg,
                                                                 exp,
                                                                 va,
                                                                 fa,
                                                                 n))),
                                  a-ev-inst-formals-proof (flg,
                                                           exp,
                                                           va,
                                                           fa,
                                                           n),
                                  tautconseq-proof (list (f-not (isin (z-list (list (val1,
                                                                                     flg1,
                                                                                     exp1,
                                                                                     va1,
                                                                                     fa1)),
                                                                        graph-trans (nil)))),
                                           f-implies (isin (z-list (list (val1,
                                                                          flg1,
                                                                          exp1,
                                                                          va1,
                                                                          fa1)),
                                                            graph-trans (nil)),
                                                      a-ev-defn-part1 (flg1,
                                                                       exp1,
                                                                       va1,
                                                                       fa1,
                                                                       val1,
                                                                       graph-trans (ev-graph (flg,
                                                                                              exp,
                                                                                              va,
                                                                                              fa,
                                                                                              n)))),
                                           list (null-axiom-proof (z-list (list (val1,
                                                                                 flg1,
                                                                                 exp1,
                                                                                 va1,
                                                                                 fa1)))))) endif
```

`;;;;;;;The definition part of "A-ev" is ‘OK’.`

THEOREM: a-ev-defn-ok-proof-proves
(subset (list (fn (16, 1),

fn (15, 2),
fn (14, 2),
fn (13, 2),
fn (12, 2),
fn (0, 0),
fn (3, 1),
fn (4, 1),
fn (5, 1),
fn (6, 1),
fn (7, 1),
fn (8, 1),
fn (9, 1),
fn (10, 1),
fn (11, 2),
fn (1, 2),
fn (2, 2),
p (2, 1),
p (1, 2)),
symbols)
$\wedge$ subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), *symbols*)
$\wedge$ subset (list (p (1, 2),
fn (0, 0),
fn (1, 2),
fn (2, 2),
fn (3, 1),
fn (6, 1),
fn (10, 1),
fn (16, 1),
fn (15, 2),
fn (9, 1),
fn (4, 1),
fn (5, 1)),
symbols)
$\wedge$ subset (list (p (1, 2), fn (1, 2)), *symbols*)
$\wedge$ (fn (0, 0) $\in$ *symbols*)
$\wedge$ (fn (1, 2) $\in$ *symbols*)
$\wedge$ (p (1, 2) $\in$ *symbols*)
$\wedge$ subset (list (fn (0, 0), p (1, 2)), *symbols*)
$\wedge$ z-apply-subr-all-hyps (*given*, *defns*, *symbols*)
$\wedge$ z-subrp-hyps (*defns*, *symbols*)
$\wedge$ ($\neg$ btmp (ev (*flg*, *exp*, *va*, *fa*, *n*))))
$\rightarrow$ proves (a-ev-defn-ok-proof (*flg*, *exp*, *va*, *fa*, *n*, 6, 7, 8, 9, 10),
f-implies (isin (z-list (list (10, 6, 7, 8, 9)),
graph-trans (ev-graph (*flg*, *exp*, *va*, *fa*, *n*))),

```
                    a-ev-defn-part1 (6,
                                     7,
                                     8,
                                     9,
                                     10,
                                     graph-trans (ev-graph (flg, exp, va, fa, n)))),
           given,
           defns,
           symbols)
```

EVENT: Enable a-ev-defn-ok-proof; name this event ‘g0896’.

EVENT: Enable ev-graph; name this event ‘g0897’.

DEFINITION:

```
a-ev (flg, exp, va, fa, val, trace)
=   f-and (isin (z-list (list (val, flg, exp, va, fa)), trace),
           list-all-quantify (list (6, 7, 8, 9, 10),
                              f-implies (isin (z-list (list (10, 6, 7, 8, 9)),
                                               trace),
                                         a-ev-defn-part1 (6, 7, 8, 9, 10, trace))))
```

DEFINITION:

```
a-ev-ok-proof (flg, exp, va, fa, n)
=   and-join-proof (isin (z-list (list (ncode (ev (flg, exp, va, fa, n)),
                                        ncode (flg),
                                        ncode (exp),
                                        ncode (va),
                                        ncode (fa))),
                          graph-trans (ev-graph (flg, exp, va, fa, n))),
                    list-all-quantify (list (6, 7, 8, 9, 10),
                                       f-implies (isin (z-list (list (10,
                                                                      6,
                                                                      7,
                                                                      8,
                                                                      9)),
                                                        graph-trans (ev-graph (flg,
                                                                               exp,
                                                                               va,
                                                                               fa,
                                                                               n))),
                                                  a-ev-defn-part1 (6,
                                                                   7,
```

```
                                                          8,
                                                          9,
                                                          10,
                                                          graph-trans (ev-graph (flg,
                                                                                 exp,
                                                                                 va,
                                                                                 fa,
                                                                                 n))))),
                    in-graph-proof (cons (ev (flg, exp, va, fa, n),
                                          cons (flg, cons (exp, cons (va, fa)))),
                                    ev-graph (flg, exp, va, fa, n)),
                    list-gen-proof (list (6, 7, 8, 9, 10),
                                    f-implies (isin (z-list (list (10, 6, 7, 8, 9)),
                                                     graph-trans (ev-graph (flg,
                                                                            exp,
                                                                            va,
                                                                            fa,
                                                                            n))),
                                    a-ev-defn-part1 (6,
                                                     7,
                                                     8,
                                                     9,
                                                     10,
                                                     graph-trans (ev-graph (flg,
                                                                            exp,
                                                                            va,
                                                                            fa,
                                                                            n)))),
                                    a-ev-defn-ok-proof (flg,
                                                        exp,
                                                        va,
                                                        fa,
                                                        n,
                                                        6,
                                                        7,
                                                        8,
                                                        9,
                                                        10)))
```

THEOREM: a-ev-defn-ok-proof-proves-again

```
 (ev-hyps (given, defns, symbols)
  ∧   (¬ btmp (ev (flg, exp, va, fa, n)))
  ∧   (concl = f-implies (isin (z-list (list (10, 6, 7, 8, 9)),
                                graph-trans (ev-graph (flg, exp, va, fa, n))),
```

```
                    a-ev-defn-part1 (6,
                                     7,
                                     8,
                                     9,
                                     10,
                                     graph-trans (ev-graph (flg,
                                                            exp,
                                                            va,
                                                            fa,
                                                            n))))))
 →   proves (a-ev-defn-ok-proof (flg, exp, va, fa, n, 6, 7, 8, 9, 10),
             concl,
             given,
             defns,
             symbols)
```

EVENT: Enable a-ev-defn-ok-proof-proves; name this event ‘g0898’.

```
;;;;;;;"A-ev" is ‘OK’, i.e.,
;;;;;;;|- "A-ev(<FLG>, <EXP>, <VA>, <FA>, <(EV FLG EXP VA FA N)>,
;;;;;;;         [(EV-GRAPH FLG EXP VA FA N)])"
```

THEOREM: a-ev-ok-proof-proves

```
 (ev-hyps (given, defns, symbols)
  ∧   (¬ btmp (ev (flg, exp, va, fa, n)))
  ∧   (concl = a-ev (ncode (flg),
                     ncode (exp),
                     ncode (va),
                     ncode (fa),
                     ncode (ev (flg, exp, va, fa, n)),
                     graph-trans (ev-graph (flg, exp, va, fa, n)))))
 →   proves (a-ev-ok-proof (flg, exp, va, fa, n), concl, given, defns, symbols)
```

EVENT: Enable a-ev-ok-proof; name this event ‘g0899’.

THEOREM: forsome-intro-proves-again

```
 (variable (var)
  ∧   (var ∉ collect-free (b, 0))
  ∧   proves (pf, f-implies (a, b), given, defns, symbols))
 →   proves (forsome-intro-proof (var, a, b, pf),
             f-implies (forsome (var, a), b),
             given,
             defns,
             symbols)
```

DEFINITION:
a-ev-good-reduc-proof(*flg*, *exp*, *expr*, *va*, *fa*, *val1*, *trace*, *conc*, *pfexpr*, *pf*)

```
=   chain-proof (isin (z-list (list (val1, ncode (flg), expr, ncode (va), ncode (fa))),
                      trace),
                conc,
                list (isin (z-list (list (val1,
                                         ncode (flg),
                                         ncode (exp),
                                         ncode (va),
                                         ncode (fa))),
                           trace)),
                list (equal-isin-imp-proof (z-list (list (val1,
                                                          ncode (flg),
                                                          expr,
                                                          ncode (va),
                                                          ncode (fa))),
                                            z-list (list (val1,
                                                          ncode (flg),
                                                          ncode (exp),
                                                          ncode (va),
                                                          ncode (fa))),
                                            trace,
                                            flip-equals-proof (z-list (list (val1,
                                                                             ncode (flg),
                                                                             ncode (exp),
                                                                             ncode (va),
                                                                             ncode (fa))),
                                                               z-list (list (val1,
                                                                             ncode (flg),
                                                                             expr,
                                                                             ncode (va),
                                                                             ncode (fa))),
                                                               ev-equal-z-list-proof (val1,
                                                                                      flg,
                                                                                      exp,
                                                                                      expr,
                                                                                      va,
                                                                                      fa,
                                                                                      pfexpr))),
                pf))
```

THEOREM: a-ev-good-reduc-proof-proves
(subset (list (p (1, 2), fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), *symbols*)
$\wedge$ termp (list (*val1*, *trace*), 1, *symbols*)

$\land$ proves (*pfexpr*, f-equal (*expr*, ncode (*exp*)), *given*, *defns*, *symbols*)
$\land$ proves (*pf*,
f-implies (isin (z-list (list (*val1*,
ncode (*flg*),
ncode (*exp*),
ncode (*va*),
ncode (*fa*))),
trace),
conc),
given,
defns,
symbols)
$\land$ (*concl* = f-implies (isin (z-list (list (*val1*,
ncode (*flg*),
expr,
ncode (*va*),
ncode (*fa*))),
trace),
conc)))
$\rightarrow$ proves (a-ev-good-reduc-proof (*flg*,
exp,
expr,
va,
fa,
val1,
trace,
conc,
pfexpr,
pf),
concl,
given,
defns,
symbols)

EVENT: Enable a-ev-good-reduc-proof; name this event 'g0900'.

DEFINITION:
a-ev-defn-good-step-proof (*cl1*, *cl2*, *dexp*, *rvar*, *rval*, *val1*, *val-bar*, *pf1*, *pf2*)
= tautconseq-proof (list (f-implies (*cl1*,
f-implies (*dexp*,
f-equal (*rvar*, ncode (*rval*)))),
f-implies (f-equal (*rvar*, ncode (*rval*)),
f-implies (*cl2*,
f-implies (*dexp*,

```
                                          f-equal (val1,
                                                   val-bar))))),
                  f-implies (f-and (cl1, cl2),
                             f-implies (dexp, f-equal (val1, val-bar))),
                  list (pf1,
                        form-equal-case-proof (list (rvar),
                                               list (ncode (rval)),
                                               f-implies (cl2,
                                                          f-implies (dexp,
                                                                     f-equal (val1,
                                                                              val-bar))),
                                               pf2)))
```

THEOREM: a-ev-defn-good-step-proof-proves

```
 (variable (rvar)
  ∧  formula (cl2, symbols)
  ∧  termp (list (val1, val-bar), 1, symbols)
  ∧  proves (pf1,
             f-implies (cl1, f-implies (dexp, f-equal (rvar, ncode (rval)))),
             given,
             defns,
             symbols)
  ∧  proves (pf2,
             parallel-subst (f-implies (cl2,
                                        f-implies (dexp, f-equal (val1, val-bar))),
                             list (rvar),
                             list (ncode (rval)),
                             0),
             given,
             defns,
             symbols))
 →   proves (a-ev-defn-good-step-proof (cl1,
                                        cl2,
                                        dexp,
                                        rvar,
                                        rval,
                                        val1,
                                        val-bar,
                                        pf1,
                                        pf2),
             f-implies (f-and (cl1, cl2), f-implies (dexp, f-equal (val1, val-bar))),
             given,
             defns,
             symbols)
```

EVENT: Enable a-ev-defn-good-step-proof; name this event 'g0901'.

DEFINITION:

```
a-ev-defn-part-list-good-proof(exp,
                               va,
                               fa,
                               val1,
                               rvar1,
                               rvar2,
                               rval1,
                               rval2,
                               trace,
                               dexp,
                               pf1,
                               pf2)
=   forsome-intro-proof(rvar1,
                        forsome (rvar2,
                                 f-and (isin (z-list (list (rvar1,
                                                            ncode (0),
                                                            z-car (ncode (exp)),
                                                            ncode (va),
                                                            ncode (fa))),
                                              trace),
                                        f-and (isin (z-list (list (rvar2,
                                                                   ncode (1),
                                                                   z-cdr (ncode (exp)),
                                                                   ncode (va),
                                                                   ncode (fa))),
                                                     trace),
                                               f-equal (val1,
                                                        z-opair (rvar1, rvar2))))),
                        f-implies (dexp,
                                   f-equal (val1, ncode (cons (rval1, rval2)))),
                        forsome-intro-proof(rvar2,
                                            f-and (isin (z-list (list (rvar1,
                                                                       ncode (0),
                                                                       z-car (ncode (exp)),
                                                                       ncode (va),
                                                                       ncode (fa))),
                                                         trace),
                                                   f-and (isin (z-list (list (rvar2,
                                                                              ncode (1),
                                                                              z-cdr (ncode (exp)),
```

```
                                   ncode (va),
                                   ncode (fa))),
                        trace),
                   f-equal (val1,
                           z-opair (rvar1,
                                    rvar2)))),
f-implies (dexp,
           f-equal (val1,
                   ncode (cons (rval1,
                                rval2)))),
a-ev-defn-good-step-proof (isin (z-list (list (rvar1,
                                                ncode (0),
                                                z-car (ncode (exp)),
                                                ncode (va),
                                                ncode (fa))),
                                     trace),
                           f-and (isin (z-list (list (rvar2,
                                                      ncode (1),
                                                      z-cdr (ncode (exp)),
                                                      ncode (va),
                                                      ncode (fa))),
                                           trace),
                                 f-equal (val1,
                                         z-opair (rvar1,
                                                  rvar2))),
                           dexp,
                           rvar1,
                           rval1,
                           val1,
                           ncode (cons (rval1,
                                        rval2)),
                           a-ev-good-reduc-proof (0,
                                                  car (exp),
                                                  z-car (ncode (exp)),
                                                  va,
                                                  fa,
                                                  rvar1,
                                                  trace,
                                                  f-implies (dexp,
                                                             f-equal (rvar1,
                                                                     ncode (rval1))),
                                                  z-car-ncode-proof (exp),
                                                  pf1),
                           a-ev-defn-good-step-proof (isin (z-list (list (rvar2,
```

```
                    ncode (1),
                    z-cdr (ncode (exp)),
                    ncode (va),
                    ncode (fa))),
      trace),
f-equal (val1,
          z-opair (ncode (rval1),
                   rvar2)),
dexp,
rvar2,
rval2,
val1,
ncode (cons (rval1,
             rval2)),
a-ev-good-reduc-proof (1,
                       cdr (exp),
                       z-cdr (ncode (exp)),
                       va,
                       fa,
                       rvar2,
                       trace,
                       f-implies (dexp,
                                  f-equal (rvar2,
                                           ncode (rval2))),
                       z-cdr-ncode-proof (exp),
                       pf2),
tautconseq-proof (nil,
                  f-implies (f-equal (val1,
                                      ncode (cons (rval1,
                                                   rval2))),
                             f-implies (dexp,
                                        f-equal (val1,
                                                 ncode (cons (rval1,
                                                              rval2))))),
                  nil)))))
```

EVENT: Disable not-free-parallel-subst; name this event 'g0908'.

THEOREM: a-ev-defn-part-list-good-proof-proves

```
 (subset (list (fn (5, 1),
                fn (4, 1),
                fn (3, 1),
                fn (2, 2),
                fn (1, 2),
```

```
              p (1, 2),
              p (2, 1),
              fn (0, 0)),
          symbols)
∧   z-succ-hyps (defns, symbols)
∧   z-car-hyps (defns, symbols)
∧   z-cdr-hyps (defns, symbols)
∧   z-int-hyps (defns, symbols)
∧   var-set (list (rvar1, rvar2, val1, trace), 4)
∧   nil-intersect (list (rvar1, rvar2), collect-free (dexp, 0))
∧   proves (pf1,
              f-implies (isin (z-list (list (rvar1,
                                              ncode (0),
                                              ncode (car (exp)),
                                              ncode (va),
                                              ncode (fa))),
                                  trace),
                           f-implies (dexp, f-equal (rvar1, ncode (rval1)))),
              given,
              defns,
              symbols)
∧   proves (pf2,
              f-implies (isin (z-list (list (rvar2,
                                              ncode (1),
                                              ncode (cdr (exp)),
                                              ncode (va),
                                              ncode (fa))),
                                  trace),
                           f-implies (dexp, f-equal (rvar2, ncode (rval2)))),
              given,
              defns,
              symbols)
∧   (concl = f-implies (a-ev-list-part (exp, va, fa, val1, rvar1, rvar2, trace),
                          f-implies (dexp,
                                     f-equal (val1, ncode (cons (rval1, rval2)))))))
→   proves (a-ev-defn-part-list-good-proof (exp,
                                              va,
                                              fa,
                                              val1,
                                              rvar1,
                                              rvar2,
                                              rval1,
                                              rval2,
                                              trace,
```

```
                                   dexp,
                                   pf1,
                                   pf2),
              concl,
              given,
              defns,
              symbols)
```

EVENT: Enable a-ev-defn-part-list-good-proof; name this event 'g0916'.

DEFINITION:

```
a-ev-defn-part-if-good-proof(exp,
                             va,
                             fa,
                             val1,
                             val,
                             trace,
                             rvar1,
                             rval1,
                             dexp,
                             pf1,
                             pf)
=   forsome-intro-proof(rvar1,
                        body (a-ev-if-part (ncode (exp),
                                            ncode (va),
                                            ncode (fa),
                                            val1,
                                            trace,
                                            rvar1)),
                        f-implies (dexp, f-equal (val1, ncode (val))),
                        a-ev-defn-good-step-proof(isin (z-list (list (rvar1,
                                                                      ncode (0),
                                                                      z-car (z-cdr (ncode (exp))),
                                                                      ncode (va),
                                                                      ncode (fa))),
                                                            trace),
                                                  f-if (f-equal (rvar1,
                                                                 ncode (0)),
                                                        isin (z-list (list (val1,
                                                                            ncode (0),
                                                                            z-car-cdr (ncode (exp),
                                                                                       list (1,
                                                                                             2,
                                                                                             2,
```

```
                                            2)),
                              ncode (va),
                              ncode (fa))),
                    trace),
          isin (z-list (list (val1,
                              ncode (0),
                              z-car-cdr (ncode (exp),
                                         list (1,
                                               2,
                                               2)),
                              ncode (va),
                              ncode (fa))),
                    trace)),
dexp,
rvar1,
rval1,
val1,
ncode (val),
a-ev-good-reduc-proof (0,
                        cadr (exp),
                        z-car (z-cdr (ncode (exp))),
                        va,
                        fa,
                        rvar1,
                        trace,
                        f-implies (dexp,
                                   f-equal (rvar1,
                                            ncode (rval1))),
                        z-car-cdr-proof (exp,
                                         list (1,
                                               2)),
                        pf1),
rec-form-z-if-eval-proof (list (list (ncode-equal (rval1,
                                                   0),
                                      ident-axiom-proof (ncode (0)),
                                      ncode-nequal-proof (rval1,
                                                          0)),
                                a-ev-good-reduc-proof (0,
                                                       cadddr (exp),
                                                       z-car-cdr (ncode (exp),
                                                                  list (1,
                                                                        2,
                                                                        2,
                                                                        2)),
```

```
                                                va,
                                                fa,
                                                val1,
                                                trace,
                                                f-implies (dexp,
                                                           f-equal (val1,
                                                                    ncode (val))),
                                                z-car-cdr-proof (exp,
                                                                 list (1,
                                                                       2,
                                                                       2,
                                                                       2)),
                                                pf),
                        a-ev-good-reduc-proof (0,
                                                caddr (exp),
                                                z-car-cdr (ncode (exp),
                                                           list (1,
                                                                 2,
                                                                 2)),
                                                va,
                                                fa,
                                                val1,
                                                trace,
                                                f-implies (dexp,
                                                           f-equal (val1,
                                                                    ncode (val))),
                                                z-car-cdr-proof (exp,
                                                                 list (1,
                                                                       2,
                                                                       2)),
                                                pf)),
                   z-if (f-equal (ncode (rval1),
                                  ncode (0)),
                        isin (z-list (list (val1,
                                            ncode (0),
                                            z-car-cdr (ncode (exp),
                                                       list (1,
                                                             2,
                                                             2,
                                                             2)),
                                            ncode (va),
                                            ncode (fa))),
                              trace),
                        isin (z-list (list (val1,
```

```
                                                ncode (0),
                                                z-car-cdr (ncode (exp),
                                                           list (1,
                                                                 2,
                                                                 2)),
                                                ncode (va),
                                                ncode (fa))),
                                      trace)),
                             f-implies (dexp,
                                        f-equal (val1,
                                                 ncode (val))))))
```

THEOREM: nlistp-nnumberp-ncode
$((\neg \text{listp}(x)) \land (x \notin \mathbf{N})) \rightarrow (\text{ncode}(x) = \text{ncode}(0))$

THEOREM: a-ev-defn-part-if-good-proof-proves

```
(subset (list (fn (5, 1),
               fn (4, 1),
               fn (3, 1),
               fn (2, 2),
               fn (1, 2),
               p (1, 2),
               p (2, 1),
               fn (0, 0)),
          symbols)
 ∧   z-succ-hyps (defns, symbols)
 ∧   z-car-hyps (defns, symbols)
 ∧   z-cdr-hyps (defns, symbols)
 ∧   z-int-hyps (defns, symbols)
 ∧   var-set (list (rvar1, val1, trace), 3)
 ∧   (rvar1 ∉ collect-free (dexp, 0))
 ∧   proves (pf1,
             f-implies (isin (z-list (list (rvar1,
                                            ncode (0),
                                            ncode (cadr (exp)),
                                            ncode (va),
                                            ncode (fa))),
                                   trace),
                        f-implies (dexp, f-equal (rvar1, ncode (rval1)))),
             given,
             defns,
             symbols)
 ∧   if ncode-equal (rval1, 0)
     then proves (pf,
```

```
                    f-implies (isin (z-list (list (val1,
                                                   ncode (0),
                                                   ncode (cadddr (exp)),
                                                   ncode (va),
                                                   ncode (fa))),
                                      trace),
                                 f-implies (dexp, f-equal (val1, ncode (val)))),
                    given,
                    defns,
                    symbols)
      else proves (pf,
                   f-implies (isin (z-list (list (val1,
                                                  ncode (0),
                                                  ncode (caddr (exp)),
                                                  ncode (va),
                                                  ncode (fa))),
                                     trace),
                                f-implies (dexp, f-equal (val1, ncode (val)))),
                   given,
                   defns,
                   symbols) endif
∧     (concl = f-implies (a-ev-if-part (ncode (exp),
                                        ncode (va),
                                        ncode (fa),
                                        val1,
                                        trace,
                                        rvar1),
                          f-implies (dexp, f-equal (val1, ncode (val))))))
→     proves (a-ev-defn-part-if-good-proof (exp,
                                            va,
                                            fa,
                                            val1,
                                            val,
                                            trace,
                                            rvar1,
                                            rval1,
                                            dexp,
                                            pf1,
                                            pf),
              concl,
              given,
              defns,
              symbols)
```

EVENT: Enable a-ev-defn-part-if-good-proof; name this event 'g0919'.

EVENT: Enable nlistp-nnumberp-ncode; name this event 'g0920'.

DEFINITION:
a-ev-defn-part-subr-good-proof(*exp*, *va*, *fa*, *val1*, *trace*, *rvar1*, *rval1*, *dexp*, *pf*)
= forsome-intro-proof(*rvar1*,
        body (a-ev-subr-part (*exp*, *va*, *fa*, *val1*, *rvar1*, *trace*)),
        f-implies (*dexp*,
                f-equal (*val1*,
                        ncode (apply-subr (car (*exp*), *rval1*)))),
        a-ev-defn-good-step-proof (isin (z-list (list (*rvar1*,
                                                ncode (1),
                                                z-cdr (ncode (*exp*)),
                                                ncode (*va*),
                                                ncode (*fa*))),
                        trace),
                f-equal (*val1*,
                        z-apply-subr (z-car (ncode (*exp*)),
                                rvar1)),
                dexp,
                rvar1,
                rval1,
                val1,
                ncode (apply-subr (car (*exp*),
                                rval1)),
                a-ev-good-reduc-proof (1,
                        cdr (*exp*),
                        z-cdr (ncode (*exp*)),
                        va,
                        fa,
                        rvar1,
                        trace,
                        f-implies (*dexp*,
                                f-equal (*rvar1*,
                                        ncode (*rval1*))),
                        z-cdr-ncode-proof (*exp*),
                        pf),
                tautconseq-proof (list (f-equal (z-apply-subr (z-car (ncode (*exp*)),
                                                        ncode (*rval1*)),
                                        ncode (apply-subr (car (*exp*),
                                                        rval1))),
                        f-implies (f-equal (*val1*,

```
                              z-apply-subr (z-car (ncode (exp)),
                                            ncode (rval1))),
                    f-implies (f-equal (z-apply-subr (z-car (ncode (exp)),
                                                      ncode (rval1)),
                                        ncode (apply-subr (car (exp),
                                                           rval1))),
                               f-equal (val1,
                                        ncode (apply-subr (car (exp),
                                                           rval1)))))),
f-implies (f-equal (val1,
                    z-apply-subr (z-car (ncode (exp)),
                                  ncode (rval1))),
           f-implies (dexp,
                      f-equal (val1,
                               ncode (apply-subr (car (exp),
                                                  rval1))))),
list (compose-proof (fn (15,
                         2),
                     ncode (apply-subr (car (exp),
                                        rval1)),
                     list (z-car (ncode (exp)),
                           ncode (rval1)),
                     list (ncode (car (exp)),
                           ncode (rval1)),
                     list (z-car-ncode-proof (exp),
                           ident-axiom-proof (ncode (rval1))),
                     z-apply-subr-ncode-proof (car (exp),
                                               rval1)),
      trans-equal-proof3 (val1,
                          z-apply-subr (z-car (ncode (exp)),
                                        ncode (rval1)),
                          ncode (apply-subr (car (exp),
                                             rval1)))))))
```

THEOREM: a-ev-defn-part-subr-good-proof-proves

```
(subset (list (fn (15, 2),
               fn (14, 2),
               fn (13, 2),
               fn (12, 2),
               fn (0, 0),
               fn (3, 1),
               fn (4, 1),
               fn (5, 1),
               fn (6, 1),
```

```
                fn (7, 1),
                fn (8, 1),
                fn (9, 1),
                fn (10, 1),
                fn (11, 2),
                fn (1, 2),
                fn (2, 2),
                p (2, 1),
                p (1, 2)),
            symbols)
∧   z-apply-subr-all-hyps (given, defns, symbols)
∧   var-set (list (rvar1, val1, trace), 3)
∧   (rvar1 ∉ collect-free (dexp, 0))
∧   proves (pf,
            f-implies (isin (z-list (list (rvar1,
                                           ncode (1),
                                           ncode (cdr (exp)),
                                           ncode (va),
                                           ncode (fa))),
                                   trace),
                        f-implies (dexp, f-equal (rvar1, ncode (rval1)))),
            given,
            defns,
            symbols)
∧   (concl = f-implies (a-ev-subr-part (exp, va, fa, val1, rvar1, trace),
                         f-implies (dexp,
                                    f-equal (val1,
                                             ncode (apply-subr (car (exp),
                                                                rval1)))))))
→   proves (a-ev-defn-part-subr-good-proof (exp,
                                            va,
                                            fa,
                                            val1,
                                            trace,
                                            rvar1,
                                            rval1,
                                            dexp,
                                            pf),
            concl,
            given,
            defns,
            symbols)
```

EVENT: Enable a-ev-defn-part-subr-good-proof; name this event 'g0921'.

THEOREM: parallel-subst-var
$(exp \in \mathbf{N}) \rightarrow (\text{parallel-subst}\,(exp,\ vars,\ terms,\ \texttt{0}) = \text{getpf}\,(exp,\ vars,\ terms))$

EVENT: Enable not-free-parallel-subst; name this event 'g0922'.

EVENT: Disable parallel-subst-nil; name this event 'g0923'.

EVENT: Enable get-graph; name this event 'g0924'.

DEFINITION:
a-ev-fun-part-body (*exp*, *fa*, *val1*, *trace*, *rvar1*, *getva*, *tr1*)
= f-implies (a-get (z-sub1-n (z-car (ncode (*exp*)), `11`),
ncode (*fa*),
getva,
`11`,
`13`,
`14`,
tr1),
isin (z-list (list (*val1*, ncode (`0`), *getva*, *rvar1*, ncode (*fa*))),
trace))

DEFINITION:
a-get-ev-good-reduc-proof (*exp*, *fa*, *getval*, *x1*, *y1*, *val1*, *tr1*)
= tautconseq-proof (list (f-implies (isin (z-list (list (*getval*,
ncode (sub1-n (car (*exp*),
`11`)),
ncode (*fa*))),
tr1),
isin (z-list (list (*getval*,
z-sub1-n (z-car (ncode (*exp*)),
`11`),
ncode (*fa*))),
tr1)),
a-get (ncode (sub1-n (car (*exp*), `11`)),
ncode (*fa*),
getval,
x1,
y1,
val1,
tr1)),
a-get (z-sub1-n (z-car (ncode (*exp*)), `11`),
ncode (*fa*),
getval,

```
                          x1,
                          y1,
                          val1,
                          tr1),
                    list(equal-z-list-isin-imp-proof(list(getval,
                                                          ncode(sub1-n(car(exp),
                                                                       11)),
                                                          ncode(fa)),
                                                     list(getval,
                                                          z-sub1-n(z-car(ncode(exp)),
                                                                   11),
                                                          ncode(fa)),
                                                     tr1,
                                                     list(ident-axiom-proof(getval),
                                                          flip-equals-proof(z-sub1-n(z-car(ncode(exp)),
                                                                                     11),
                                                                            ncode(sub1-n(car(exp),
                                                                                         11)),
                                                                            z-sub1-n-ncode-proof(z-car(ncode(exp)),
                                                                                                 11,
                                                                                                 car(exp),
                                                                                                 z-car-ncode-proof(exp))),
                                                          ident-axiom-proof(ncode(fa)))),
                      a-get-ok-proof1(sub1-n(car(exp), 11),
                                      fa,
                                      x1,
                                      y1,
                                      val1)))
```

THEOREM: a-get-ev-good-reduc-proof-proves

```
(subset(list(fn(10, 1),
             fn(9, 1),
             fn(5, 1),
             fn(4, 1),
             fn(3, 1),
             fn(2, 2),
             fn(1, 2),
             p(1, 2),
             p(2, 1),
             fn(0, 0)),
        symbols)
 ∧ z-succ-hyps(defns, symbols)
 ∧ z-car-hyps(defns, symbols)
 ∧ z-cdr-hyps(defns, symbols)
```

$\wedge$ z-zerop-hyps (*defns*, *symbols*)
$\wedge$ z-sub1-hyps (*defns*, *symbols*)
$\wedge$ z-int-hyps (*defns*, *symbols*)
$\wedge$ var-set (list (*x1*, *y1*, *val1*), 3)
$\wedge$ (*getval* = ncode (get (sub1-n (car (*exp*), 11), *fa*)))
$\wedge$ (*tr1* = graph-trans (get-graph (sub1-n (car (*exp*), 11), *fa*)))
$\wedge$ (*concl* = a-get (z-sub1-n (z-car (ncode (*exp*)), 11),
ncode (*fa*),
getval,
x1,
y1,
val1,
tr1)))
$\rightarrow$ proves (a-get-ev-good-reduc-proof (*exp*, *fa*, *getval*, *x1*, *y1*, *val1*, *tr1*),
concl,
given,
defns,
symbols)

EVENT: Enable a-get-ev-good-reduc-proof; name this event 'g0925'.

DEFINITION:
a-ev-defn-part-fun-good-proof (*exp*, *va*, *fa*, *val1*, *val*, *trace*, *rval1*, *dexp*, *pf1*, *pf2*)
= forsome-intro-proof (11,
body (a-ev-fun-part (*exp*, *va*, *fa*, *val1*, *trace*)),
f-implies (*dexp*, f-equal (*val1*, ncode (*val*))),
chain-proof (body (a-ev-fun-part (*exp*,
va,
fa,
val1,
trace)),
f-implies (*dexp*,
f-equal (*val1*, ncode (*val*))),
list (f-and (isin (z-list (list (11,
ncode (1),
z-cdr (ncode (*exp*)),
ncode (*va*),
ncode (*fa*))),
trace),
a-ev-fun-part-body (*exp*,
fa,
val1,
trace,
11,

```
                    ncode (get (sub1-n (car (exp),
                                        11),
                               fa)),
                    graph-trans (get-graph (sub1-n (car (exp),
                                                    11),
                                           fa))))),
list (subst-thm-b-proof (f-and (isin (z-list (list (11,
                                                    ncode (1),
                                                    z-cdr (ncode (exp)),
                                                    ncode (va),
                                                    ncode (fa))),
                                     trace),
                               a-ev-fun-part-body (exp,
                                                   fa,
                                                   val1,
                                                   trace,
                                                   11,
                                                   15,
                                                   16)),
                        list (15, 16),
                        list (ncode (get (sub1-n (car (exp),
                                                  11),
                                         fa)),
                              graph-trans (get-graph (sub1-n (car (exp),
                                                              11),
                                                     fa)))),
      a-ev-defn-good-step-proof (isin (z-list (list (11,
                                                     ncode (1),
                                                     z-cdr (ncode (exp)),
                                                     ncode (va),
                                                     ncode (fa))),
                                       trace),
                                 a-ev-fun-part-body (exp,
                                                     fa,
                                                     val1,
                                                     trace,
                                                     11,
                                                     ncode (get (sub1-n (car (exp),
                                                                         11),
                                                                fa)),
                                                     graph-trans (get-graph (sub1-n (car (exp),
                                                                                     11),
                                                                            fa))),
                                 dexp,
```

```
11,
rval1,
val1,
ncode (val),
a-ev-good-reduc-proof (1,
                       cdr (exp),
                       z-cdr (ncode (exp)),
                       va,
                       fa,
                       11,
                       trace,
                       f-implies (dexp,
                                  f-equal (11,
                                           ncode (rval1))),
                       z-cdr-ncode-proof (exp),
                       pf1),
tautconseq-proof (list (a-get (z-sub1-n (z-car (ncode (exp)),
                                         11),
                               ncode (fa),
                               ncode (get (sub1-n (car (exp),
                                                   11),
                                           fa)),
                               11,
                               13,
                               14,
                               graph-trans (get-graph (sub1-n (car (exp),
                                                               11),
                                                       fa))),
                      f-implies (isin (z-list (list (val1,
                                                     ncode (0),
                                                     ncode (get (sub1-n (car (exp),
                                                                         11),
                                                                 fa)),
                                                     ncode (rval1),
                                                     ncode (fa))),
                                       trace),
                                 f-implies (dexp,
                                            f-equal (val1,
                                                     ncode (val))))),
                 f-implies (a-ev-fun-part-body (exp,
                                                fa,
                                                val1,
                                                trace,
                                                ncode (rval1),
```

```
                                                                                                    ncode (get (sub1-n (car (exp),
                                                                                                                        11),
                                                                                                                fa)),
                                                                                                    graph-trans (get-graph (sub1-n (car (exp),
                                                                                                                                    11),
                                                                                                                            fa))),
                                                                                      f-implies (dexp,
                                                                                                 f-equal (val1,
                                                                                                          ncode (val)))),
                                                                              list (a-get-ev-good-reduc-proof (exp,
                                                                                                               fa,
                                                                                                               ncode (get (sub1-n (car (exp),
                                                                                                                                   11),
                                                                                                                           fa)),
                                                                                                               11,
                                                                                                               13,
                                                                                                               14,
                                                                                                               graph-trans (get-graph (sub1-n (car (exp),
                                                                                                                                               11),
                                                                                                                                       fa))),
                                                                                   pf2))))))
```

THEOREM: a-ev-defn-part-fun-good-proof-proves

```
(subset (list (fn (10, 1),
               fn (9, 1),
               fn (5, 1),
               fn (4, 1),
               fn (3, 1),
               fn (2, 2),
               fn (1, 2),
               p (1, 2),
               p (2, 1),
               fn (0, 0)),
          symbols)
 ∧   z-succ-hyps (defns, symbols)
 ∧   z-car-hyps (defns, symbols)
 ∧   z-cdr-hyps (defns, symbols)
 ∧   z-int-hyps (defns, symbols)
 ∧   z-zerop-hyps (defns, symbols)
 ∧   z-sub1-hyps (defns, symbols)
 ∧   var-set (list (11, 15, 16, val1, trace), 5)
 ∧   (11 ∉ collect-free (dexp, 0))
 ∧   proves (pf1,
             f-implies (isin (z-list (list (11,
```

ncode (1),
ncode (cdr (*exp*)),
ncode (*va*),
ncode (*fa*))),
trace),
f-implies (*dexp*, f-equal (11, ncode (*rval1*)))),
given,
defns,
symbols)
∧ proves (*pf2*,
f-implies (isin (z-list (list (*val1*,
ncode (0),
ncode (get (sub1-n (car (*exp*), 11), *fa*)),
ncode (*rval1*),
ncode (*fa*))),
trace),
f-implies (*dexp*, f-equal (*val1*, ncode (*val*)))),
given,
defns,
symbols)
∧ (*concl* = f-implies (a-ev-fun-part (*exp*, *va*, *fa*, *val1*, *trace*),
f-implies (*dexp*, f-equal (*val1*, ncode (*val*))))))
→ proves (a-ev-defn-part-fun-good-proof (*exp*,
va,
fa,
val1,
val,
trace,
rval1,
dexp,
pf1,
pf2),
concl,
given,
defns,
symbols)

EVENT: Enable a-ev-defn-part-fun-good-proof; name this event 'g0926'.

DEFINITION:
sub-rvar-proof (*flg*, *exp*, *va*, *fa*, *trace*, *rvar*, *val1*, *dexp*, *rval*, *pf*)
= subst-rule-proof (f-implies (isin (z-list (list (*val1*,
ncode (*flg*),
ncode (*exp*),

ncode (va),
ncode (fa))),
$trace$),
f-implies ($dexp$, f-equal ($val1$, ncode ($rval$)))),
$val1$,
$rvar$,
pf)

EVENT: Disable not-free-covering; name this event 'g0927'.

THEOREM: sub-rvar-proof-proves
(var-set (list ($val1$, $trace$), 2)
∧ termp ($rvar$, 0, $symbols$)
∧ ($val1 \notin$ collect-free ($dexp$, 0))
∧ proves (pf,
f-implies (isin (z-list (list ($val1$,
ncode (flg),
ncode (exp),
ncode (va),
ncode (fa))),
$trace$),
f-implies ($dexp$, f-equal ($val1$, ncode ($rval$)))),
$given$,
$defns$,
$symbols$)
∧ ($concl$ = f-implies (isin (z-list (list ($rvar$,
ncode (flg),
ncode (exp),
ncode (va),
ncode (fa))),
$trace$),
f-implies ($dexp$, f-equal ($rvar$, ncode ($rval$))))))
→ proves (sub-rvar-proof (flg, exp, va, fa, $trace$, $rvar$, $val1$, $dexp$, $rval$, pf),
$concl$,
$given$,
$defns$,
$symbols$)

EVENT: Enable sub-rvar-proof; name this event 'g0928'.

DEFINITION:
ind-form (flg, exp, va, fa, $val1$, val, $trace$, $dexp$)
= f-implies (isin (z-list (list ($val1$,

ncode (*flg*),
ncode (*exp*),
ncode (*va*),
ncode (*fa*))),
trace),
f-implies (*dexp*, f-equal (*val1*, ncode (*val*))))

THEOREM: form-z-if-ev-defn-part-again
(subset (list (fn (16, 1),
fn (15, 2),
fn (14, 2),
fn (13, 2),
fn (12, 2),
fn (0, 0),
fn (3, 1),
fn (4, 1),
fn (5, 1),
fn (6, 1),
fn (7, 1),
fn (8, 1),
fn (9, 1),
fn (10, 1),
fn (11, 2),
fn (1, 2),
fn (2, 2),
p (2, 1),
p (1, 2)),
symbols)
$\wedge$ termp (list (*val1*, *trace*), 1, *symbols*))
$\rightarrow$ form-z-if (z-ev-defn-part (*flg*, *exp*, *va*, *fa*, *val1*, *trace*), *symbols*)

THEOREM: ncode-equal-ident-axiom-proof-proves
(subset (list (fn (0, 0), fn (1, 2), fn (2, 2), fn (3, 1)), *symbols*)
$\wedge$ ncode-equal (x, y))
$\rightarrow$ proves (ident-axiom-proof (ncode (x)),
f-equal (ncode (x), ncode (y)),
given,
defns,
symbols)

EVENT: Disable a-ev-defn-part; name this event 'g0929'.

DEFINITION:
a-ev-defn-part-good-pftree (*flg*, *exp*, *va*, *fa*, *n*, *val1*, *trace*, *dexp*, *pf1*, *pf2*)

```
=  list (list (flg = 1,
               ident-axiom-proof (ncode (1)),
               ncode-nequal-proof (flg, 1)),
         list (list (listp (exp),
                     z-listp-ncode-proof (exp),
                     trans-not-equal-proof (ncode (1),
                                            z-listp (ncode (exp)),
                                            ncode (0),
                                            ncode-nequal-proof (1, 0),
                                            z-listp-ncode-proof (exp))),
               a-ev-defn-part-list-good-proof (exp,
                                               va,
                                               fa,
                                               val1,
                                               l1,
                                               l2,
                                               ev (0, car (exp), va, fa, n),
                                               ev (1, cdr (exp), va, fa, n),
                                               trace,
                                               dexp,
                                               sub-rvar-proof (0,
                                                               car (exp),
                                                               va,
                                                               fa,
                                                               trace,
                                                               l1,
                                                               val1,
                                                               dexp,
                                                               ev (0,
                                                                   car (exp),
                                                                   va,
                                                                   fa,
                                                                   n),
                                                               pf1),
                                               sub-rvar-proof (1,
                                                               cdr (exp),
                                                               va,
                                                               fa,
                                                               trace,
                                                               l2,
                                                               val1,
                                                               dexp,
                                                               ev (1,
                                                                   cdr (exp),
```

```
                                                    va,
                                                    fa,
                                                    n),
                                                  pf2)),
  tautconseq-proof(nil,
                   f-implies(f-equal(val1, ncode(nil)),
                             f-implies(dexp,
                                       f-equal(val1,
                                               ncode(nil)))),
                   nil)),
list(list(listp(exp),
          z-listp-ncode-proof(exp),
          trans-not-equal-proof(ncode(1),
                                z-listp(ncode(exp)),
                                ncode(0),
                                ncode-nequal-proof(1, 0),
                                z-listp-ncode-proof(exp))),
  list(list(car(exp) ≃ 0,
            compose-car-proof(fn(9, 1),
                              ncode(1),
                              exp,
                              z-zerop-ncode-proof(car(exp))),
            trans-not-equal-proof(ncode(1),
                                  z-zerop(z-car(ncode(exp))),
                                  ncode(0),
                                  ncode-nequal-proof(1, 0),
                                  compose-car-proof(fn(9, 1),
                                                    ncode(0),
                                                    exp,
                                                    z-zerop-ncode-proof(car(exp))))),
       a-ev-defn-part-if-good-proof(exp,
                                    va,
                                    fa,
                                    val1,
                                    ev(flg, exp, va, fa, n),
                                    trace,
                                    11,
                                    ev(0, cadr(exp), va, fa, n),
                                    dexp,
                                    sub-rvar-proof(0,
                                                   cadr(exp),
                                                   va,
                                                   fa,
                                                   trace,
```

```
                                              11,
                                              val1,
                                              dexp,
                                              ev (0,
                                                  cadr (exp),
                                                  va,
                                                  fa,
                                                  n),
                                              pf1),
                                 pf2),
list (list (car (exp) = 10,
            z-car-ncode-proof (exp),
            trans-not-equal-proof (ncode (10),
                                   z-car (ncode (exp)),
                                   ncode (car (exp)),
                                   ncode-nequal-proof (10,
                                                       car (exp)),
                                   z-car-ncode-proof (exp))),
      tautconseq-proof (list (f-equal (z-car-cdr (ncode (exp),
                                                  list (1, 2)),
                                       ncode (cadr (exp))),
                              f-implies (f-equal (val1,
                                                  z-car-cdr (ncode (exp),
                                                             list (1,
                                                                   2))),
                                         f-implies (f-equal (z-car-cdr (ncode (exp),
                                                                        list (1,
                                                                              2)),
                                                             ncode (cadr (exp))),
                                                    f-equal (val1,
                                                             ncode (cadr (exp)))))),
                        f-implies (f-equal (val1,
                                            z-car-cdr (ncode (exp),
                                                       list (1,
                                                             2))),
                                   f-implies (dexp,
                                              f-equal (val1,
                                                       ncode (cadr (exp))))),
                        list (z-car-cdr-proof (exp,
                                               list (1, 2)),
                              trans-equal-proof3 (val1,
                                                  z-car-cdr (ncode (exp),
                                                             list (1,
                                                                   2)),
```

```
                                        ncode (cadr (exp))))),
list (list (subrp (car (exp)),
        compose-car-proof (fn (16, 1),
                           ncode (1),
                           exp,
                           z-subrp-ncode-proof (car (exp))),
        trans-not-equal-proof (ncode (1),
                               z-subrp (z-car (ncode (exp))),
                               ncode (0),
                               ncode-nequal-proof (1,
                                                   0),
                               compose-car-proof (fn (16,
                                                      1),
                                                  ncode (0),
                                                  exp,
                                                  z-subrp-ncode-proof (car (exp))))),
    a-ev-defn-part-subr-good-proof (exp,
                                    va,
                                    fa,
                                    val1,
                                    trace,
                                    11,
                                    ev (1,
                                        cdr (exp),
                                        va,
                                        fa,
                                        n),
                                    dexp,
                                    sub-rvar-proof (1,
                                                    cdr (exp),
                                                    va,
                                                    fa,
                                                    trace,
                                                    11,
                                                    val1,
                                                    dexp,
                                                    ev (1,
                                                        cdr (exp),
                                                        va,
                                                        fa,
                                                        n),
                                                    pf1)),
    a-ev-defn-part-fun-good-proof (exp,
                                   va,
```

```
                                        fa,
                                        val1,
                                        ev (flg,
                                            exp,
                                            va,
                                            fa,
                                            n),
                                        trace,
                                        ev (1,
                                            cdr (exp),
                                            va,
                                            fa,
                                            n),
                                        dexp,
                                        sub-rvar-proof (1,
                                                        cdr (exp),
                                                        va,
                                                        fa,
                                                        trace,
                                                        11,
                                                        val1,
                                                        dexp,
                                                        ev (1,
                                                            cdr (exp),
                                                            va,
                                                            fa,
                                                            n),
                                                        pf1),
                                        pf2)))),
tautconseq-proof (list (f-implies (a-get1 (ncode (exp),
                                          ncode (va),
                                          val1,
                                          11,
                                          13,
                                          14,
                                          16),
                                  f-equal (val1,
                                           ncode (ev (flg,
                                                      exp,
                                                      va,
                                                      fa,
                                                      n))))),
                       f-implies (a-get1 (ncode (exp),
                                         ncode (va),
```

```
                                     val1,
                                     11,
                                     13,
                                     14,
                                     16),
                              f-implies(dexp,
                                        f-equal(val1,
                                                ncode(ev(flg,
                                                         exp,
                                                         va,
                                                         fa,
                                                         n))))),
                     list(a-get1-good-proof(exp,
                                            va,
                                            val1,
                                            11,
                                            13,
                                            14,
                                            16)))))
```

DEFINITION:
a-ev-defn-part-good-proof(*flg*, *exp*, *va*, *fa*, *n*, *val1*, *trace*, *dexp*, *pf1*, *pf2*)

```
=    rec-form-z-if-eval-proof(a-ev-defn-part-good-pftree(flg,
                                                          exp,
                                                          va,
                                                          fa,
                                                          n,
                                                          val1,
                                                          trace,
                                                          dexp,
                                                          pf1,
                                                          pf2),
                              z-ev-defn-part(flg, exp, va, fa, val1, trace),
                              f-implies(dexp,
                                        f-equal(val1,
                                                ncode(ev(flg, exp, va, fa, n)))))
```

THEOREM: a-ev-defn-part-good-proof-proves-quote-case
$((flg \neq 1)$
$\land$ listp(*exp*)
$\land$ (car(*exp*) $=$ 10)
$\land$ ev-hyps(*given*, *defns*, *symbols*)
$\land$ var-set(list(11, 12, 15, 16, *val1*, *trace*), 6)
$\land$ formula(*dexp*, *symbols*)

$$
\begin{array}{ll}
\land & (\neg\ \text{btmp}\,(\text{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))\\
\land & (\mathit{concl} = \text{f-implies}\,(\text{a-ev-defn-part}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ \mathit{val1},\ \mathit{trace}),\\
 & \qquad\qquad\qquad \text{f-implies}\,(\mathit{dexp},\\
 & \qquad\qquad\qquad\qquad \text{f-equal}\,(\mathit{val1},\\
 & \qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\text{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))))))\\
\rightarrow & \text{proves}\,(\text{a-ev-defn-part-good-proof}\,(\mathit{flg},\\
 & \qquad\qquad \mathit{exp},\\
 & \qquad\qquad \mathit{va},\\
 & \qquad\qquad \mathit{fa},\\
 & \qquad\qquad n,\\
 & \qquad\qquad \mathit{val1},\\
 & \qquad\qquad \mathit{trace},\\
 & \qquad\qquad \mathit{dexp},\\
 & \qquad\qquad \mathit{pf1},\\
 & \qquad\qquad \mathit{pf2}),\\
 & \quad \mathit{concl},\\
 & \quad \mathit{given},\\
 & \quad \mathit{defns},\\
 & \quad \mathit{symbols})
\end{array}
$$

Theorem: a-ev-defn-part-good-proof-proves-list-case

$$
\begin{array}{ll}
((\mathit{flg} = \mathtt{1}) & \\
\land & \text{listp}\,(\mathit{exp})\\
\land & \text{ev-hyps}\,(\mathit{given},\ \mathit{defns},\ \mathit{symbols})\\
\land & \text{var-set}\,(\text{list}\,(\mathtt{11},\ \mathtt{12},\ \mathtt{15},\ \mathtt{16},\ \mathit{val1},\ \mathit{trace}),\ \mathtt{6})\\
\land & \text{nil-intersect}\,(\text{list}\,(\mathtt{11},\ \mathtt{12},\ \mathit{val1}),\ \text{collect-free}\,(\mathit{dexp},\ \mathtt{0}))\\
\land & (\neg\ \text{btmp}\,(\text{ev}\,(\mathtt{1},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))\\
\land & (\text{proves}\,(\mathit{pf1},\\
 & \quad \text{ind-form}\,(\mathtt{0},\\
 & \qquad \text{car}\,(\mathit{exp}),\\
 & \qquad \mathit{va},\\
 & \qquad \mathit{fa},\\
 & \qquad \mathit{val1},\\
 & \qquad \text{ev}\,(\mathtt{0},\ \text{car}\,(\mathit{exp}),\ \mathit{va},\ \mathit{fa},\ n),\\
 & \qquad \mathit{trace},\\
 & \qquad \mathit{dexp}),\\
 & \quad \mathit{given},\\
 & \quad \mathit{defns},\\
 & \quad \mathit{symbols})\\
 & \land\quad \text{proves}\,(\mathit{pf2},\\
 & \qquad \text{ind-form}\,(\mathtt{1},\\
 & \qquad\quad \text{cdr}\,(\mathit{exp}),\\
 & \qquad\quad \mathit{va},\\
 & \qquad\quad \mathit{fa},
\end{array}
$$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{val1},\\
\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{ev}\,(\mathtt{1},\ \mathrm{cdr}\,(\mathit{exp}),\ \mathit{va},\ \mathit{fa},\ n),\\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{trace},\\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{dexp}),\\
\qquad\qquad\qquad\qquad \mathit{given},\\
\qquad\qquad\qquad\qquad \mathit{defns},\\
\qquad\qquad\qquad\qquad \mathit{symbols}))\\
\land \quad (\mathit{concl} = \text{f-implies}\,(\text{a-ev-defn-part}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ \mathit{val1},\ \mathit{trace}),\\
\qquad\qquad\qquad\qquad\qquad \text{f-implies}\,(\mathit{dexp},\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{f-equal}\,(\mathit{val1},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{ncode}\,(\mathrm{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))))))\\
\to \quad \mathrm{proves}\,(\text{a-ev-defn-part-good-proof}\,(\mathit{flg},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{exp},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{va},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{fa},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad n,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{val1},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{trace},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{dexp},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{pf1},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{pf2}),\\
\qquad\qquad \mathit{concl},\\
\qquad\qquad \mathit{given},\\
\qquad\qquad \mathit{defns},\\
\qquad\qquad \mathit{symbols})
\end{array}
$$

THEOREM: a-ev-defn-part-good-proof-proves-nlist-case

$$
\begin{array}{l}
((\mathit{flg} = \mathtt{1})\\
\land \quad (\mathit{exp} \simeq \mathbf{nil})\\
\land \quad \text{ev-hyps}\,(\mathit{given},\ \mathit{defns},\ \mathit{symbols})\\
\land \quad \text{var-set}\,(\mathrm{list}\,(\mathtt{11},\ \mathtt{12},\ \mathtt{15},\ \mathtt{16},\ \mathit{val1},\ \mathit{trace}),\ \mathtt{6})\\
\land \quad \mathrm{formula}\,(\mathit{dexp},\ \mathit{symbols})\\
\land \quad (\neg\ \mathrm{btmp}\,(\mathrm{ev}\,(\mathtt{1},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))\\
\land \quad (\mathit{concl} = \text{f-implies}\,(\text{a-ev-defn-part}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ \mathit{val1},\ \mathit{trace}),\\
\qquad\qquad\qquad\qquad\qquad \text{f-implies}\,(\mathit{dexp},\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{f-equal}\,(\mathit{val1},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{ncode}\,(\mathrm{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n)))))))\\
\to \quad \mathrm{proves}\,(\text{a-ev-defn-part-good-proof}\,(\mathit{flg},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{exp},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{va},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{fa},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad n,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{val1},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{trace},
\end{array}
$$

dexp,
pf1,
pf2),
concl,
given,
defns,
symbols)

THEOREM: a-ev-defn-part-good-proof-proves-if-case
(($flg \neq$ 1)
$\land$ listp (*exp*)
$\land$ (car (*exp*) $\simeq$ 0)
$\land$ ev-hyps (*given*, *defns*, *symbols*)
$\land$ var-set (list (11, 12, 15, 16, *val1*, *trace*), 6)
$\land$ nil-intersect (list (11, 12, *val1*), collect-free (*dexp*, 0))
$\land$ ($\neg$ btmp (ev (*flg*, *exp*, *va*, *fa*, *n*)))
$\land$ (proves (*pf1*,
ind-form (0,
cadr (*exp*),
va,
fa,
val1,
ev (0, cadr (*exp*), *va*, *fa*, *n*),
trace,
dexp),
given,
defns,
symbols)
$\land$ **if** ncode-equal (ev (0, cadr (*exp*), *va*, *fa*, *n*), 0)
then proves (*pf2*,
ind-form (0,
cadddr (*exp*),
va,
fa,
val1,
ev (0, cadddr (*exp*), *va*, *fa*, *n*),
trace,
dexp),
given,
defns,
symbols)
else proves (*pf2*,
ind-form (0,
caddr (*exp*),

```
                                        va,
                                        fa,
                                        val1,
                                        ev (0, caddr (exp), va, fa, n),
                                        trace,
                                        dexp),
                                 given,
                                 defns,
                                 symbols) endif)
  ∧  (concl = f-implies (a-ev-defn-part (flg, exp, va, fa, val1, trace),
                                  f-implies (dexp,
                                             f-equal (val1,
                                                      ncode (ev (flg, exp, va, fa, n)))))))
  →  proves (a-ev-defn-part-good-proof (flg,
                                         exp,
                                         va,
                                         fa,
                                         n,
                                         val1,
                                         trace,
                                         dexp,
                                         pf1,
                                         pf2),
             concl,
             given,
             defns,
             symbols)
```

THEOREM: a-ev-defn-part-good-proof-proves-subr-case

```
 ((flg ≠ 1)
  ∧  listp (exp)
  ∧  (car (exp) ≄ 0)
  ∧  (car (exp) ≠ 10)
  ∧  subrp (car (exp))
  ∧  ev-hyps (given, defns, symbols)
  ∧  var-set (list (11, 12, 15, 16, val1, trace), 6)
  ∧  nil-intersect (list (11, 12, val1), collect-free (dexp, 0))
  ∧  (¬ btmp (ev (flg, exp, va, fa, n)))
  ∧  proves (pf1,
             ind-form (1,
                       cdr (exp),
                       va,
                       fa,
                       val1,
```

$$
\begin{array}{ll}
 & \qquad\qquad\qquad \mathrm{ev}(\mathtt{1},\ \mathrm{cdr}(\mathit{exp}),\ \mathit{va},\ \mathit{fa},\ n), \\
 & \qquad\qquad\qquad \mathit{trace}, \\
 & \qquad\qquad\qquad \mathit{dexp}), \\
 & \qquad \mathit{given}, \\
 & \qquad \mathit{defns}, \\
 & \qquad \mathit{symbols}) \\
\wedge & (\mathit{concl} = \text{f-implies}(\text{a-ev-defn-part}(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ \mathit{val1},\ \mathit{trace}), \\
 & \qquad\qquad\qquad \text{f-implies}(\mathit{dexp}, \\
 & \qquad\qquad\qquad\qquad \text{f-equal}(\mathit{val1}, \\
 & \qquad\qquad\qquad\qquad\qquad \mathrm{ncode}(\mathrm{ev}(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n))))))) \\
\rightarrow & \mathrm{proves}(\text{a-ev-defn-part-good-proof}(\mathit{flg}, \\
 & \qquad\qquad\qquad\qquad \mathit{exp}, \\
 & \qquad\qquad\qquad\qquad \mathit{va}, \\
 & \qquad\qquad\qquad\qquad \mathit{fa}, \\
 & \qquad\qquad\qquad\qquad n, \\
 & \qquad\qquad\qquad\qquad \mathit{val1}, \\
 & \qquad\qquad\qquad\qquad \mathit{trace}, \\
 & \qquad\qquad\qquad\qquad \mathit{dexp}, \\
 & \qquad\qquad\qquad\qquad \mathit{pf1}, \\
 & \qquad\qquad\qquad\qquad \mathit{pf2}), \\
 & \qquad \mathit{concl}, \\
 & \qquad \mathit{given}, \\
 & \qquad \mathit{defns}, \\
 & \qquad \mathit{symbols})
\end{array}
$$

THEOREM: a-ev-defn-part-good-proof-proves-fun-case

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\mathit{flg} \neq \mathtt{1})} \\
\wedge & \mathrm{listp}(\mathit{exp}) \\
\wedge & (\mathrm{car}(\mathit{exp}) \not\simeq \mathtt{0}) \\
\wedge & (\mathrm{car}(\mathit{exp}) \neq \mathtt{10}) \\
\wedge & (\neg\ \mathrm{subrp}(\mathrm{car}(\mathit{exp}))) \\
\wedge & \text{ev-hyps}(\mathit{given},\ \mathit{defns},\ \mathit{symbols}) \\
\wedge & \text{var-set}(\mathrm{list}(\mathtt{11},\ \mathtt{12},\ \mathtt{15},\ \mathtt{16},\ \mathit{val1},\ \mathit{trace}),\ \mathtt{6}) \\
\wedge & \text{nil-intersect}(\mathrm{list}(\mathtt{11},\ \mathtt{12},\ \mathit{val1}),\ \text{collect-free}(\mathit{dexp},\ \mathtt{0})) \\
\wedge & (\neg\ \mathrm{btmp}(\mathrm{ev}(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n))) \\
\wedge & \mathrm{proves}(\mathit{pf1}, \\
 & \qquad \text{ind-form}(\mathtt{1}, \\
 & \qquad\qquad\qquad \mathrm{cdr}(\mathit{exp}), \\
 & \qquad\qquad\qquad \mathit{va}, \\
 & \qquad\qquad\qquad \mathit{fa}, \\
 & \qquad\qquad\qquad \mathit{val1}, \\
 & \qquad\qquad\qquad \mathrm{ev}(\mathtt{1},\ \mathrm{cdr}(\mathit{exp}),\ \mathit{va},\ \mathit{fa},\ n), \\
 & \qquad\qquad\qquad \mathit{trace}, \\
 & \qquad\qquad\qquad \mathit{dexp}),
\end{array}
$$

            given,
            defns,
            symbols)
$\land$ proves (*pf2*,
            ind-form (0,
                        get (sub1-n (car (*exp*), 11), *fa*),
                        ev (1, cdr (*exp*), *va*, *fa*, *n*),
                        fa,
                        val1,
                        ev (*flg*, *exp*, *va*, *fa*, *n*),
                        trace,
                        dexp),
            given,
            defns,
            symbols)
$\land$ (*concl* $=$ f-implies (a-ev-defn-part (*flg*, *exp*, *va*, *fa*, *val1*, *trace*),
                          f-implies (*dexp*,
                                    f-equal (*val1*,
                                            ncode (ev (*flg*, *exp*, *va*, *fa*, *n*)))))))
$\rightarrow$ proves (a-ev-defn-part-good-proof (*flg*,
                                        exp,
                                        va,
                                        fa,
                                        n,
                                        val1,
                                        trace,
                                        dexp,
                                        pf1,
                                        pf2),
            concl,
            given,
            defns,
            symbols)

THEOREM: a-ev-defn-part-good-proof-proves-var-case
((*flg* $\neq$ 1)
$\land$ ($\neg$ listp (*exp*))
$\land$ ev-hyps (*given*, *defns*, *symbols*)
$\land$ formula (*dexp*, *symbols*)
$\land$ var-set (list (11, 12, 15, 16, *val1*, *trace*), 6)
$\land$ nil-intersect (list (11, 12, *val1*), collect-free (*dexp*, 0))
$\land$ ($\neg$ btmp (ev (*flg*, *exp*, *va*, *fa*, *n*)))
$\land$ (*concl* $=$ f-implies (a-ev-defn-part (*flg*, *exp*, *va*, *fa*, *val1*, *trace*),
                          f-implies (*dexp*,

f-equal (*val1*,
ncode (ev (*flg*, *exp*, *va*, *fa*, *n*)))))))
→ proves (a-ev-defn-part-good-proof (*flg*,
exp,
va,
fa,
n,
val1,
trace,
dexp,
pf1,
pf2),
concl,
given,
defns,
symbols)

EVENT: Enable a-ev-defn-part-good-proof; name this event 'g0930'.

THEOREM: free-for-x-x
variable (*x*) → free-for (*exp*, *x*, *x*, *flg*)

EVENT: Enable free-for; name this event 'g0932'.

EVENT: Disable free-for-list; name this event 'g0933'.

DEFINITION:
a-ev-reduc-proof (*flg*, *exp*, *va*, *fa*, *val1*, *trace*)
= f-iff-mp-proof2 (f-implies (a-ev (ncode (*flg*),
ncode (*exp*),
ncode (*va*),
ncode (*fa*),
val1,
trace),
a-ev-defn-part (*flg*, *exp*, *va*, *fa*, *val1*, *trace*)),
f-implies (list-all-quantify (list (6, 7, 8, 9, 10),
f-implies (isin (z-list (list (10,
6,
7,
8,
9)),
trace),
a-ev-defn-part1 (6,

```
                                                    7,
                                                    8,
                                                    9,
                                                    10,
                                                    trace))),
      f-implies (isin (z-list (list (val1,
                                     ncode (flg),
                                     ncode (exp),
                                     ncode (va),
                                     ncode (fa))),
                         trace),
                 a-ev-defn-part (flg,
                                 exp,
                                 va,
                                 fa,
                                 val1,
                                 trace))),
good-frame1-proof (isin (z-list (list (val1,
                                       ncode (flg),
                                       ncode (exp),
                                       ncode (va),
                                       ncode (fa))),
                         trace),
                   list-all-quantify (list (6, 7, 8, 9, 10),
                                      f-implies (isin (z-list (list (10,
                                                                     6,
                                                                     7,
                                                                     8,
                                                                     9)),
                                                       trace),
                                                 a-ev-defn-part1 (6,
                                                                  7,
                                                                  8,
                                                                  9,
                                                                  10,
                                                                  trace))),
                   a-ev-defn-part (flg,
                                   exp,
                                   va,
                                   fa,
                                   val1,
                                   trace)),
subst-thm-b-proof (f-implies (isin (z-list (list (10,
                                                  6,
```

7,
8,
9)),
trace),
a-ev-defn-part1 (6,
7,
8,
9,
10,
trace)),
list (6, 7, 8, 9, 10),
list (ncode (*flg*),
ncode (*exp*),
ncode (*va*),
ncode (*fa*),
val1)))

THEOREM: parallel-subst-a-ev-defn-part1-again
(variable (*trace*) ∧ (*trace* ∉ list (6, 7, 8, 9, 10)))
→ (parallel-subst (a-ev-defn-part1 (6, 7, 8, 9, 10, *trace*),
list (6, 7, 8, 9, 10),
list (ncode (*flg*),
ncode (*exp*),
ncode (*va*),
ncode (*fa*),
val-bar),
0)
= a-ev-defn-part (*flg*, *exp*, *va*, *fa*, *val-bar*, *trace*))

EVENT: Enable a-ev-defn-part; name this event 'g0935'.

THEOREM: free-for-list-reduc
free-for-list (*exp*, *vars*, *terms*)
= **if** listp (*vars*) ∧ listp (*terms*)
then free-for (*exp*, car (*vars*), car (*terms*), 0)
∧ free-for-list (*exp*, cdr (*vars*), cdr (*terms*))
else t endif

EVENT: Enable free-for-list; name this event 'g0940'.

DEFINITION:
all-symbs (*symbols*)
= subset (list (fn (16, 1),
fn (15, 2),

fn (14, 2),
fn (13, 2),
fn (12, 2),
fn (0, 0),
fn (3, 1),
fn (4, 1),
fn (5, 1),
fn (6, 1),
fn (7, 1),
fn (8, 1),
fn (9, 1),
fn (10, 1),
fn (11, 2),
fn (1, 2),
fn (2, 2),
p (2, 1),
p (1, 2)),
symbols)

THEOREM: a-ev-reduc-proof-proves
(all-symbs (*symbols*)
∧ variable (*trace*)
∧ (*trace* $\not\in$ list (6, 7, 8, 9, 10))
∧ (*concl* = f-implies (a-ev (ncode (*flg*),
ncode (*exp*),
ncode (*va*),
ncode (*fa*),
10,
trace),
a-ev-defn-part (*flg*, *exp*, *va*, *fa*, 10, *trace*))))
→ proves (a-ev-reduc-proof (*flg*, *exp*, *va*, *fa*, 10, *trace*),
concl,
given,
defns,
symbols)

EVENT: Enable a-ev-reduc-proof; name this event ‘g0947’.

EVENT: Disable a-ev-defn-part1; name this event ‘g0949’.

DEFINITION:
a-ev-defn (*trace*)
= list-all-quantify (list (6, 7, 8, 9, 10),

f-implies (isin (z-list (list (10, 6, 7, 8, 9)), *trace*),
a-ev-defn-part1 (6, 7, 8, 9, 10, *trace*)))

THEOREM: collect-free-z-sub1-n
($var \in$ collect-free (z-sub1-n (exp, n), 0))
$=$ ($var \in$ collect-free (exp, 0))

THEOREM: nil-int-a-ev-defn
(variable (*trace*) $\land$ (*trace* $\notin$ list (11, 12, 10)))
$\rightarrow$ nil-intersect (list (11, 12, 10), collect-free (a-ev-defn (*trace*), 0))

THEOREM: formula-a-ev-defn
(all-symbs (*symbols*) $\land$ variable (*trace*))
$\rightarrow$ formula (a-ev-defn (*trace*), *symbols*)

EVENT: Enable a-ev-defn-part1; name this event 'g0950'.

THEOREM: a-ev-reduc
a-ev (*flg-bar*, *exp-bar*, *va-bar*, *fa-bar*, *val1*, *trace*)
$=$ f-and (isin (z-list (list (*val1*, *flg-bar*, *exp-bar*, *va-bar*, *fa-bar*)), *trace*),
a-ev-defn (*trace*))

EVENT: Enable a-ev-defn; name this event 'g0951'.

EVENT: Enable a-ev; name this event 'g0952'.

THEOREM: good-frame2-proof-proves-again
(proves (*pf1*, f-implies (f-and (a, b), c), *given*, *defns*, *symbols*)
$\land$ proves (*pf2*, f-implies (c, f-implies (b, d)), *given*, *defns*, *symbols*))
$\rightarrow$ proves (good-frame2-proof (a, b, c, d, *pf1*, *pf2*),
f-implies (f-and (a, b), d),
given,
defns,
symbols)

EVENT: Enable good-frame2-proof-proves; name this event 'g0953'.

DEFINITION:
a-ev-good-proof (*flg*, *exp*, *va*, *fa*, *n*, *trace*)
$=$ **if** *flg* $=$ 1
then if listp (*exp*)
then good-frame2-proof (isin (z-list (list (10,
ncode (*flg*),

```
                    ncode (exp),
                    ncode (va),
                    ncode (fa))),
          trace),
     a-ev-defn (trace),
     a-ev-defn-part (flg, exp, va, fa, 10, trace),
     f-equal (10, ncode (ev (flg, exp, va, fa, n))),
     a-ev-reduc-proof (flg, exp, va, fa, 10, trace),
     a-ev-defn-part-good-proof (flg,
                                exp,
                                va,
                                fa,
                                n,
                                10,
                                trace,
                                a-ev-defn (trace),
                                good-frame3-proof (isin (z-list (list (10,
                                                                      ncode (0),
                                                                      ncode (car (exp)),
                                                                      ncode (va),
                                                                      ncode (fa))),
                                                            trace),
                                                   a-ev-defn (trace),
                                                   f-equal (10,
                                                            ncode (ev (0,
                                                                       car (exp),
                                                                       va,
                                                                       fa,
                                                                       n))),
                                                   a-ev-good-proof (0,
                                                                    car (exp),
                                                                    va,
                                                                    fa,
                                                                    n,
                                                                    trace)),
                                good-frame3-proof (isin (z-list (list (10,
                                                                      ncode (1),
                                                                      ncode (cdr (exp)),
                                                                      ncode (va),
                                                                      ncode (fa))),
                                                            trace),
                                                   a-ev-defn (trace),
                                                   f-equal (10,
                                                            ncode (ev (1,
```

```
                                                                                      cdr (exp),
                                                                                      va,
                                                                                      fa,
                                                                                      n))),
                                                                 a-ev-good-proof (1,
                                                                                  cdr (exp),
                                                                                  va,
                                                                                  fa,
                                                                                  n,
                                                                                  trace))))
   else good-frame2-proof (isin (z-list (list (10,
                                               ncode (flg),
                                               ncode (exp),
                                               ncode (va),
                                               ncode (fa))),
                                       trace),
                                 a-ev-defn (trace),
                                 a-ev-defn-part (flg,
                                                 exp,
                                                 va,
                                                 fa,
                                                 10,
                                                 trace),
                                 f-equal (10,
                                          ncode (ev (flg, exp, va, fa, n))),
                                 a-ev-reduc-proof (flg,
                                                   exp,
                                                   va,
                                                   fa,
                                                   10,
                                                   trace),
                                 a-ev-defn-part-good-proof (flg,
                                                            exp,
                                                            va,
                                                            fa,
                                                            n,
                                                            10,
                                                            trace,
                                                            a-ev-defn (trace),
                                                            nil,
                                                            nil)) endif
elseif listp (exp)
then if car (exp) ≃ 0
   then good-frame2-proof (isin (z-list (list (10,
```

```
               ncode (flg),
               ncode (exp),
               ncode (va),
               ncode (fa))),
     trace),
a-ev-defn (trace),
a-ev-defn-part (flg, exp, va, fa, 10, trace),
f-equal (10, ncode (ev (flg, exp, va, fa, n))),
a-ev-reduc-proof (flg, exp, va, fa, 10, trace),
a-ev-defn-part-good-proof (flg,
                          exp,
                          va,
                          fa,
                          n,
                          10,
                          trace,
                          a-ev-defn (trace),
                          good-frame3-proof (isin (z-list (list (10,
                                                                  ncode (0),
                                                                  ncode (cadr (exp)),
                                                                  ncode (va),
                                                                  ncode (fa))),
                                                        trace),
                                                   a-ev-defn (trace),
                                                   f-equal (10,
                                                           ncode (ev (0,
                                                                      cadr (exp),
                                                                      va,
                                                                      fa,
                                                                      n))),
                                                   a-ev-good-proof (0,
                                                                    cadr (exp),
                                                                    va,
                                                                    fa,
                                                                    n,
                                                                    trace)),
                          if ncode-equal (ev (0,
                                              cadr (exp),
                                              va,
                                              fa,
                                              n),
                                          0)
                          then good-frame3-proof (isin (z-list (list (10,
                                                                       ncode (0),
```

```
                                                                ncode (cadddr (exp)),
                                                                ncode (va),
                                                                ncode (fa))),
                                                           trace),
                                                        a-ev-defn (trace),
                                                        f-equal (10,
                                                                 ncode (ev (0,
                                                                            cadddr (exp),
                                                                            va,
                                                                            fa,
                                                                            n))),
                                                        a-ev-good-proof (0,
                                                                         cadddr (exp),
                                                                         va,
                                                                         fa,
                                                                         n,
                                                                         trace))
                                     else good-frame3-proof (isin (z-list (list (10,
                                                                ncode (0),
                                                                ncode (caddr (exp)),
                                                                ncode (va),
                                                                ncode (fa))),
                                                           trace),
                                                        a-ev-defn (trace),
                                                        f-equal (10,
                                                                 ncode (ev (0,
                                                                            caddr (exp),
                                                                            va,
                                                                            fa,
                                                                            n))),
                                                        a-ev-good-proof (0,
                                                                         caddr (exp),
                                                                         va,
                                                                         fa,
                                                                         n,
                                                                         trace)) endif))
elseif car (exp) = 10
then good-frame2-proof (isin (z-list (list (10,
                                            ncode (flg),
                                            ncode (exp),
                                            ncode (va),
                                            ncode (fa))),
                                       trace),
                                   a-ev-defn (trace),
```

```
                         a-ev-defn-part (flg, exp, va, fa, 10, trace),
                         f-equal (10, ncode (ev (flg, exp, va, fa, n))),
                         a-ev-reduc-proof (flg, exp, va, fa, 10, trace),
                         a-ev-defn-part-good-proof (flg,
                                                    exp,
                                                    va,
                                                    fa,
                                                    n,
                                                    10,
                                                    trace,
                                                    a-ev-defn (trace),
                                                    nil,
                                                    nil))
elseif subrp (car (exp))
then good-frame2-proof (isin (z-list (list (10,
                                             ncode (flg),
                                             ncode (exp),
                                             ncode (va),
                                             ncode (fa))),
                               trace),
                         a-ev-defn (trace),
                         a-ev-defn-part (flg, exp, va, fa, 10, trace),
                         f-equal (10, ncode (ev (flg, exp, va, fa, n))),
                         a-ev-reduc-proof (flg, exp, va, fa, 10, trace),
                         a-ev-defn-part-good-proof (flg,
                                                    exp,
                                                    va,
                                                    fa,
                                                    n,
                                                    10,
                                                    trace,
                                                    a-ev-defn (trace),
                                                    good-frame3-proof (isin (z-list (list (10,
                                                                                           ncode (1),
                                                                                           ncode (cdr (exp)),
                                                                                           ncode (va),
                                                                                           ncode (fa))),
                                                                             trace),
                                                                       a-ev-defn (trace),
                                                                       f-equal (10,
                                                                                ncode (ev (1,
                                                                                           cdr (exp),
                                                                                           va,
                                                                                           fa,
```

```
                                                                    n))),
                                                      a-ev-good-proof (1,
                                                                       cdr (exp),
                                                                       va,
                                                                       fa,
                                                                       n,
                                                                       trace)),
                                              nil))
elseif n ≃ 0 then nil
else good-frame2-proof (isin (z-list (list (10,
                                            ncode (flg),
                                            ncode (exp),
                                            ncode (va),
                                            ncode (fa))),
                              trace),
                        a-ev-defn (trace),
                        a-ev-defn-part (flg,
                                        exp,
                                        va,
                                        fa,
                                        10,
                                        trace),
                        f-equal (10,
                                 ncode (ev (flg, exp, va, fa, n))),
                        a-ev-reduc-proof (flg,
                                          exp,
                                          va,
                                          fa,
                                          10,
                                          trace),
                        a-ev-defn-part-good-proof (flg,
                                                   exp,
                                                   va,
                                                   fa,
                                                   n,
                                                   10,
                                                   trace,
                                                   a-ev-defn (trace),
                                                   good-frame3-proof (isin (z-list (list (10,
                                                                                         ncode (1),
                                                                                         ncode (cdr (exp)),
                                                                                         ncode (va),
                                                                                         ncode (fa))),
                                                                            trace),
```

```
                      a-ev-defn (trace),
                      f-equal (10,
                               ncode (ev (1,
                                          cdr (exp),
                                          va,
                                          fa,
                                          n))),
                      a-ev-good-proof (1,
                                       cdr (exp),
                                       va,
                                       fa,
                                       n,
                                       trace)),
good-frame3-proof (isin (z-list (list (10,
                                       ncode (0),
                                       ncode (get (sub1-n (car (exp),
                                                            11),
                                                   fa)),
                                       ncode (ev (1,
                                                  cdr (exp),
                                                  va,
                                                  fa,
                                                  n)),
                                       ncode (fa))),
                        trace),
                      a-ev-defn (trace),
                      f-equal (10,
                               ncode (ev (0,
                                          get (sub1-n (car (exp),
                                                       11),
                                               fa),
                                          ev (1,
                                              cdr (exp),
                                              va,
                                              fa,
                                              n),
                                          fa,
                                          n - 1))),
                      a-ev-good-proof (0,
                                       get (sub1-n (car (exp),
                                                    11),
                                            fa),
                                       ev (1,
                                           cdr (exp),
```

```
                                                                         va,
                                                                         fa,
                                                                         n),
                                                                       fa,
                                                                       n − 1,
                                                                       trace)))) endif
     else good-frame2-proof (isin (z-list (list (10,
                                                 ncode (flg),
                                                 ncode (exp),
                                                 ncode (va),
                                                 ncode (fa))),
                                          trace),
                                   a-ev-defn (trace),
                                   a-ev-defn-part (flg, exp, va, fa, 10, trace),
                                   f-equal (10, ncode (ev (flg, exp, va, fa, n))),
                                   a-ev-reduc-proof (flg, exp, va, fa, 10, trace),
                                   a-ev-defn-part-good-proof (flg,
                                                              exp,
                                                              va,
                                                              fa,
                                                              n,
                                                              10,
                                                              trace,
                                                              a-ev-defn (trace),
                                                              nil,
                                                              nil)) endif
```

```
;;;;;;;"A-EV" is ‘GOOD’, i.e.,
;;;;;;; |- "(A-ev(<FLG>, <EXP>, <VA>, <FA>, val, trace) -> (val = <(EV FLG VA FA N)>))".
```

THEOREM: a-ev-good-proof-proves

```
 (ev-hyps (given, defns, symbols)
  ∧   variable (trace)
  ∧   (trace ∉ list (6, 7, 8, 9, 10, 11, 12, 15, 16))
  ∧   (¬ btmp (ev (flg, exp, va, fa, n))))
 →    proves (a-ev-good-proof (flg, exp, va, fa, n, trace),
              f-implies (a-ev (ncode (flg),
                               ncode (exp),
                               ncode (va),
                               ncode (fa),
                               10,
                               trace),
                         f-equal (10, ncode (ev (flg, exp, va, fa, n)))),
              given,
```

defns,
symbols)

EVENT: Enable a-ev-good-proof; name this event 'g0954'.

```
;;;;;;;
;;;;;;;This brings us to an entirely different section of the proof.  So far we have
;;;;;;;shown that a Lisp interpreter is representable by a Z2 formula.  In order to
;;;;;;;construct the undecidable sentence, we need to show that the metatheory of Z2
;;;;;;;is representable in Z2.  We do that by showing that the metatheoretic formalization
;;;;;;;of Z2 is computable by means of the Lisp interpreter EV.  Since EV is only aware
;;;;;;;of data-structures which are lists or numbers (num-trees), we need to first
;;;;;;;represent the Z2 expressions and proofs in these terms.  The function GCODE
;;;;;;;translates the shell-constructors F-NOT, F-OR, FORSOME, CONS, and ADD1, into
;;;;;;;distinct num-trees.  If it is none of the above (NIL, for instance), then GCODE
;;;;;;;returns the num-tree (CONS 5 0).
;;;;;;;
```

DEFINITION:
gcode (x)
= **if** listp (x) **then** cons (1, cons (gcode (car (x)), gcode (cdr (x))))
elseif f-notp (x) **then** cons (2, cons (gcode (arg (x)), 0))
elseif f-orp (x) **then** cons (3, cons (gcode (arg1 (x)), gcode (arg2 (x))))
elseif forsomep (x) **then** cons (4, cons (bind (x), gcode (body (x))))
elseif $x \in \mathbf{N}$ **then** x
else cons (5, 0) **endif**

```
;;;;;;;The lemma below establishes the connection between EQL and GCODE.
```

THEOREM: eql-gcode-equal
eql (x, y) = (gcode (x) = gcode (y))

```
;;;;;;;Then we introduce functions which form e-expressions corresponding to e-function
;;;;;;;applications of the Lisp primitives.  The prefix PR stands for 'primitive'.
```

DEFINITION: pr-cons (x, y) = list (7, x, y)

DEFINITION: pr-car (x) = list (8, x)

DEFINITION: pr-cdr (x) = list (9, x)

DEFINITION: pr-equal (x, y) = list (1, x, y)

DEFINITION: pr-numberp (x) = list (2, x)

DEFINITION: pr-zerop (x) = list (3, x)

DEFINITION: pr-add1(x) = list$(\mathtt{4},\ x)$

DEFINITION: pr-sub1(x) = list$(\mathtt{5},\ x)$

DEFINITION: pr-listp(x) = list$(\mathtt{6},\ x)$

```
;;;;;;;The functions suffixed with 'N' are the Lisp analogues of the e-functions.
;;;;;;;CONS is not the Lisp analogue of PR-CONS since PR-CONS can return (BTM),
;;;;;;;whereas CONS cannot.  The function PR-CONSN returns (BTM) if one of its
;;;;;;;arguments is (BTM), and (CONS X Y) otherwise.  Similarly, we have the
;;;;;;;Lisp analogues for the other primitives.
```

DEFINITION:
pr-consn$(x,\ y)$
= **if** btmp$(x) \lor$ btmp(y) **then** BTM
else cons$(x,\ y)$ **endif**

DEFINITION:
pr-carn(x)
= **if** btmp(x) **then** BTM
else car(x) **endif**

DEFINITION:
pr-cdrn(x)
= **if** btmp(x) **then** BTM
else cdr(x) **endif**

DEFINITION:
pr-equaln$(x,\ y)$
= **if** btmp$(x) \lor$ btmp(y) **then** BTM
else bool-fix(ncode-equal$(x,\ y)$) **endif**

DEFINITION:
pr-numberpn(x)
= **if** btmp(x) **then** BTM
else bool-fix($\neg$ listp(x)) **endif**

DEFINITION:
pr-zeropn(x)
= **if** btmp(x) **then** BTM
else bool-fix$(x \simeq \mathtt{0})$ **endif**

DEFINITION:
pr-add1n(x)
= **if** btmp(x) **then** BTM
else $1 + x$ **endif**

DEFINITION:
pr-listpn (x)
$=$ **if** btmp (x) **then** BTM
else bool-fix (listp (x)) **endif**

EVENT: Disable subrp; name this event 'g0955'.

EVENT: Disable apply-subr; name this event 'g0956'.

EVENT: Disable apply-subr1; name this event 'g0957'.

EVENT: Disable apply-subr2; name this event 'g0958'.

```
;;;;;;;The relationship between PR-CONS and PR-CONSN is established below.  The lemma
;;;;;;;asserts that the result of evaluating (PR-CONS X Y) with EV is (PR-CONSN x y),
;;;;;;;where x and y are the results of evaluating X and Y, respectively.
```

THEOREM: pr-cons-eval
ev $(0,$ pr-cons $(x,\ y),\ va,\ fa,\ n) =$ pr-consn (ev $(0,\ x,\ va,\ fa,\ n)$, ev $(0,\ y,\ va,\ fa,\ n)$)

THEOREM: pr-car-eval
ev $(0,$ pr-car $(x),\ va,\ fa,\ n) =$ pr-carn (ev $(0,\ x,\ va,\ fa,\ n)$)

THEOREM: pr-cdr-eval
ev $(0,$ pr-cdr $(x),\ va,\ fa,\ n) =$ pr-cdrn (ev $(0,\ x,\ va,\ fa,\ n)$)

THEOREM: pr-equal-eval
ev $(0,$ pr-equal $(x,\ y),\ va,\ fa,\ n) =$ pr-equaln (ev $(0,\ x,\ va,\ fa,\ n)$, ev $(0,\ y,\ va,\ fa,\ n)$)

THEOREM: pr-numberp-eval
ev $(0,$ pr-numberp $(x),\ va,\ fa,\ n) =$ pr-numberpn (ev $(0,\ x,\ va,\ fa,\ n)$)

THEOREM: pr-zerop-eval
ev $(0,$ pr-zerop $(x),\ va,\ fa,\ n) =$ pr-zeropn (ev $(0,\ x,\ va,\ fa,\ n)$)

THEOREM: pr-add1-eval
ev $(0,$ pr-add1 $(x),\ va,\ fa,\ n) =$ pr-add1n (ev $(0,\ x,\ va,\ fa,\ n)$)

DEFINITION:
pr-sub1n (x)
$=$ **if** btmp (x) **then** BTM
else $x - 1$ **endif**

THEOREM: pr-sub1-eval
ev $(0,$ pr-sub1 $(x),\ va,\ fa,\ n) =$ pr-sub1n (ev $(0,\ x,\ va,\ fa,\ n)$)

THEOREM: pr-listp-eval
ev (0, pr-listp (x), va, fa, n) = pr-listpn (ev (0, x, va, fa, n))

EVENT: Enable pr-cons; name this event ‘g0959’.

EVENT: Enable pr-car; name this event ‘g0960’.

EVENT: Enable pr-cdr; name this event ‘g0961’.

EVENT: Enable pr-equal; name this event ‘g0962’.

EVENT: Enable pr-zerop; name this event ‘g0963’.

EVENT: Enable pr-numberp; name this event ‘g0964’.

EVENT: Enable pr-add1; name this event ‘g0965’.

EVENT: Enable pr-sub1; name this event ‘g0966’.

EVENT: Enable pr-listp; name this event ‘g0967’.

```
;;;;;;;Similarly we construct e-expressions for IF-expressions and QUOTE-expressions,
;;;;;;;and show how they are evaluated.
```

DEFINITION: pr-if (x, y, z) = list (0, x, y, z)

THEOREM: pr-if-eval
ev (0, pr-if (x, y, z), va, fa, n)
= **if** btmp (ev (0, x, va, fa, n)) **then** BTM
elseif ncode-equal (ev (0, x, va, fa, n), 0) **then** ev (0, z, va, fa, n)
else ev (0, y, va, fa, n) **endif**

EVENT: Enable pr-if; name this event ‘g0968’.

DEFINITION: pr-quote (x) = list (10, x)

THEOREM: pr-quote-eval
ev (0, pr-quote (x), va, fa, n) = x

```
;;;;;;;Having dealt with the primitives, we now get into the defined notions.
;;;;;;;There are two kinds of definitions: explicit and recursive.  The explicit
;;;;;;;definitions will be treated as abbreviations and no efunction symbol will
;;;;;;;introduced on their behalf.  E-function symbols will be introduced for
;;;;;;;recursively defined e-functions.  We first develop the basic logical connectives
;;;;;;;and analogues for the GCODEd versions of the shell-constructors/destructors/recognizers.
;;;;;;;G-CONS constructs the e-expression for the GCODE analogue of CONS, i.e., (G-CONS X Y)
;;;;;;;evaluates to (GCODE (CONS x y)).  We will adopt the convention of representing the
;;;;;;;value of X by x.
```

DEFINITION: g-cons (x, y) = pr-cons (pr-quote (1), pr-cons (x, y))

```
;;;;;;;The logical-and connective is defined below.  Note that PR-AND evaluates to 1 or 0.
```

DEFINITION:
pr-and (x, y) = pr-if (x, pr-if (y, pr-quote (1), pr-quote (0)), pr-quote (0))

DEFINITION:
pr-andn (x, y)
= **if** btmp (x) **then** BTM
elseif ncode-equal $(x, 0)$ **then** 0
elseif btmp (y) **then** BTM
elseif ncode-equal $(y, 0)$ **then** 0
else 1 **endif**

THEOREM: pr-and-eval
ev (0, pr-and (x, y), va, fa, n) = pr-andn (ev (0, x, va, fa, n), ev (0, y, va, fa, n))

EVENT: Enable pr-and; name this event ‘g0990’.

THEOREM: pr-andn-reduc
pr-andn (bool-fix (x), bool-fix (y)) = bool-fix $(x \land y)$

```
;;;;;;;The logical-not connective.
```

DEFINITION: pr-not (x) = pr-if (x, pr-quote (0), pr-quote (1))

DEFINITION:
pr-notn (x)
= **if** btmp (x) **then** BTM
elseif ncode-equal $(x, 0)$ **then** 1
else 0 **endif**

THEOREM: pr-not-eval
ev (0, pr-not (x), va, fa, n) = pr-notn (ev (0, x, va, fa, n))

EVENT: Enable pr-quote; name this event ‘g0991’.

EVENT: Enable pr-not; name this event ‘g0992’.

THEOREM: pr-notn-reduc
pr-notn (bool-fix (x)) = bool-fix ($\neg$ x)

```
;;;;;;;The logical-or connective.
```

DEFINITION:
pr-or (x, y) = pr-if (x, pr-quote (1), pr-if (y, pr-quote (1), pr-quote (0)))

DEFINITION:
pr-orn (x, y)
= **if** btmp (x) **then** BTM
elseif ncode-equal (x, 0)
then if btmp (y) **then** BTM
elseif ncode-equal (y, 0) **then** 0
else 1 **endif**
else 1 **endif**

THEOREM: pr-or-eval
ev (0, pr-or (x, y), va, fa, n) = pr-orn (ev (0, x, va, fa, n), ev (0, y, va, fa, n))

EVENT: Enable pr-or; name this event ‘g0993’.

THEOREM: pr-orn-reduc
pr-orn (bool-fix (x), bool-fix (y)) = bool-fix (x $\vee$ y)

```
;;;;;;;The next e-definition is an important one.  Given (GCODE X), we can recover
;;;;;;;the type of X by looking at (CAR X).  If it is 1, X is a list; if it is 2,
;;;;;;;X is of the form (F-NOT Y); if it is 3, X is of the form (F-OR Y Z); if it is
;;;;;;;4, X is of the form (FORSOME Y Z); otherwise it could either be a natural number
;;;;;;;for which we already have a NUMBERP primitive, or any remaining type which is
;;;;;;;coerced to NIL.  G-TYPEP checks if X is of type TYPE.  Note that TYPE is going
;;;;;;;to be a number, not an e-expression, since it appears withing PR-QUOTE in G-TYPEP.
```

DEFINITION:
g-typep (x, $type$) = pr-equal (pr-car (x), pr-quote ($type$))

DEFINITION: g-typepn (x, $type$) = pr-equaln (pr-carn (x), $type$)

THEOREM: g-typep-eval
ev (0, g-typep (x, $type$), va, fa, n) = g-typepn (ev (0, x, va, fa, n), $type$)

EVENT: Enable g-typep; name this event ‘g0995’.

```
;;;;;;;PR-IFN is the Lisp analogue of PR-IF.
```

DEFINITION:
pr-ifn (x, y, z)
= **if** btmp (x) **then** BTM
elseif ncode-equal (x, 0) **then** z
else y **endif**

```
;;;;;;;PR-CAR-CDR is the iterated CAR/CDR e-expression.
```

DEFINITION:
pr-car-cdr (x, *list*)
= **if** listp (*list*)
then if car (*list*) = 1 **then** pr-car (pr-car-cdr (x, cdr (*list*)))
else pr-cdr (pr-car-cdr (x, cdr (*list*))) **endif**
else x **endif**

```
;;;;;;;The destructors for GCODEd objects are derived from G-ARG below.
;;;;;;;Let us represent (GCODE X) as \X\.
;;;;;;;G-ARG selects a field from \X\ according to its type TYPE and the
;;;;;;;desired field ARGNUM.  E.g., (G-ARG Y (PR-QUOTE 1) 1) evaluates to
;;;;;;;\(CAR X)\, if Y evaluates to \X\.
```

DEFINITION:
g-arg (x, *argnum*, *type*)
= pr-if (g-typep (x, *type*),
pr-if (*argnum*, pr-car-cdr (x, list (2, 2)), pr-car-cdr (x, list (1, 2))),
pr-quote (0))

EVENT: Enable g-typepn; name this event ‘g0997’.

EVENT: Enable pr-carn; name this event ‘g0998’.

EVENT: Enable pr-cdrn; name this event ‘g0999’.

DEFINITION:
g-argn (x, *argnum*, *type*)
= pr-ifn (g-typepn (x, *type*),
pr-ifn (*argnum*, pr-cdrn (pr-cdrn (x)), pr-carn (pr-cdrn (x))),
0)

THEOREM: g-arg-eval
ev (0, g-arg (x, *argnum*, *type*), *va*, *fa*, n)
= g-argn (ev (0, x, *va*, *fa*, n), ev (0, *argnum*, *va*, *fa*, n), *type*)

EVENT: Enable g-arg; name this event 'g1000'.

EVENT: Disable g-typepn; name this event 'g1001'.

EVENT: Disable pr-carn; name this event 'g1003'.

THEOREM: pr-carn-gcode
pr-carn (gcode (x)) = car (gcode (x))

EVENT: Disable pr-cdrn; name this event 'g1004'.

THEOREM: pr-cdrn-gcode
pr-cdrn (gcode (x)) = cdr (gcode (x))

```
;;;;;;;G-TYPEPN on Lists, negations, disjunctions, and existential quantifications.
```

THEOREM: g-typepn-gcode-list
g-typepn (gcode (x), 1) = bool-fix (listp (x))

THEOREM: g-typepn-gcode-f-notp
g-typepn (gcode (x), 2) = bool-fix (f-notp (x))

THEOREM: g-typepn-gcode-f-orp
g-typepn (gcode (x), 3) = bool-fix (f-orp (x))

THEOREM: g-typepn-gcode-forsomep
g-typepn (gcode (x), 4) = bool-fix (forsomep (x))

THEOREM: car-gcode
g-argn (gcode (x), 0, 1) = gcode (car (x))

THEOREM: cdr-gcode
g-argn (gcode (x), 1, 1) = gcode (cdr (x))

THEOREM: arg-gcode
g-argn (gcode (x), 0, 2) = gcode (arg (x))

THEOREM: arg1-gcode
g-argn (gcode (x), 0, 3) = gcode (arg1 (x))

THEOREM: arg2-gcode
g-argn (gcode (x), 1, 3) = gcode (arg2 (x))

THEOREM: bind-gcode
g-argn (gcode (x), 0, 4) = gcode (bind (x))

THEOREM: body-gcode
g-argn (gcode (x), 1, 4) = gcode (body (x))

```
;;;;;;;From G-ARG, we derive the selectors (destructors) for the GCODEd objects.
```

DEFINITION: g-car (x) = g-arg (x, pr-quote (0), 1)

DEFINITION: g-cdr (x) = g-arg (x, pr-quote (1), 1)

DEFINITION: g-carn (x) = g-argn (x, 0, 1)

```
;;;;;;;G-FUNCTION is the EV analogue of the metatheoretic function FUNCTION.
```

DEFINITION: g-function (fn) = pr-equal (g-car (fn), pr-quote (2))

DEFINITION: g-functionn (fn) = pr-equaln (g-carn (fn), 2)

THEOREM: eval-g-function
ev (0, g-function (fn), va, fa, n) = g-functionn (ev (0, fn, va, fa, n))

EVENT: Disable function; name this event ‘g0969’.

THEOREM: g-functionn-gcode
g-functionn (gcode (fn)) = bool-fix (function (fn))

EVENT: Enable function; name this event ‘g0970’.

EVENT: Enable g-function; name this event ‘g0971’.

EVENT: Enable g-functionn; name this event ‘g0972’.

```
;;;;;;;G-VARIABLE is the EV analogue of VARIABLE.
```

DEFINITION: g-variable (x) = pr-numberp (x)

DEFINITION: g-variablen (x) = pr-numberpn (x)

THEOREM: eval-g-variable
ev (0, g-variable (x), va, fa, n) = g-variablen (ev (0, x, va, fa, n))

Theorem: g-variablen-gcode
g-variablen (gcode (x)) = bool-fix (variable (x))

Event: Enable g-variable; name this event ‘g0973’.

Event: Enable g-variablen; name this event ‘g0976’.

Event: Enable pr-consn; name this event ‘g0977’.

Definition: g-consn (x, y) = pr-consn (1, pr-consn (x, y))

Theorem: g-cons-eval
ev (0, g-cons (x, y), va, fa, n) = g-consn (ev (0, x, va, fa, n), ev (0, y, va, fa, n))

Event: Disable pr-consn; name this event ‘g0978’.

Theorem: g-consn-gcode
g-consn (gcode (x), gcode (y)) = gcode (cons (x, y))

Event: Enable pr-consn; name this event ‘g0979’.

Event: Enable g-consn; name this event ‘g0980’.

Event: Enable g-cons; name this event ‘g0981’.

;;;;;;;G-FIX is the EV analogue of FIX.

Definition: g-fix (x) = pr-if (pr-numberp (x), x, pr-quote (0))

Definition: g-fixn (x) = pr-ifn (pr-numberpn (x), x, 0)

Theorem: eval-g-fix
ev (0, g-fix (x), va, fa, n) = g-fixn (ev (0, x, va, fa, n))

Theorem: g-fix-gcode
g-fixn (gcode (x)) = gcode (fix (x))

Event: Enable g-fix; name this event ‘g0982’.

Event: Enable g-fixn; name this event ‘g0983’.

DEFINITION:
g-f-not (x) = pr-cons (pr-quote (2), pr-cons $(x$, pr-quote (0)))

DEFINITION: g-f-notn (x) = pr-consn (2, pr-consn $(x, 0)$)

THEOREM: eval-g-f-not
ev (0, g-f-not (x), va, fa, n) = g-f-notn (ev (0, x, va, fa, n))

EVENT: Disable pr-consn; name this event 'g0984'.

THEOREM: g-f-notn-gcode
g-f-notn (gcode (x)) = gcode (f-not (x))

EVENT: Enable g-f-not; name this event 'g0985'.

EVENT: Enable g-f-notn; name this event 'g0986'.

DEFINITION: g-f-or (x, y) = pr-cons (pr-quote (3), pr-cons (x, y))

DEFINITION: g-f-orn (x, y) = pr-consn (3, pr-consn (x, y))

THEOREM: eval-g-f-or
ev (0, g-f-or (x, y), va, fa, n) = g-f-orn (ev (0, x, va, fa, n), ev (0, y, va, fa, n))

THEOREM: g-f-orn-gcode
g-f-orn (gcode (x), gcode (y)) = gcode (f-or (x, y))

EVENT: Enable g-f-or; name this event 'g0987'.

EVENT: Enable g-f-orn; name this event 'g0988'.

DEFINITION:
g-forsome (x, y) = pr-cons (pr-quote (4), pr-cons (g-fix (x), y))

DEFINITION:
g-forsomen (x, y) = pr-consn (4, pr-consn (g-fixn (x), y))

THEOREM: eval-g-forsome
ev (0, g-forsome (x, y), va, fa, n)
= g-forsomen (ev (0, x, va, fa, n), ev (0, y, va, fa, n))

THEOREM: g-forsomen-gcode
g-forsomen (gcode (x), gcode (y)) = gcode (forsome (x, y))

EVENT: Enable g-forsome; name this event 'g0989'.

EVENT: Enable g-forsomen; name this event 'g0994'.

```
;;;;;;;G-PREDICATE is the EV analogue of PREDICATE.  In general G-FOO is the
;;;;;;;EV analogue of FOO.
```

DEFINITION: g-predicate (x) = pr-equal (g-car (x), pr-quote (3))

DEFINITION: g-predicaten (x) = pr-equaln (g-carn (x), 3)

THEOREM: eval-g-predicate
ev (0, g-predicate (x), va, fa, n) = g-predicaten (ev (0, x, va, fa, n))

EVENT: Disable predicate; name this event 'g0996'.

THEOREM: g-predicaten-gcode
g-predicaten (gcode (x)) = bool-fix (predicate (x))

EVENT: Enable predicate; name this event 'g1002'.

EVENT: Enable g-predicate; name this event 'g1005'.

EVENT: Enable g-predicaten; name this event 'g1006'.

DEFINITION:
g-func-pred (x) = pr-or (g-function (x), g-predicate (x))

DEFINITION:
g-func-predn (x) = pr-orn (g-functionn (x), g-predicaten (x))

THEOREM: eval-g-func-pred
ev (0, g-func-pred (x), va, fa, n) = g-func-predn (ev (0, x, va, fa, n))

THEOREM: g-func-predn-gcode
g-func-predn (gcode (x)) = bool-fix (func-pred (x))

EVENT: Enable g-func-pred; name this event 'g1007'.

EVENT: Enable g-func-predn; name this event 'g1008'.

```
;;;;;;;Now we start defining the EV analogues of the recursive metatheoretic functions.
;;;;;;;These will all be defined first, and then shown to correspond to their
;;;;;;;Boyer-Moore counterparts.
```

DEFINITION: g-append (x, y) = list $(\mathtt{11}, x, y)$

DEFINITION: g-cdrn (x) = g-argn $(x, \mathtt{1}, \mathtt{1})$

EVENT: Enable g-argn; name this event 'g1009'.

EVENT: Enable g-typepn; name this event 'g1010'.

THEOREM: eval-pr-if-again
ev $(\mathtt{0}$, pr-if (x, y, z), va, fa, $n)$
= pr-ifn (ev $(\mathtt{0}, x, va, fa, n)$, ev $(\mathtt{0}, y, va, fa, n)$, ev $(\mathtt{0}, z, va, fa, n)$)

THEOREM: pr-ifn-gcode
pr-ifn (bool-fix (x), y, z)
= **if** x **then** y
else z **endif**

EVENT: Enable pr-ifn; name this event 'g1011'.

THEOREM: eval-var
$(x \in \mathbf{N}) \rightarrow$ (ev $(\mathtt{0}, x, va, fa, n)$ = get (x, va))

THEOREM: eval-fun-zero
$((\mathtt{10} < fn) \wedge (n \simeq \mathtt{0})) \rightarrow$ (ev $(\mathtt{0}$, cons $(fn, args)$, va, fa, $n)$ = BTM)

THEOREM: eval-fun1
$((flg \neq \mathtt{1}) \wedge$ listp $(exp) \wedge (\mathtt{10} <$ car $(exp)))$
$\rightarrow$ (ev (flg, exp, va, fa, n)
= **if** btmp (ev $(\mathtt{1}$, cdr (exp), va, fa, $n)$) **then** BTM
elseif $n \simeq \mathtt{0}$ **then** BTM
else ev $(\mathtt{0}$,
get (sub1-n (car (exp), $\mathtt{11}$), fa),
ev $(\mathtt{1}$, cdr (exp), va, fa, $n)$,
fa,
$n - 1)$ **endif**)

THEOREM: eval-fun2
$(\mathtt{10} < fn)$
$\rightarrow$ (ev $(\mathtt{0}$, cons $(fn, args)$, va, fa, $n)$
= **if** btmp (ev $(\mathtt{1}, args, va, fa, n)$) **then** BTM

$\quad\quad$ **elseif** $n \simeq 0$ **then** BTM
$\quad\quad$ **else** ev (0,
$\quad\quad\quad\quad$ get (sub1-n (*fn*, 11), *fa*),
$\quad\quad\quad\quad$ ev (1, *args*, *va*, *fa*, *n*),
$\quad\quad\quad\quad$ *fa*,
$\quad\quad\quad\quad$ $n - 1$) **endif**)

THEOREM: ev-list1
ev (1, cons (x, y), *va*, *fa*, n)
$=$ **if** btmp (ev (0, x, *va*, *fa*, n)) $\vee$ btmp (ev (1, y, *va*, *fa*, n)) **then** BTM
$\quad$ **else** cons (ev (0, x, *va*, *fa*, n), ev (1, y, *va*, *fa*, n)) **endif**

THEOREM: ev-list2
ev (1, **nil**, *va*, *fa*, n) $=$ **nil**

EVENT: Enable ev; name this event ‘g1012’.

EVENT: Disable sub1; name this event ‘g1013’.

EVENT: Disable get; name this event ‘g1014’.

EVENT: Enable g-typepn; name this event ‘g1015’.

EVENT: Enable eval-fun1; name this event ‘g1016’.

THEOREM: eval-fun3
$((10 < fn) \wedge (n \not\simeq 0))$
$\rightarrow$ (ev (0, cons (*fn*, *args*), *va*, *fa*, n)
$\quad\quad = $ **if** btmp (ev (1, *args*, *va*, *fa*, n)) **then** BTM
$\quad\quad\quad$ **else** ev (0,
$\quad\quad\quad\quad$ get (sub1-n (*fn*, 11), *fa*),
$\quad\quad\quad\quad$ ev (1, *args*, *va*, *fa*, n),
$\quad\quad\quad\quad$ *fa*,
$\quad\quad\quad\quad$ $n - 1$) **endif**)

```
;;;;;;;The e-definition of G-APPEND.
```

DEFINITION:
G-APPEND-DEFN
$=$ pr-if (g-typep (0, 1), g-cons (g-car (0), g-append (g-cdr (0), 1)), 1)

EVENT: Enable g-append; name this event ‘g1017’.

```
;;;;;;;The e-definition of G-DEL.
```

DEFINITION: g-del (x, y) = list (12, x, y)

DEFINITION:
G-DEL-DEFN
= pr-if (g-typep (1, 1),
  pr-if (pr-equal (0, g-car (1)),
    g-del (0, g-cdr (1)),
    g-cons (g-car (1), g-del (0, g-cdr (1)))),
  1)

EVENT: Enable g-del; name this event ‘g1018’.

```
;;;;;;;The e-definition of G-MEMB.
```

DEFINITION: g-memb (x, y) = list (13, x, y)

DEFINITION:
G-MEMB-DEFN
= pr-if (g-typep (1, 1),
  pr-if (pr-equal (0, g-car (1)), pr-quote (1), g-memb (0, g-cdr (1))),
  pr-quote (0))

EVENT: Enable g-memb; name this event ‘g1019’.

```
;;;;;;;The e-definition of G-COLLECT-FREE.
```

DEFINITION: g-collect-free (*exp*, *flg*) = list (14, *exp*, *flg*)

DEFINITION:
G-COLLECT-FREE-DEFN
= pr-if (pr-zerop (1),
  pr-if (g-variable (0),
    g-cons (0, pr-quote (gcode (**nil**))),
    pr-if (g-typep (0, 2),
      g-collect-free (g-arg (0, pr-quote (0), 2), pr-quote (0)),
      pr-if (g-typep (0, 3),
        g-append (g-collect-free (g-arg (0,
              pr-quote (0),
              3),
            pr-quote (0)),
          g-collect-free (g-arg (0,
              pr-quote (1),

```
                                                      3),
                                                 pr-quote (0))),
                         pr-if (g-typep (0, 4),
                             g-del (g-arg (0, pr-quote (0), 4),
                                  g-collect-free (g-arg (0,
                                                       pr-quote (1),
                                                       4),
                                                  pr-quote (0))),
                             pr-if (g-typep (0, 1),
                                 g-collect-free (g-cdr (0),
                                                 pr-quote (1)),
                                 pr-quote (gcode (nil))))))),
             pr-if (g-typep (0, 1),
                 g-append (g-collect-free (g-car (0), pr-quote (0)),
                         g-collect-free (g-cdr (0), pr-quote (1))),
                 pr-quote (gcode (nil))))
```

EVENT: Enable g-collect-free; name this event ‘g1020’.

```
;;;;;;;The e-definition of G-COVERING.
```

DEFINITION: g-covering (*exp*, *var*, *flg*) = list (15, *exp*, *var*, *flg*)

DEFINITION:
G-COVERING-DEFN

```
=   pr-if (pr-zerop (2),
        pr-if (g-typep (0, 2),
            g-covering (g-arg (0, pr-quote (0), 2), 1, pr-quote (0)),
            pr-if (g-typep (0, 3),
                g-append (g-covering (g-arg (0, pr-quote (0), 3),
                                      1,
                                      pr-quote (0)),
                        g-covering (g-arg (0, pr-quote (1), 3),
                                    1,
                                    pr-quote (0))),
                pr-if (g-typep (0, 4),
                    pr-if (g-memb (1, g-collect-free (0, pr-quote (0))),
                        g-cons (g-arg (0, pr-quote (0), 4),
                              g-covering (g-arg (0,
                                                 pr-quote (1),
                                                 4),
                                          1,
                                          pr-quote (0))),
                        pr-quote (gcode (nil))),
```

$$
\begin{array}{l}
\quad\quad\quad\quad\quad\quad \text{pr-if}\,(\text{g-typep}\,(\mathtt{0},\ \mathtt{1}), \\
\quad\quad\quad\quad\quad\quad\quad\quad \text{g-covering}\,(\text{g-cdr}\,(\mathtt{0}),\ \mathtt{1},\ \text{pr-quote}\,(\mathtt{1})), \\
\quad\quad\quad\quad\quad\quad\quad\quad \text{pr-quote}\,(\text{gcode}\,(\mathbf{nil}))))))), \\
\quad\quad \text{pr-if}\,(\text{g-typep}\,(\mathtt{0},\ \mathtt{1}), \\
\quad\quad\quad\quad \text{g-append}\,(\text{g-covering}\,(\text{g-car}\,(\mathtt{0}),\ \mathtt{1},\ \text{pr-quote}\,(\mathtt{0})), \\
\quad\quad\quad\quad\quad\quad\quad\quad \text{g-covering}\,(\text{g-cdr}\,(\mathtt{0}),\ \mathtt{1},\ \text{pr-quote}\,(\mathtt{1}))), \\
\quad\quad\quad\quad \text{pr-quote}\,(\text{gcode}\,(\mathbf{nil}))))
\end{array}
$$

EVENT: Enable g-covering; name this event ‘g1021’.

```
;;;;;;;The e-definition of G-NIL-INTERSECT.
```

DEFINITION: g-nil-intersect $(x,\ y) = \text{list}\,(\mathtt{16},\ x,\ y)$

EVENT: Enable bool-fix; name this event ‘g1022’.

DEFINITION:
G-NIL-INTERSECT-DEFN

$$
\begin{array}{ll}
= & \text{pr-if}\,(\text{g-typep}\,(\mathtt{0},\ \mathtt{1}), \\
& \quad\quad \text{pr-and}\,(\text{pr-not}\,(\text{g-memb}\,(\text{g-car}\,(\mathtt{0}),\ \mathtt{1})),\ \text{g-nil-intersect}\,(\text{g-cdr}\,(\mathtt{0}),\ \mathtt{1})), \\
& \quad\quad \text{pr-quote}\,(\text{bool-fix}\,(\mathtt{t})))
\end{array}
$$

EVENT: Enable g-nil-intersect; name this event ‘g1023’.

```
;;;;;;;The e-definition of G-VAR-LIST.
```

DEFINITION: g-var-list $(\mathit{list},\ \mathit{len}) = \text{list}\,(\mathtt{17},\ \mathit{list},\ \mathit{len})$

THEOREM: pr-sub1n-gcode
$\text{pr-sub1n}\,(\text{gcode}\,(x)) = \text{gcode}\,(x - 1)$

EVENT: Enable pr-sub1n; name this event ‘g1024’.

DEFINITION:
G-VAR-LIST-DEFN

$$
\begin{array}{ll}
= & \text{pr-if}\,(\text{pr-or}\,(\text{pr-zerop}\,(\mathtt{1}),\ \text{pr-not}\,(\text{g-typep}\,(\mathtt{0},\ \mathtt{1}))), \\
& \quad\quad \text{pr-and}\,(\text{pr-zerop}\,(\mathtt{1}),\ \text{pr-not}\,(\text{g-typep}\,(\mathtt{0},\ \mathtt{1}))), \\
& \quad\quad \text{pr-and}\,(\text{g-variable}\,(\text{g-car}\,(\mathtt{0})),\ \text{g-var-list}\,(\text{g-cdr}\,(\mathtt{0}),\ \text{pr-sub1}\,(\mathtt{1}))))
\end{array}
$$

EVENT: Enable g-var-list; name this event ‘g1025’.

```
;;;;;;;The e-definition of G-TERMP.
```

DEFINITION:
g-termp (*exp*, *flg*, *symbols*) = list (19, *exp*, *flg*, *symbols*)

EVENT: Enable pr-equaln; name this event 'g1026'.

DEFINITION:
g-symb (x, *symbols*)
= pr-or (pr-equal (x,
            g-cons (pr-quote (gcode (3)),
                g-cons (pr-quote (gcode (0)), pr-quote (gcode (2))))),
    g-memb (x, *symbols*))

EVENT: Enable g-symb; name this event 'g1027'.

DEFINITION: g-length (x) = list (18, x)

DEFINITION:
G-LENGTH-DEFN
= pr-if (g-typep (0, 1), pr-add1 (g-length (g-cdr (0))), pr-quote (gcode (0)))

DEFINITION:
G-TERMP-DEFN
= pr-if (pr-zerop (1),
    pr-if (g-typep (0, 1),
        pr-and (g-function (g-car (0)),
            pr-and (g-symb (g-car (0), 2),
                pr-and (g-termp (g-cdr (0), pr-quote (1), 2),
                    pr-equal (g-length (g-cdr (0)),
                        g-fix (g-cdr (g-cdr (g-car (0)))))))),
        g-variable (0)),
    pr-if (g-typep (0, 1),
        pr-and (g-termp (g-car (0), pr-quote (0), 2),
            g-termp (g-cdr (0), pr-quote (1), 2)),
        pr-quote (bool-fix (t))))

```
;;;;;;;The e-definition of G-SUBST.
```

DEFINITION:
g-subst (*exp*, *var*, *term*, *flg*) = list (20, *exp*, *var*, *term*, *flg*)

DEFINITION:
G-SUBST-DEFN
= pr-if (pr-zerop (3),
    pr-if (g-variable (0),

```
        pr-if (pr-equal (0, 1), 2, 0),
        pr-if (g-typep (0, 2),
              g-f-not (g-subst (g-arg (0, pr-quote (0), 2),
                                1,
                                2,
                                pr-quote (gcode (0)))),
              pr-if (g-typep (0, 3),
                    g-f-or (g-subst (g-arg (0, pr-quote (0), 3),
                                     1,
                                     2,
                                     pr-quote (gcode (0))),
                            g-subst (g-arg (0, pr-quote (1), 3),
                                     1,
                                     2,
                                     pr-quote (gcode (0)))),
                    pr-if (g-typep (0, 4),
                          pr-if (pr-equal (g-arg (0, pr-quote (0), 4),
                                           1),
                                0,
                                g-forsome (g-arg (0, pr-quote (0), 4),
                                           g-subst (g-arg (0,
                                                           pr-quote (1),
                                                           4),
                                                    1,
                                                    2,
                                                    pr-quote (gcode (0))))),
                          pr-if (g-typep (0, 1),
                                g-cons (g-car (0),
                                        g-subst (g-cdr (0),
                                                 1,
                                                 2,
                                                 pr-quote (gcode (1)))),
                                0))))),
      pr-if (g-typep (0, 1),
            g-cons (g-subst (g-car (0), 1, 2, pr-quote (gcode (0))),
                    g-subst (g-cdr (0), 1, 2, pr-quote (gcode (1)))),
            0))
```

DEFINITION: g-sub $(x,\ y)$ = list (21, x, y)

DEFINITION:
G-SUB-DEFN

```
=   pr-if (g-typep (0, 1),
          pr-and (g-memb (g-car (0), 1), g-sub (g-cdr (0), 1)),
          pr-quote (bool-fix (t)))
```

`;;;;;;;The e-definition of G-FORMULA.`

DEFINITION: g-formula (*exp*, *symbols*) = list (22, *exp*, *symbols*)

DEFINITION:
g-atomp (*exp*, *symbols*)
= pr-and (g-predicate (g-car (*exp*)),
  pr-and (g-symb (g-car (*exp*), *symbols*),
   pr-and (pr-equal (g-length (g-cdr (*exp*)),
     g-fix (g-cdr (g-cdr (g-car (*exp*))))),
    g-termp (g-cdr (*exp*), pr-quote (1), *symbols*))))

EVENT: Enable g-atomp; name this event ‘g1028’.

DEFINITION:
G-FORMULA-DEFN
= pr-if (g-typep (0, 2),
  g-formula (g-arg (0, pr-quote (0), 2), 1),
  pr-if (g-typep (0, 3),
   pr-and (g-formula (g-arg (0, pr-quote (0), 3), 1),
    g-formula (g-arg (0, pr-quote (1), 3), 1)),
   pr-if (g-typep (0, 4),
    g-formula (g-arg (0, pr-quote (1), 4), 1),
    g-atomp (0, 1))))

DEFINITION:
g-pairequals (*vars1*, *vars2*, *exp*) = list (23, *vars1*, *vars2*, *exp*)

DEFINITION:
g-f-and (x, y) = g-f-not (g-f-or (g-f-not (x), g-f-not (y)))

DEFINITION:
g-f-andn (x, y) = g-f-notn (g-f-orn (g-f-notn (x), g-f-notn (y)))

THEOREM: g-f-and-eval
ev (0, g-f-and (x, y), *va*, *fa*, n) = g-f-andn (ev (0, x, *va*, *fa*, n), ev (0, y, *va*, *fa*, n))

EVENT: Disable f-and; name this event ‘g1098’.

THEOREM: g-f-andn-gcode
g-f-andn (gcode (x), gcode (y)) = gcode (f-and (x, y))

EVENT: Enable f-and; name this event ‘g1099’.

EVENT: Enable g-f-and; name this event ‘g1100’.

EVENT: Enable g-f-andn; name this event 'g1101'.

DEFINITION: g-f-implies $(x,\ y)$ = g-f-or (g-f-not $(x),\ y)$

DEFINITION: g-f-impliesn $(x,\ y)$ = g-f-orn (g-f-notn $(x),\ y)$

THEOREM: g-f-implies-eval
ev (0, g-f-implies $(x,\ y),\ va,\ fa,\ n)$
= g-f-impliesn (ev (0, $x,\ va,\ fa,\ n)$, ev (0, $y,\ va,\ fa,\ n)$)

EVENT: Disable f-implies; name this event 'g1102'.

THEOREM: g-f-impliesn-gcode
g-f-impliesn (gcode (x), gcode (y)) = gcode (f-implies $(x,\ y)$)

EVENT: Enable f-implies; name this event 'g1103'.

EVENT: Enable g-f-implies; name this event 'g1104'.

EVENT: Enable g-f-impliesn; name this event 'g1105'.

DEFINITION:
g-forall $(var,\ exp)$ = g-f-not (g-forsome $(var$, g-f-not (exp)))

DEFINITION:
g-foralln $(var,\ exp)$ = g-f-notn (g-forsomen $(var$, g-f-notn (exp)))

THEOREM: g-forall-eval
ev (0, g-forall $(x,\ y),\ va,\ fa,\ n)$ = g-foralln (ev (0, $x,\ va,\ fa,\ n)$, ev (0, $y,\ va,\ fa,\ n)$)

EVENT: Disable forall; name this event 'g1106'.

THEOREM: g-foralln-gcode
g-foralln (gcode (x), gcode (y)) = gcode $(\forall\ x\ y)$

EVENT: Enable forall; name this event 'g1107'.

EVENT: Enable g-forall; name this event 'g1108'.

EVENT: Enable g-foralln; name this event 'g1109'.

DEFINITION:
g-f-equal (x, y)
$=$ g-cons (g-cons (pr-quote (gcode (3)),
g-cons (pr-quote (gcode (0)), pr-quote (gcode (2)))),
g-cons (x, g-cons (y, pr-quote (gcode (**nil**)))))

DEFINITION:
g-f-equaln (x, y)
$=$ g-consn (g-consn (gcode (3), g-consn (gcode (0), gcode (2))),
g-consn (x, g-consn (y, gcode (**nil**))))

THEOREM: g-f-equal-eval
ev (0, g-f-equal (x, y), va, fa, n)
$=$ g-f-equaln (ev (0, x, va, fa, n), ev (0, y, va, fa, n))

EVENT: Disable p; name this event 'g1049'.

EVENT: Disable f-equal; name this event 'g1050'.

THEOREM: g-f-equaln-gcode
g-f-equaln (gcode (x), gcode (y)) $=$ gcode (f-equal (x, y))

EVENT: Enable p; name this event 'g1051'.

EVENT: Enable f-equal; name this event 'g1052'.

EVENT: Enable g-f-equal; name this event 'g1053'.

EVENT: Enable g-f-equaln; name this event 'g1054'.

```
;;;;;;;The e-definition of G-PAIREQUALS.
```

DEFINITION:
G-PAIREQUALS-DEFN
$=$ pr-if (pr-and (g-typep (0, 1), g-typep (1, 1)),
g-f-implies (g-f-equal (g-car (0), g-car (1)),
g-pairequals (g-cdr (0), g-cdr (1), 2)),
2)

```
;;;;;;;The e-definition of G-SET.
```

DEFINITION: g-set (x) $=$ list (24, x)

DEFINITION:
G-SET-DEFN
$=$ pr-if (g-typep (0, 1),
pr-and (pr-not (g-memb (g-car (0), g-cdr (0))), g-set (g-cdr (0))),
pr-quote (bool-fix (t)))

DEFINITION: g-conc (pf, flg) = list (25, pf, flg)

DEFINITION: g-cadr (x) = g-car (g-cdr (x))

DEFINITION: g-caddr (x) = g-car (g-cdr (g-cdr (x)))

DEFINITION: g-cadddr (x) = g-car (g-cdr (g-cdr (g-cdr (x))))

`;;;;;;;The e-definition of G-CONC.`

DEFINITION:
G-CONC-DEFN
$=$ pr-if (pr-zerop (1),
g-caddr (0),
pr-if (g-typep (0, 1),
g-cons (g-conc (g-car (0), pr-quote (0)),
g-conc (g-cdr (0), pr-quote (1))),
pr-quote (gcode (**nil**))))

`;;;;;;;The e-definition of G-REST-OF.`

DEFINITION: g-rest-of (sym, $symbols$) = list (26, sym, $symbols$)

DEFINITION:
G-REST-OF-DEFN
$=$ pr-if (g-typep (1, 1),
pr-if (pr-and (pr-equal (g-car (1), 0), pr-not (g-memb (0, g-cdr (1)))),
g-cdr (1),
g-rest-of (0, g-cdr (1))),
pr-quote (gcode (**nil**)))

`;;;;;;;The e-definition of G-ASSOC.`

DEFINITION: g-assoc (x, y) = list (27, x, y)

DEFINITION:
G-ASSOC-DEFN
$=$ pr-if (g-typep (1, 1),
pr-if (pr-equal (g-car (g-car (1)), 0),
g-cdr (g-car (1)),
g-assoc (0, g-cdr (1))),
pr-quote (0))

Definition:
g-prf (*pf* , *given*, *defns*, *flg*, *symbols*) = list (28, *pf* , *given*, *defns*, *flg*, *symbols*)

Definition:
g-free-for (*exp*, *var*, *term*, *flg*)
= g-nil-intersect (g-covering (*exp*, *var*, *flg*),
g-collect-free (*term*, pr-quote (0)))

Event: Enable g-free-for; name this event ‘g1029’.

```
;;;;;;;The EV analogues of the FOL axiom-checkers.
```

Definition:
g-subst-axiom-proof (*conc*, *hints*, *symbols*)
= pr-and (g-formula (g-cadr (*hints*), *symbols*),
pr-and (g-variable (g-caddr (*hints*)),
pr-and (g-termp (g-cadddr (*hints*), pr-quote (0), *symbols*),
pr-and (g-free-for (g-cadr (*hints*),
g-caddr (*hints*),
g-cadddr (*hints*),
pr-quote (0)),
pr-equal (*conc*,
g-f-implies (g-subst (g-cadr (*hints*),
g-caddr (*hints*),
g-cadddr (*hints*),
pr-quote (0)),
g-forsome (g-caddr (*hints*),
g-cadr (*hints*))))))))

Definition: g-degree (x) = g-fix (g-cdr (g-cdr (x)))

Definition:
g-fun-equal-axiom-proof (*conc*, *hints*, *symbols*)
= pr-and (g-termp (g-caddr (*hints*), pr-quote (1), *symbols*),
pr-and (g-termp (g-cadddr (*hints*), pr-quote (1), *symbols*),
pr-and (g-function (g-cadr (*hints*)),
pr-and (pr-equal (g-length (g-caddr (*hints*)),
g-degree (g-cadr (*hints*))),
pr-and (pr-equal (g-length (g-cadddr (*hints*)),
g-degree (g-cadr (*hints*))),
pr-equal (*conc*,
g-pairequals (g-caddr (*hints*),
g-cadddr (*hints*),
g-f-equal (g-cons (g-cadr (*hints*),

g-caddr (*hints*)),
g-cons (g-cadr (*hints*),
g-cadddr (*hints*)))))))))

DEFINITION:
g-pred-equal-axiom-proof (*conc*, *hints*, *symbols*)
= pr-and (g-termp (g-caddr (*hints*), pr-quote (1), *symbols*),
pr-and (g-termp (g-cadddr (*hints*), pr-quote (1), *symbols*),
pr-and (g-predicate (g-cadr (*hints*)),
pr-and (pr-equal (g-length (g-caddr (*hints*)),
g-degree (g-cadr (*hints*))),
pr-and (pr-equal (g-length (g-cadddr (*hints*)),
g-degree (g-cadr (*hints*))),
pr-equal (*conc*,
g-pairequals (g-caddr (*hints*),
g-cadddr (*hints*),
g-f-implies (g-cons (g-cadr (*hints*),
g-caddr (*hints*)),
g-cons (g-cadr (*hints*),
g-cadddr (*hints*))))))))))

```
;;;;;;;The EV analogue of FOL-AXIOM-PROOF.
```

DEFINITION:
g-fol-axiom-proof (*conc*, *hints*, *symbols*)
= pr-if (pr-equal (g-car (*hints*), pr-quote (1)),
pr-and (g-formula (g-cadr (*hints*), *symbols*),
pr-equal (*conc*,
g-f-or (g-f-not (g-cadr (*hints*)), g-cadr (*hints*)))),
pr-if (pr-equal (g-car (*hints*), pr-quote (2)),
g-subst-axiom-proof (*conc*, *hints*, *symbols*),
pr-if (pr-equal (g-car (*hints*), pr-quote (3)),
pr-and (g-termp (g-cadr (*hints*), pr-quote (0), *symbols*),
pr-equal (*conc*,
g-f-equal (g-cadr (*hints*),
g-cadr (*hints*)))),
pr-if (pr-equal (g-car (*hints*), pr-quote (4)),
g-fun-equal-axiom-proof (*conc*, *hints*, *symbols*),
pr-if (pr-equal (g-car (*hints*), pr-quote (5)),
g-pred-equal-axiom-proof (*conc*,
hints,
symbols),
pr-quote (bool-fix (**f**)))))))

```
;;;;;;;The EV analogues of the various FOL rule-checkers.
```

DEFINITION:

```
g-expan-rule (conc, hints, sub-goals, symbols)
=   pr-and (g-formula (g-cadr (hints), symbols),
            pr-and (g-formula (g-car (sub-goals), symbols),
                    pr-equal (conc, g-f-or (g-cadr (hints), g-car (sub-goals)))))
```

DEFINITION:

```
g-contrac-rule (conc, hints, sub-goals, symbols)
=   pr-and (g-formula (g-cadr (hints), symbols),
            pr-and (pr-equal (sub-goals,
                              g-cons (g-f-or (g-cadr (hints), g-cadr (hints)),
                                      pr-quote (gcode (nil)))),
                    pr-equal (conc, g-cadr (hints))))
```

DEFINITION:

```
g-assoc-rule (conc, hints, sub-goals, symbols)
=   pr-and (g-formula (g-cadr (hints), symbols),
            pr-and (g-formula (g-caddr (hints), symbols),
                    pr-and (g-formula (g-cadddr (hints), symbols),
                            pr-and (pr-equal (sub-goals,
                                              g-cons (g-f-or (g-cadr (hints),
                                                              g-f-or (g-caddr (hints),
                                                                      g-cadddr (hints))),
                                                      pr-quote (gcode (nil)))),
                                    pr-equal (conc,
                                              g-f-or (g-f-or (g-cadr (hints),
                                                              g-caddr (hints)),
                                                      g-cadddr (hints)))))))
```

DEFINITION:

```
g-cut-rule (conc, hints, sub-goals, symbols)
=   pr-and (g-formula (g-cadr (hints), symbols),
            pr-and (g-formula (g-caddr (hints), symbols),
                    pr-and (g-formula (g-cadddr (hints), symbols),
                            pr-and (pr-equal (sub-goals,
                                              g-cons (g-f-or (g-cadr (hints),
                                                              g-caddr (hints)),
                                                      g-cons (g-f-or (g-f-not (g-cadr (hints)),
                                                                      g-cadddr (hints)),
                                                              pr-quote (gcode (nil))))),
                                    pr-equal (conc,
                                              g-f-or (g-caddr (hints),
                                                      g-cadddr (hints)))))))
```

DEFINITION:

```
g-forsome-intro-rule (conc, hints, sub-goals, symbols)
=   pr-and (g-variable (g-cadr (hints)),
           pr-and (g-formula (g-caddr (hints), symbols),
                   pr-and (g-formula (g-cadddr (hints), symbols),
                           pr-and (pr-not (g-memb (g-cadr (hints),
                                                   g-collect-free (g-cadddr (hints),
                                                                   pr-quote (0)))),
                                   pr-and (pr-equal (sub-goals,
                                                     g-cons (g-f-implies (g-caddr (hints),
                                                                          g-cadddr (hints)),
                                                             pr-quote (gcode (nil)))),
                                           pr-equal (conc,
                                                     g-f-implies (g-forsome (g-cadr (hints),
                                                                             g-caddr (hints)),
                                                                  g-cadddr (hints))))))))
```

```
;;;;;;;The EV analogue of G-FOL-RULES.
```

DEFINITION:

```
g-fol-rules (conc, hints, sub-goals, symbols)
=   pr-if (pr-equal (g-car (hints), pr-quote (1)),
           g-expan-rule (conc, hints, sub-goals, symbols),
           pr-if (pr-equal (g-car (hints), pr-quote (2)),
                  g-contrac-rule (conc, hints, sub-goals, symbols),
                  pr-if (pr-equal (g-car (hints), pr-quote (3)),
                         g-assoc-rule (conc, hints, sub-goals, symbols),
                         pr-if (pr-equal (g-car (hints), pr-quote (4)),
                                g-cut-rule (conc, hints, sub-goals, symbols),
                                pr-if (pr-equal (g-car (hints), pr-quote (5)),
                                       g-forsome-intro-rule (conc,
                                                             hints,
                                                             sub-goals,
                                                             symbols),
                                       pr-quote (bool-fix (f)))))))
```

```
;;;;;;;The EV analogues of the various set-theoretic operations.
```

DEFINITION:

```
g-isin (x, y)
=   g-cons (g-cons (pr-quote (3), g-cons (pr-quote (1), pr-quote (2))),
            g-cons (x, g-cons (y, pr-quote (gcode (nil)))))
```

DEFINITION:

```
G-PHI
=   g-cons (g-cons (pr-quote (2), g-cons (pr-quote (0), pr-quote (0))),
            pr-quote (gcode (nil)))
```

Definition:
g-z-pair $(x,\ y)$
$=$ g-cons (g-cons (pr-quote (2), g-cons (pr-quote (1), pr-quote (2))),
g-cons (x, g-cons (y, pr-quote (gcode (**nil**)))))

Definition:
g-z-union $(x,\ y)$
$=$ g-cons (g-cons (pr-quote (2), g-cons (pr-quote (2), pr-quote (2))),
g-cons (x, g-cons (y, pr-quote (gcode (**nil**)))))

Definition:
g-z-int (x)
$=$ g-cons (g-cons (pr-quote (3), g-cons (pr-quote (2), pr-quote (1))),
g-cons (x, pr-quote (gcode (**nil**))))

Definition:
g-z-succ (x)
$=$ g-cons (g-cons (pr-quote (2), g-cons (pr-quote (3), pr-quote (1))),
g-cons (x, pr-quote (gcode (**nil**))))

Definition:
g-f-iff $(x,\ y) =$ g-f-and (g-f-implies $(x,\ y)$, g-f-implies $(y,\ x)$)

```
;;;;;;;The EV analogue of Z2-AXIOMS.
```

```
;;;;;;;The EV analogue of Z2-AXIOMS.
(DEFN
 G-Z2-AXIOMS
 (CONC HINTS SYMBOLS)
 (PR-IF
  (PR-EQUAL (G-CAR HINTS) (PR-QUOTE 1))
  (PR-AND
   (G-VARIABLE (G-CADDDR HINTS))
   (PR-AND
    (G-TERMP (G-CADR HINTS) (PR-QUOTE 0) SYMBOLS)
    (PR-AND
     (G-TERMP (G-CADDR HINTS) (PR-QUOTE 0) SYMBOLS)
     (PR-AND (PR-NOT (G-MEMB (G-CADDDR HINTS) (G-COLLECT-FREE (G-CADR HINTS) (PR-QUOTE 0))))
             (PR-AND (PR-NOT (G-MEMB (G-CADDDR HINTS)
                                     (G-COLLECT-FREE (G-CADDR HINTS) (PR-QUOTE 0))))
                     (PR-EQUAL CONC
                               (G-F-IFF (G-F-EQUAL (G-CADR HINTS) (G-CADDR HINTS))
                                        (G-FORALL (G-CADDDR HINTS)
                                                  (G-F-IFF (G-ISIN (G-CADDDR HINTS)
```

```
                                                                  (G-CADR HINTS))
                                                          (G-ISIN (G-CADDDR HINTS)
                                                                  (G-CADDR HINTS))))))))))))
(PR-IF
 (PR-EQUAL (G-CAR HINTS) (PR-QUOTE 2))
 (PR-AND (G-TERMP (G-CADR HINTS) (PR-QUOTE 0) SYMBOLS)
         (PR-EQUAL CONC (G-F-NOT (G-ISIN (G-CADR HINTS) (G-PHI)))))
 (PR-IF
  (PR-EQUAL (G-CAR HINTS) (PR-QUOTE 3))
  (PR-AND (G-TERMP (G-CONS (G-CADR HINTS)
                           (G-CONS (G-CADDR HINTS)
                                   (G-CONS (G-CADDDR HINTS) (PR-QUOTE (GCODE NIL)))))
                   (PR-QUOTE 1)
                   SYMBOLS)
          (PR-EQUAL CONC
                    (G-F-IFF (G-ISIN (G-CADDDR HINTS)
                                     (G-Z-PAIR (G-CADR HINTS) (G-CADDR HINTS)))
                             (G-F-OR (G-F-EQUAL (G-CADDDR HINTS) (G-CADR HINTS))
                                     (G-F-EQUAL (G-CADDDR HINTS) (G-CADDR HINTS))))))
  (PR-IF
   (PR-EQUAL (G-CAR HINTS) (PR-QUOTE 4))
   (PR-AND (G-TERMP (G-CONS (G-CADR HINTS)
                            (G-CONS (G-CADDR HINTS)
                                    (G-CONS (G-CADDDR HINTS) (PR-QUOTE (GCODE NIL)))))
                    (PR-QUOTE 1)
                    SYMBOLS)
           (PR-EQUAL CONC
                     (G-F-IFF (G-ISIN (G-CADDDR HINTS)
                                      (G-Z-UNION (G-CADR HINTS) (G-CADDR HINTS)))
                              (G-F-OR (G-ISIN (G-CADDDR HINTS) (G-CADR HINTS))
                                      (G-ISIN (G-CADDDR HINTS) (G-CADDR HINTS))))))
   (PR-IF
    (PR-EQUAL (G-CAR HINTS) (PR-QUOTE 5))
    (PR-AND
     (G-VARIABLE (G-CADDR HINTS))
     (PR-AND
      (G-FORMULA (G-CADR HINTS) SYMBOLS)
      (PR-EQUAL
       CONC
       (G-F-IMPLIES
        (G-F-AND (G-SUBST (G-CADR HINTS) (G-CADDR HINTS) (G-PHI) (PR-QUOTE 0))
                 (G-FORALL (G-CADDR HINTS)
                           (G-F-IMPLIES (G-F-AND (G-Z-INT (G-CADDR HINTS))
                                                 (G-CADR HINTS))
```

```
                                          (G-SUBST (G-CADR HINTS)
                                                   (G-CADDR HINTS)
                                                   (G-Z-SUCC (G-CADDR HINTS))
                                                   (PR-QUOTE 0)))))
          (G-FORALL (G-CADDR HINTS)
                    (G-F-IMPLIES (G-Z-INT (G-CADDR HINTS)) (G-CADR HINTS)))))))
      (PR-QUOTE (BOOL-FIX F))))))))

;;;;;;;The EV analogue of SYMB-DEFN-PROOF.
```

;;;;;;;The EV analogue of SYMB-DEFN-PROOF.

```
(DEFN
 G-SYMB-DEFN-PROOF
 (CONC HINTS SUB-GOALS DEFN SYMBOLS)
 (PR-IF
  (G-PREDICATE (G-CAR (G-CAR HINTS)))
  (PR-AND (G-SET (G-CDR (G-CAR HINTS)))
          (PR-AND (G-VAR-LIST (G-CDR (G-CAR HINTS)) (G-DEGREE (G-CAR (G-CAR HINTS))))
                  (PR-AND (PR-EQUAL CONC (G-F-IFF (G-CAR HINTS) DEFN))
                          (PR-AND (G-SUB (G-COLLECT-FREE DEFN (PR-QUOTE 0))
                                         (G-CDR (G-CAR HINTS)))
                                  (G-FORMULA DEFN (G-REST-OF (G-CAR (G-CAR HINTS)) SYMBOLS))))))
  (PR-IF
   (G-FUNCTION (G-CAR (G-CAR HINTS)))
   (PR-AND
    (G-SET (G-CONS (G-CADDDR HINTS) (G-CONS (G-CADR HINTS) (G-CDR (G-CAR HINTS)))))
    (PR-AND
     (G-VAR-LIST (G-CDR (G-CAR HINTS)) (G-DEGREE (G-CAR (G-CAR HINTS))))
     (PR-AND
      (G-VARIABLE (G-CADR HINTS))
      (PR-AND
       (G-VARIABLE (G-CADDDR HINTS))
       (PR-AND
        (G-FORMULA (G-CADDR HINTS) (G-REST-OF (G-CAR (G-CAR HINTS)) SYMBOLS))
        (PR-AND
         (G-SUB (G-COLLECT-FREE (G-CADDR HINTS) (PR-QUOTE 0))
                (G-CONS (G-CADR HINTS) (G-CDR (G-CAR HINTS))))
         (PR-AND
          (G-FREE-FOR (G-CADDR HINTS) (G-CADR HINTS) (G-CAR HINTS) (PR-QUOTE 0))
          (PR-AND
           (G-FREE-FOR (G-CADDR HINTS) (G-CADR HINTS) (G-CADDDR HINTS) (PR-QUOTE 0))
           (PR-AND
            (PR-EQUAL CONC
```

```
                    (G-SUBST (G-CADDR HINTS)
                             (G-CADR HINTS)
                             (G-CAR HINTS)
                             (PR-QUOTE 0)))
          (PR-AND
           (PR-EQUAL CONC DEFN)
           (PR-EQUAL
            SUB-GOALS
            (G-CONS (G-FORSOME (G-CADR HINTS) (G-CADDR HINTS))
                    (G-CONS (G-F-IMPLIES (G-F-AND (G-CADDR HINTS)
                                                  (G-SUBST (G-CADDR HINTS)
                                                           (G-CADR HINTS)
                                                           (G-CADDDR HINTS)
                                                           (PR-QUOTE 0)))
                                         (G-F-EQUAL (G-CADR HINTS)
                                                    (G-CADDDR HINTS)))
                            (PR-QUOTE (GCODE NIL))))))))))))))
 (PR-QUOTE (BOOL-FIX F)))))
```

EVENT: Enable g-symb-defn-proof; name this event ‘g1286’.

EVENT: Enable g-z2-axioms; name this event ‘g1287’.

EVENT: Enable g-fol-rules; name this event ‘g1288’.

```
;;;;;;;The e-definition of G-PRF.
```

DEFINITION:
g-prf-defn (*pf*, *given*, *defns*, *flg*, *symbols*)
= pr-if (pr-zerop (*flg*),
  pr-if (g-typep (*pf*, 1),
    pr-if (pr-not (g-formula (g-conc (*pf*, pr-quote (0)), *symbols*)),
      pr-quote (bool-fix (**f**)),
      pr-if (pr-equal (g-car (*pf*), pr-quote (1)),
        g-fol-axiom-proof (g-conc (*pf*, pr-quote (0)),
                           g-cadr (*pf*),
                           symbols),
        pr-if (pr-equal (g-car (*pf*), pr-quote (2)),
          pr-and (g-fol-rules (g-conc (*pf*,
                                       pr-quote (0)),
                               g-cadr (*pf*),
                               g-conc (g-cadddr (*pf*),
                                       pr-quote (1)),

```
                                      symbols),
                                g-prf (g-cadddr (pf),
                                       given,
                                       defns,
                                       pr-quote (1),
                                       symbols)),
                          pr-if (pr-equal (g-car (pf), pr-quote (3)),
                                g-z2-axioms (g-conc (pf,
                                                     pr-quote (0)),
                                             g-cadr (pf),
                                             symbols),
                                pr-if (pr-equal (g-car (pf),
                                                 pr-quote (4)),
                                      pr-and (g-symb-defn-proof (g-conc (pf,
                                                                         pr-quote (0)),
                                                                 g-cadr (pf),
                                                                 g-conc (g-cadddr (pf),
                                                                         pr-quote (1)),
                                                                 g-assoc (g-car (g-car (g-cadr (pf))),
                                                                          defns),
                                                                 symbols),
                                             pr-if (g-function (g-car (g-car (g-cadr (pf)))),
                                                    g-prf (g-cadddr (pf),
                                                           given,
                                                           defns,
                                                           pr-quote (1),
                                                           g-rest-of (g-car (g-car (g-cadr (pf))),
                                                                      symbols)),
                                                    pr-quote (bool-fix (t)))),
                                      pr-if (pr-equal (g-car (pf),
                                                       pr-quote (5)),
                                            g-memb (g-conc (pf,
                                                            pr-quote (0)),
                                                    given),
                                            pr-quote (bool-fix (f)))))))),
          pr-quote (bool-fix (f))),
    pr-if (g-typep (pf, 1),
          pr-and (g-prf (g-car (pf), given, defns, pr-quote (0), symbols),
                  g-prf (g-cdr (pf), given, defns, pr-quote (1), symbols)),
          pr-quote (bool-fix (t))))

;;;;;;;So far we have presented the e-definitions leading to the EV analogue
;;;;;;;of the proof-checker PRF.  We need a few more e-definitions in order to
;;;;;;;construct the undecidable sentence.
```

```
;;;;;;;Below is the e-definition for G-NUMERAL.
```

DEFINITION: g-numeral (x) = list $(29,\ x)$

DEFINITION:
G-NUMERAL-DEFN
= pr-if (pr-zerop (0), G-PHI, g-z-succ (g-numeral (pr-sub1 (0))))

DEFINITION: g-z-sing (x) = g-z-pair $(x,\ x)$

DEFINITION:
g-z-opair $(x,\ y)$ = g-z-pair (g-z-sing (x), g-z-pair $(x,\ y)$)

```
;;;;;;;The e-definition for G-PLUS (addition).
```

DEFINITION: g-plus $(x,\ y)$ = list $(30,\ x,\ y)$

DEFINITION:
G-PLUS-DEFN = pr-if (pr-zerop (0), g-fix (1), pr-add1 (g-plus (pr-sub1 (0), 1)))

```
;;;;;;;The e-definition for G-TIMES (multiplication).
```

DEFINITION: g-times $(x,\ y)$ = list $(31,\ x,\ y)$

DEFINITION:
G-TIMES-DEFN
= pr-if (pr-zerop (0),
    pr-quote (gcode (0)),
    g-plus (1, g-times (pr-sub1 (0), 1)))

```
;;;;;;;The e-definition for G-EXP (exponentiation).
```

DEFINITION: g-exp $(x,\ y)$ = list $(32,\ x,\ y)$

DEFINITION:
G-EXP-DEFN
= pr-if (pr-zerop (1), pr-quote (gcode (1)), g-times (0, g-exp (0, pr-sub1 (1))))

```
;;;;;;;The e-definition for G-HALVE (integer division by 2).
```

DEFINITION: g-halve (x) = list $(33,\ x)$

DEFINITION:
G-HALVE-DEFN
= pr-if (pr-zerop (0),
    pr-quote (gcode (0)),
    pr-if (pr-zerop (pr-sub1 (0)),
        pr-quote (gcode (0)),
        pr-add1 (g-halve (pr-sub1 (pr-sub1 (0))))))

```
;;;;;;;The e-definition for G-LESSP.
```

DEFINITION: g-lessp (x, y) = list (34, x, y)

DEFINITION:
G-LESSP-DEFN
= pr-if (pr-zerop (1),
    pr-quote (bool-fix (**f**)),
    pr-if (pr-zerop (0),
        pr-quote (bool-fix (**t**)),
        g-lessp (pr-sub1 (0), pr-sub1 (1))))

```
;;;;;;;With PLUS, TIMES, and EXP, we can cipher num-trees as numbers, and with these
;;;;;;;and the HALVE function, we can decipher back the num-trees from the numbers
;;;;;;;by a yet-to-be-defined function DECIPHER1.  The e-definition for G-DECIPHER1 is
;;;;;;;shown below.
```

DEFINITION: g-decipher1 (x, y, z) = list (35, x, y, z)

DEFINITION:
G-DECIPHER1-DEFN
= pr-if (pr-zerop (1),
    g-halve (pr-sub1 (0)),
    pr-if (pr-zerop (2),
        g-decipher1 (0, pr-sub1 (1), 0),
        pr-if (pr-and (g-lessp (1, 0), g-lessp (2, 0)),
            pr-if (pr-equal (0,
                    g-times (g-exp (pr-quote (gcode (2)), 1),
                        g-exp (pr-quote (gcode (3)), 2))),
                pr-cons (g-decipher1 (1, pr-sub1 (1), pr-sub1 (1)),
                    g-decipher1 (2, pr-sub1 (2), pr-sub1 (2))),
                g-decipher1 (0, 1, pr-sub1 (2))),
            g-decipher1 (0, 1, pr-sub1 (2)))))

DEFINITION:
g-proves (*pf*, *exp*, *given*, *defns*, *symbols*)
= pr-and (pr-equal (g-conc (*pf*, pr-quote (0)), *exp*),
    pr-and (g-formula (*exp*, *symbols*),
        g-prf (*pf*, *given*, *defns*, pr-quote (0), *symbols*)))

EVENT: Enable g-proves; name this event 'g1035'.

DEFINITION:
g-decipher (x) = g-decipher1 (x, pr-sub1 (x), pr-sub1 (x))

```
;;;;;;;G-INV-CODE is the EV analogue of a function INV-CODE which is the inverse for GCODE.
;;;;;;;At first sight, G-INV-CODE might seem like an identity mapping but that is only
;;;;;;;true when its argument is a GCODE and as we shall see, this is not always the case.
```

DEFINITION: g-inv-code (x) = list (36, x)

DEFINITION:
G-INV-CODE-DEFN
= pr-if (pr-listp (0),
    pr-if (pr-equal (pr-car (0), pr-quote (1)),
        g-cons (g-inv-code (pr-car (pr-cdr (0))),
                g-inv-code (pr-cdr (pr-cdr (0)))),
        pr-if (pr-equal (pr-car (0), pr-quote (2)),
            g-f-not (g-inv-code (pr-car (pr-cdr (0)))),
            pr-if (pr-equal (pr-car (0), pr-quote (3)),
                g-f-or (g-inv-code (pr-car (pr-cdr (0))),
                        g-inv-code (pr-cdr (pr-cdr (0)))),
                pr-if (pr-equal (pr-car (0), pr-quote (4)),
                    g-forsome (g-inv-code (pr-car (pr-cdr (0))),
                               g-inv-code (pr-cdr (pr-cdr (0)))),
                    pr-if (pr-equal (pr-car (0), pr-quote (5)),
                        pr-quote (gcode (**t**)),
                        pr-if (pr-equal (pr-car (0),
                                         pr-quote (6)),
                            pr-quote (gcode (**f**)),
                            pr-quote (gcode (**nil**)))))))),
    g-fix (0))

```
;;;;;;;G-ISTHM is the EV analogue of the 'theorem-checker' ISTHM which will be described
;;;;;;;later.  The e-definition is presented below.
```

DEFINITION:
g-isthm (*pfn*, *exp*, *given*, *defns*, *symbols*)
= list (37, *pfn*, *exp*, *given*, *defns*, *symbols*)

DEFINITION:
G-ISTHM-DEFN
= pr-if (g-proves (g-inv-code (g-decipher (0)), 1, 2, 3, 4),
    pr-quote (bool-fix (**t**)),
    pr-if (g-proves (g-inv-code (g-decipher (0)), g-f-not (1), 2, 3, 4),
        pr-quote (bool-fix (**f**)),
        g-isthm (pr-add1 (0), 1, 2, 3, 4)))

```
;;;;;;;The last e-definition is that of GC-NCODE which is the EV analogue of NCODE.
```

DEFINITION: gc-ncode (x) = list (38, x)

DEFINITION:
GC-NCODE-DEFN
= pr-if (pr-listp (0),
g-z-opair (gc-ncode (pr-car (0)), gc-ncode (pr-cdr (0))),
g-numeral (0))

```
;;;;;;;For the time being, we disable all the e-definitions.
```

EVENT: Enable g-append-defn; name this event 'g1036'.

EVENT: Enable g-del-defn; name this event 'g1037'.

EVENT: Enable g-memb-defn; name this event 'g1038'.

EVENT: Enable g-collect-free-defn; name this event 'g1039'.

EVENT: Enable g-covering-defn; name this event 'g1040'.

EVENT: Enable g-nil-intersect-defn; name this event 'g1041'.

EVENT: Enable g-var-list-defn; name this event 'g1042'.

EVENT: Enable g-length-defn; name this event 'g1043'.

EVENT: Enable g-termp-defn; name this event 'g1044'.

EVENT: Enable g-subst-defn; name this event 'g1045'.

EVENT: Enable g-sub-defn; name this event 'g1046'.

EVENT: Enable g-formula-defn; name this event 'g1047'.

EVENT: Enable g-pairequals-defn; name this event 'g1048'.

EVENT: Enable g-set-defn; name this event 'g1055'.

EVENT: Enable g-conc-defn; name this event 'g1056'.

EVENT: Enable g-rest-of-defn; name this event 'g1057'.

EVENT: Enable g-assoc-defn; name this event 'g1058'.

EVENT: Enable g-prf-defn; name this event 'g1059'.

EVENT: Enable g-numeral-defn; name this event 'g1060'.

EVENT: Enable g-plus-defn; name this event 'g1061'.

EVENT: Enable g-times-defn; name this event 'g1062'.

EVENT: Enable g-exp-defn; name this event 'g1063'.

EVENT: Enable g-halve-defn; name this event 'g1064'.

EVENT: Enable g-lessp-defn; name this event 'g1065'.

EVENT: Enable g-decipher1-defn; name this event 'g1066'.

EVENT: Enable g-inv-code-defn; name this event 'g1067'.

EVENT: Enable g-isthm-defn; name this event 'g1068'.

EVENT: Enable gc-ncode-defn; name this event 'g1069'.

EVENT: Disable sub1-n; name this event 'g1070'.

```
;;;;;;;The function FA below is the big list of e-definitions.
```

DEFINITION:
FA
= list (G-APPEND-DEFN,
G-DEL-DEFN,
G-MEMB-DEFN,
G-COLLECT-FREE-DEFN,
G-COVERING-DEFN,

G-NIL-INTERSECT-DEFN,
G-VAR-LIST-DEFN,
G-LENGTH-DEFN,
G-TERMP-DEFN,
G-SUBST-DEFN,
G-SUB-DEFN,
G-FORMULA-DEFN,
G-PAIREQUALS-DEFN,
G-SET-DEFN,
G-CONC-DEFN,
G-REST-OF-DEFN,
G-ASSOC-DEFN,
g-prf-defn (0, 1, 2, 3, 4),
G-NUMERAL-DEFN,
G-PLUS-DEFN,
G-TIMES-DEFN,
G-EXP-DEFN,
G-HALVE-DEFN,
G-LESSP-DEFN,
G-DECIPHER1-DEFN,
G-INV-CODE-DEFN,
G-ISTHM-DEFN,
GC-NCODE-DEFN)

EVENT: Disable sub1-n; name this event 'g1072'.

THEOREM: difference-lessp
$(x \leq y) \rightarrow ((x - y) = 0)$

THEOREM: difference-sub1-n
$(x \in \mathbf{N}) \rightarrow (\text{sub1-n}\,(x,\, y) = (x - y))$

EVENT: Enable sub1-n; name this event 'g1073'.

```
;;;;;;;We prove that GET of the e-function symbol (minus 11) on (FA) yields the
;;;;;;;appropriate e-definition for each of the defined e-function symbols.
```

THEOREM: get-g-append-defn
get (sub1-n (11, 11), FA) = G-APPEND-DEFN

THEOREM: get-sub1-n-nzerop
get $(1 + x,\, \mathit{list})$ = get $(x,\, \text{cdr}\,(\mathit{list}))$

THEOREM: difference-lessp2
$(y \leq x) \rightarrow (((1 + x) - y) = (1 + (x - y)))$

THEOREM: get-g-del-defn
get (sub1-n (12, 11), FA) = G-DEL-DEFN

THEOREM: get-g-memb-defn
get (sub1-n (13, 11), FA) = G-MEMB-DEFN

THEOREM: get-g-collect-free-defn
get (sub1-n (14, 11), FA) = G-COLLECT-FREE-DEFN

THEOREM: get-g-covering-defn
get (sub1-n (15, 11), FA) = G-COVERING-DEFN

THEOREM: get-g-nil-intersect-defn
get (sub1-n (16, 11), FA) = G-NIL-INTERSECT-DEFN

THEOREM: get-g-var-list-defn
get (sub1-n (17, 11), FA) = G-VAR-LIST-DEFN

THEOREM: get-g-length-defn
get (sub1-n (18, 11), FA) = G-LENGTH-DEFN

THEOREM: get-g-termp-defn
get (sub1-n (19, 11), FA) = G-TERMP-DEFN

THEOREM: get-g-subst-defn
get (sub1-n (20, 11), FA) = G-SUBST-DEFN

THEOREM: get-g-sub-defn
get (sub1-n (21, 11), FA) = G-SUB-DEFN

THEOREM: get-g-pairequals-defn
get (sub1-n (23, 11), FA) = G-PAIREQUALS-DEFN

THEOREM: get-g-set-defn
get (sub1-n (24, 11), FA) = G-SET-DEFN

THEOREM: get-g-conc-defn
get (sub1-n (25, 11), FA) = G-CONC-DEFN

THEOREM: get-g-rest-of-defn
get (sub1-n (26, 11), FA) = G-REST-OF-DEFN

THEOREM: get-g-assoc-defn
get (sub1-n (27, 11), FA) = G-ASSOC-DEFN

THEOREM: get-g-prf-defn
get (sub1-n (28, 11), FA) = g-prf-defn (0, 1, 2, 3, 4)

THEOREM: get-g-numeral-defn
get (sub1-n (29, 11), FA) = G-NUMERAL-DEFN

THEOREM: get-g-plus-defn
get (sub1-n (30, 11), FA) = G-PLUS-DEFN

THEOREM: get-g-times-defn
get (sub1-n (31, 11), FA) = G-TIMES-DEFN

THEOREM: get-g-exp-defn
get (sub1-n (32, 11), FA) = G-EXP-DEFN

THEOREM: get-g-halve-defn
get (sub1-n (33, 11), FA) = G-HALVE-DEFN

THEOREM: get-g-lessp-defn
get (sub1-n (34, 11), FA) = G-LESSP-DEFN

THEOREM: get-g-decipher1-defn
get (sub1-n (35, 11), FA) = G-DECIPHER1-DEFN

THEOREM: get-g-inv-code-defn
get (sub1-n (36, 11), FA) = G-INV-CODE-DEFN

THEOREM: get-g-isthm-defn
get (sub1-n (37, 11), FA) = G-ISTHM-DEFN

THEOREM: get-gc-ncode-defn
get (sub1-n (38, 11), FA) = GC-NCODE-DEFN

EVENT: Enable fa; name this event 'g1074'.

EVENT: Enable difference-sub1-n; name this event 'g1075'.

EVENT: Enable difference-lessp; name this event 'g1076'.

EVENT: Enable difference-lessp2; name this event 'g1077'.

```
;;;;;;;In the next series of events, we prove that the e-functions defined correspond
;;;;;;;to their Lisp analogues.  This involves two steps, in the first step we show,
;;;;;;;for example, that G-APPEND when evaluated corresponds to G-APPENDN.  This is done
;;;;;;;in EVAL-G-APPEND.  In the
;;;;;;;second step we show that G-APPENDN when given a large enough stack bound N
;;;;;;;corresponds to APPEND.  This is done in G-APPENDN-GCODE.
;;;;;;;We do the same thing for each of the e-functions.
;;;;;;;G-APPEND-IND is the induction scheme for EVAL-G-APPEND, and is quite subtle
;;;;;;;and could not have been guessed by the theorem-prover.
```

DEFINITION:
g-append-ind $(x,\ y,\ va,\ n)$
$=$ **if** $n \simeq 0$ **then t**
else g-append-ind (g-cdr (0), 1, ev (1, list $(x,\ y)$, va, FA, n), $n-1$) **endif**

```
;;;;;;;G-APPENDN is the Lisp analogue of G-APPEND with N.
```

DEFINITION:
g-appendn $(x,\ y,\ n)$
$=$ **if** $(n \simeq 0) \lor$ btmp $(x) \lor$ btmp (y) **then** BTM
else pr-ifn (g-typepn $(x,\ 1)$,
g-consn (g-carn (x), g-appendn (g-cdrn (x), y, $n-1$)),
y) **endif**

EVENT: Disable g-append; name this event 'g1078'.

EVENT: Disable g-append-defn; name this event 'g1079'.

```
;;;;;;;EV of G-APPEND applied to arguments is G-APPENDN of the EV of the arguments.
```

THEOREM: eval-g-append
ev (0, g-append $(x,\ y)$, va, FA, n)
$=$ g-appendn (ev (0, x, va, FA, n), ev (0, y, va, FA, n), n)

```
;;;;;;;APPENDN-IND is the induction scheme for G-APPENDN-GCODE.
```

DEFINITION:
appendn-ind $(x,\ y,\ n)$
$=$ **if** listp (x) **then** appendn-ind (cdr (x), y, $n-1$)
else t endif

```
;;;;;;;For a sufficiently large N, G-APPENDN of the GCODE of the arguments is
;;;;;;;the GCODE of the APPEND of the arguments.
```

THEOREM: g-appendn-gcode
(length $(x) < n$)
$\rightarrow$ (g-appendn (gcode (x), gcode (y), n) $=$ gcode (append $(x,\ y)$))

EVENT: Enable g-appendn; name this event 'g1080'.

EVENT: Enable g-append; name this event 'g1081'.

EVENT: Enable g-append-defn; name this event 'g1082'.

EVENT: Enable eval-fun2; name this event 'g1083'.

```
;;;;;;;Correspondence between G-DEL and G-DELN, and G-DELN and DEL.
```

DEFINITION:
g-deln $(x,\ y,\ n)$
$=$ **if** $(n \simeq \texttt{0}) \vee \text{btmp}(x) \vee \text{btmp}(y)$ **then** BTM
else pr-ifn (g-typepn $(y,\ \texttt{1})$,
pr-ifn (pr-equaln $(x$, g-carn $(y))$,
g-deln $(x$, g-cdrn $(y),\ n-1)$,
g-consn (g-carn (y), g-deln $(x$, g-cdrn $(y),\ n-1)))$,
$y)$ **endif**

DEFINITION:
g-del-ind $(x,\ y,\ va,\ n)$
$=$ **if** $n \simeq \texttt{0}$ **then t**
else g-del-ind $(\texttt{0}$, g-cdr $(\texttt{1})$, ev $(\texttt{1}$, list $(x,\ y),\ va,\ \text{FA},\ n),\ n-1)$ **endif**

EVENT: Disable g-del; name this event 'g1084'.

EVENT: Disable g-del-defn; name this event 'g1085'.

THEOREM: eval-g-del
ev $(\texttt{0}$, g-del $(x,\ y),\ va,\ \text{FA},\ n) =$ g-deln (ev $(\texttt{0},\ x,\ va,\ \text{FA},\ n)$, ev $(\texttt{0},\ y,\ va,\ \text{FA},\ n),\ n)$

THEOREM: ncode-equal-reflex
ncode-equal $(x,\ x)$

THEOREM: numberp-ncode-equal
$((x \in \mathbf{N}) \wedge (y \in \mathbf{N})) \rightarrow$ (ncode-equal $(x,\ y) = (x = y))$

THEOREM: ncode-gcode-equal
ncode-equal (gcode (x), gcode $(y)) =$ eql $(x,\ y)$

EVENT: Enable numberp-ncode-equal; name this event 'g1086'.

EVENT: Disable pr-equaln; name this event 'g1087'.

THEOREM: g-deln-gcode
(length $(y) < n) \rightarrow$ (g-deln (gcode (x), gcode $(y),\ n) =$ gcode (del $(x,\ y)))$

EVENT: Enable g-deln; name this event 'g1088'.

EVENT: Enable g-del; name this event 'g1089'.

EVENT: Enable g-del-defn; name this event 'g1090'.

`;;;;;;;Correspondence between G-MEMB, G-MEMBN, and MEMB.`

DEFINITION:
g-membn $(x,\ y,\ n)$
$=$ **if** $(n \simeq \texttt{0}) \lor \text{btmp}\,(x) \lor \text{btmp}\,(y)$ **then** BTM
else pr-ifn (g-typepn $(y,\ \texttt{1})$,
pr-ifn (pr-equaln $(x,\ \text{g-carn}\,(y))$,
`1`,
g-membn $(x,\ \text{g-cdrn}\,(y),\ n - 1)$),
`0`) **endif**

EVENT: Disable g-memb; name this event 'g1091'.

EVENT: Disable g-memb-defn; name this event 'g1092'.

DEFINITION:
g-memb-ind $(x,\ y,\ va,\ n)$
$=$ **if** $n \simeq \texttt{0}$ **then t**
else g-memb-ind (`0`, g-cdr (`1`), ev (`1`, list $(x,\ y)$, va, FA, n), $n - 1$) **endif**

THEOREM: g-memb-eval
ev (`0`, g-memb $(x,\ y)$, va, FA, n) $=$ g-membn (ev (`0`, x, va, FA, n), ev (`0`, y, va, FA, n), n)

THEOREM: g-membn-gcode
(length $(y) < n$)
$\rightarrow$ (g-membn (gcode (x), gcode (y), n)
$=$ **if** memb $(x,\ y)$ **then** `1`
else `0` **endif**)

EVENT: Enable g-memb; name this event 'g1093'.

EVENT: Enable g-membn; name this event 'g1094'.

EVENT: Enable g-memb-defn; name this event 'g1095'.

`;;;;;;;Correspondence between G-COLLECT-FREE, G-COLLECT-FREEN, and COLLECT-FREE.`

DEFINITION:
g-collect-freen $(exp,\ flg,\ n)$
$=$ **if** $(n \simeq \texttt{0}) \lor \text{btmp}\,(exp) \lor \text{btmp}\,(flg)$ **then** BTM
else pr-ifn (pr-zeropn (flg),
pr-ifn (g-variablen (exp),
g-consn (exp, gcode (**nil**)),

```
          pr-ifn (g-typepn (exp, 2),
                  g-collect-freen (g-argn (exp, 0, 2),
                                   0,
                                   n − 1),
                  pr-ifn (g-typepn (exp, 3),
                          g-appendn (g-collect-freen (g-argn (exp,
                                                              0,
                                                              3),
                                                      0,
                                                      n − 1),
                                     g-collect-freen (g-argn (exp,
                                                              1,
                                                              3),
                                                      0,
                                                      n − 1),
                                     n − 1),
                          pr-ifn (g-typepn (exp, 4),
                                  g-deln (g-argn (exp, 0, 4),
                                          g-collect-freen (g-argn (exp,
                                                                   1,
                                                                   4),
                                                           0,
                                                           n − 1),
                                          n − 1),
                                  pr-ifn (g-typepn (exp, 1),
                                          g-collect-freen (g-cdrn (exp),
                                                           1,
                                                           n − 1),
                                          gcode (nil)))))),
   pr-ifn (g-typepn (exp, 1),
           g-appendn (g-collect-freen (g-carn (exp),
                                       0,
                                       n − 1),
                      g-collect-freen (g-cdrn (exp),
                                       1,
                                       n − 1),
                      n − 1),
           gcode (nil))) endif
```

EVENT: Disable g-collect-free; name this event ‘g1096’.

EVENT: Disable g-collect-free-defn; name this event ‘g1097’.

DEFINITION:
g-collect-free-ind $(exp,\ flg,\ va,\ n)$
$=$ **if** $n \simeq$ 0 **then t**
else g-collect-free-ind (g-arg (0, pr-quote (0), 2),
pr-quote (0),
ev (1, list $(exp,\ flg)$, va, FA, n),
$n - 1$)
$\wedge$ g-collect-free-ind (g-arg (0, pr-quote (0), 3),
pr-quote (0),
ev (1, list $(exp,\ flg)$, va, FA, n),
$n - 1$)
$\wedge$ g-collect-free-ind (g-arg (0, pr-quote (1), 3),
pr-quote (0),
ev (1, list $(exp,\ flg)$, va, FA, n),
$n - 1$)
$\wedge$ g-collect-free-ind (g-arg (0, pr-quote (1), 4),
pr-quote (0),
ev (1, list $(exp,\ flg)$, va, FA, n),
$n - 1$)
$\wedge$ g-collect-free-ind (g-cdr (0),
pr-quote (1),
ev (1, list $(exp,\ flg)$, va, FA, n),
$n - 1$)
$\wedge$ g-collect-free-ind (g-car (0),
pr-quote (0),
ev (1, list $(exp,\ flg)$, va, FA, n),
$n - 1$) **endif**

THEOREM: g-collect-free-eval
ev (0, g-collect-free $(exp,\ flg)$, va, FA, n)
$=$ g-collect-freen (ev (0, exp, va, FA, n), ev (0, flg, va, FA, n), n)

DEFINITION:
exp-count (exp)
$=$ **if** listp (exp) **then** $1 +$ (exp-count (car (exp)) $+$ exp-count (cdr (exp)))
elseif f-notp (exp) **then** $1 +$ exp-count (arg (exp))
elseif f-orp (exp)
then $1 +$ (exp-count (arg1 (exp)) $+$ exp-count (arg2 (exp)))
elseif forsomep (exp) **then** $1 +$ exp-count (body (exp))
else 1 **endif**

THEOREM: exp-count-append
exp-count (append $(x,\ y)$) $<$ (exp-count (x) $+$ exp-count (y))

THEOREM: exp-count-del
exp-count (del $(x,\ y)$) $<$ ($1 +$ exp-count (y))

THEOREM: length-exp-count
length $(x) < (1 + \text{exp-count}\,(x))$

THEOREM: length-append
length (append $(x,\ y)) = (\text{length}\,(x) + \text{length}\,(y))$

THEOREM: length-del
length (del $(x,\ y)) < (1 + \text{length}\,(y))$

THEOREM: collect-free-count
length (collect-free (*exp*, *flg*)) $<$ (1 + exp-count (*exp*))

DEFINITION:
collect-freen-ind (*exp*, *flg*, *n*)
= **if** *flg* $\simeq$ `0`
then if variable (*exp*) **then nil**
elseif f-notp (*exp*) **then** collect-freen-ind (arg (*exp*), `0`, $n - 1$)
elseif f-orp (*exp*)
then collect-freen-ind (arg1 (*exp*), `0`, $n - 1$)
$\wedge$ collect-freen-ind (arg2 (*exp*), `0`, $n - 1$)
elseif forsomep (*exp*) **then** collect-freen-ind (body (*exp*), `0`, $n - 1$)
elseif listp (*exp*) **then** collect-freen-ind (cdr (*exp*), `1`, $n - 1$)
else nil endif
elseif listp (*exp*)
then collect-freen-ind (car (*exp*), `0`, $n - 1$)
$\wedge$ collect-freen-ind (cdr (*exp*), `1`, $n - 1$)
else nil endif

THEOREM: g-collect-freen-gcode
(($\neg$ btmp (*flg*)) $\wedge$ (exp-count (*exp*) $< n$))
$\rightarrow$ (g-collect-freen (gcode (*exp*), *flg*, *n*) = gcode (collect-free (*exp*, *flg*)))

THEOREM: numberp-gcode
(gcode (*exp*) $\in \mathbf{N}$) = (*exp* $\in \mathbf{N}$)

THEOREM: collect-free-gcode-flg
collect-free (*exp*, gcode (*flg*)) = collect-free (*exp*, *flg*)

EVENT: Enable g-collect-free; name this event 'g1110'.

EVENT: Enable g-collect-freen; name this event 'g1111'.

EVENT: Enable g-collect-free-defn; name this event 'g1112'.

EVENT: Enable pr-andn; name this event 'g1113'.

```
;;;;;;;Correspondence between G-COVERING, G-COVERINGN, and COVERING.
```

DEFINITION:

```
g-coveringn (exp, var, flg, n)
=   if (n ≃ 0) ∨ btmp (exp) ∨ btmp (var) ∨ btmp (flg)  then BTM
    else pr-ifn (pr-zeropn (flg),
                 pr-ifn (g-typepn (exp, 2),
                         g-coveringn (g-argn (exp, 0, 2),
                                      var,
                                      gcode (0),
                                      n - 1),
                         pr-ifn (g-typepn (exp, 3),
                                 g-appendn (g-coveringn (g-argn (exp, 0, 3),
                                                         var,
                                                         gcode (0),
                                                         n - 1),
                                            g-coveringn (g-argn (exp, 1, 3),
                                                         var,
                                                         gcode (0),
                                                         n - 1),
                                            n - 1),
                                 pr-ifn (g-typepn (exp, 4),
                                         pr-ifn (g-membn (var,
                                                          g-collect-freen (exp,
                                                                           gcode (0),
                                                                           n - 1),
                                                          n - 1),
                                                 g-consn (g-argn (exp, 0, 4),
                                                          g-coveringn (g-argn (exp,
                                                                               1,
                                                                               4),
                                                                       var,
                                                                       gcode (0),
                                                                       n - 1)),
                                                 gcode (nil)),
                                         pr-ifn (g-typepn (exp, 1),
                                                 g-coveringn (g-cdrn (exp),
                                                              var,
                                                              gcode (1),
                                                              n - 1),
                                                 gcode (nil))))),
                 pr-ifn (g-typepn (exp, 1),
                         g-appendn (g-coveringn (g-carn (exp),
                                                 var,
```

```
                                    gcode (0),
                                    n - 1),
                        g-coveringn (g-cdrn (exp),
                                    var,
                                    gcode (1),
                                    n - 1),
                        n - 1),
                gcode (nil))) endif
```

EVENT: Disable g-covering; name this event 'g1114'.

EVENT: Disable g-covering-defn; name this event 'g1115'.

DEFINITION:

```
g-covering-ind (exp, var, flg, va, n)
=   if n ≃ 0 then t
    else g-covering-ind (g-arg (0, pr-quote (0), 2),
                         1,
                         pr-quote (0),
                         ev (1, list (exp, var, flg), va, FA, n),
                         n - 1)
         ∧  g-covering-ind (g-arg (0, pr-quote (0), 3),
                            1,
                            pr-quote (0),
                            ev (1, list (exp, var, flg), va, FA, n),
                            n - 1)
         ∧  g-covering-ind (g-arg (0, pr-quote (1), 3),
                            1,
                            pr-quote (0),
                            ev (1, list (exp, var, flg), va, FA, n),
                            n - 1)
         ∧  g-covering-ind (g-arg (0, pr-quote (1), 4),
                            1,
                            pr-quote (0),
                            ev (1, list (exp, var, flg), va, FA, n),
                            n - 1)
         ∧  g-covering-ind (g-cdr (0),
                            1,
                            pr-quote (1),
                            ev (1, list (exp, var, flg), va, FA, n),
                            n - 1)
         ∧  g-covering-ind (g-car (0),
                            1,
```

pr-quote (0),
ev (1, list (exp, var, flg), va, FA, n),
$n - 1$) **endif**

THEOREM: g-covering-eval
ev (0, g-covering (exp, var, flg), va, FA, n)
= g-coveringn (ev (0, exp, va, FA, n), ev (0, var, va, FA, n), ev (0, flg, va, FA, n), n)

THEOREM: covering-count
length (covering (exp, var, flg)) $<$ exp-count (exp)

THEOREM: pr-zeropn-gcode
pr-zeropn (gcode (x)) = bool-fix ($x \simeq 0$)

EVENT: Enable pr-zeropn; name this event 'g1116'.

EVENT: Disable bool-fix; name this event 'g1117'.

THEOREM: g-membn-gcode-again
(length (y) $< n$)
$\rightarrow$ (g-membn (gcode (x), gcode (y), n) = bool-fix (memb (x, y)))

EVENT: Enable bool-fix; name this event 'g1118'.

EVENT: Enable g-membn-gcode; name this event 'g1119'.

DEFINITION:
coveringn-ind (exp, var, flg, n)
= **if** $flg \simeq 0$
then if f-notp (exp) **then** coveringn-ind (arg (exp), var, 0, $n - 1$)
elseif f-orp (exp)
then coveringn-ind (arg1 (exp), var, 0, $n - 1$)
$\land$ coveringn-ind (arg2 (exp), var, 0, $n - 1$)
elseif forsomep (exp)
then if memb (var, collect-free (exp, 0))
then coveringn-ind (body (exp), var, 0, $n - 1$)
else nil endif
elseif listp (exp) **then** coveringn-ind (cdr (exp), var, 1, $n - 1$)
elseif $exp \in \mathbf{N}$ **then nil**
else nil endif
elseif listp (exp)
then coveringn-ind (car (exp), var, 0, $n - 1$)
$\land$ coveringn-ind (cdr (exp), var, 1, $n - 1$)
else nil endif

THEOREM: g-coveringn-gcode
$((1 + \text{exp-count}(exp)) < n)$
$\rightarrow \quad (\text{g-coveringn}(\text{gcode}(exp),\ \text{gcode}(var),\ \text{gcode}(flg),\ n)$
$\quad = \quad \text{gcode}(\text{covering}(exp,\ var,\ flg)))$

EVENT: Enable g-coveringn; name this event 'g1120'.

EVENT: Enable g-covering; name this event 'g1121'.

EVENT: Enable g-covering-defn; name this event 'g1122'.

```
;;;;;;;Correspondence between G-NIL-INTERSECT, G-NIL-INTERSECTN, and NIL-INTERSECT.
```

DEFINITION:
g-nil-intersectn $(x,\ y,\ n)$
$=$ **if** btmp $(x) \vee$ btmp $(y) \vee (n \simeq 0)$ **then** BTM
else pr-ifn (g-typepn $(x,\ 1)$,
pr-andn (pr-notn (g-membn (g-carn $(x),\ y,\ n - 1)$),
g-nil-intersectn (g-cdrn $(x),\ y,\ n - 1)$),
bool-fix (**t**)) **endif**

EVENT: Disable g-nil-intersect-defn; name this event 'g1123'.

EVENT: Disable g-nil-intersect; name this event 'g1124'.

THEOREM: g-nil-intersect-eval
ev $(0,$ g-nil-intersect $(x,\ y),\ va,$ FA, $n)$
$=$ g-nil-intersectn (ev $(0,\ x,\ va,$ FA, $n)$, ev $(0,\ y,\ va,$ FA, $n),\ n)$

EVENT: Enable pr-notn; name this event 'g1125'.

DEFINITION:
nil-intersect-ind $(x,\ y,\ n)$
$=$ **if** $n \simeq 0$ **then t**
elseif listp (x) **then** nil-intersect-ind (cdr $(x),\ y,\ n - 1)$
else t endif

THEOREM: g-nil-intersectn-gcode
$((\text{length}(x) + \text{length}(y)) < n)$
$\rightarrow \quad (\text{g-nil-intersectn}(\text{gcode}(x),\ \text{gcode}(y),\ n)$
$\quad = \quad \text{bool-fix}(\text{nil-intersect}(x,\ y)))$

EVENT: Enable g-nil-intersect; name this event ‘g1126’.

EVENT: Enable g-nil-intersectn; name this event ‘g1127’.

EVENT: Enable g-nil-intersect-defn; name this event ‘g1128’.

EVENT: Disable g-free-for; name this event ‘g1129’.

```
;;;;;;;Correspondence between G-FREE-FOR, G-FREE-FORN, and FREE-FOR.
```

DEFINITION:
g-free-forn (*exp*, *var*, *term*, *flg*, *n*)
= g-nil-intersectn (g-coveringn (*exp*, *var*, *flg*, *n*),
g-collect-freen (*term*, gcode (0), *n*),
n)

THEOREM: eval-g-free-for
ev (0, g-free-for (*exp*, *var*, *term*, *flg*), *va*, FA, *n*)
= g-free-forn (ev (0, *exp*, *va*, FA, *n*),
ev (0, *var*, *va*, FA, *n*),
ev (0, *term*, *va*, FA, *n*),
ev (0, *flg*, *va*, FA, *n*),
n)

EVENT: Disable free-for; name this event ‘g1130’.

THEOREM: g-free-forn-gcode
((1 + (exp-count (*exp*) + exp-count (*term*))) < *n*)
→ (g-free-forn (gcode (*exp*), gcode (*var*), gcode (*term*), gcode (*flg*), *n*)
= bool-fix (free-for (*exp*, *var*, *term*, *flg*)))

EVENT: Enable g-free-for; name this event ‘g1131’.

EVENT: Enable g-free-forn; name this event ‘g1132’.

EVENT: Enable free-for; name this event ‘g1133’.

```
;;;;;;;Correspondence between G-VAR-LIST, G-VAR-LISTN, and VAR-LIST.
```

DEFINITION:
g-var-listn (*list*, *len*, *n*)

$=$ **if** $(n \simeq 0) \vee$ btmp $(list) \vee$ btmp (len) **then** BTM
else pr-ifn (pr-orn (pr-zeropn (len), pr-notn (g-typepn $(list, 1)$)),
pr-andn (pr-zeropn (len), pr-notn (g-typepn $(list, 1)$)),
pr-andn (g-variablen (g-carn $(list)$),
g-var-listn (g-cdrn $(list)$,
pr-sub1n (len),
$n - 1$))) **endif**

EVENT: Disable g-var-list; name this event 'g1139'.

EVENT: Disable g-var-list-defn; name this event 'g1140'.

DEFINITION:
g-var-list-ind $(list, len, va, n)$
$=$ **if** $n \simeq 0$ **then t**
else g-var-list-ind (g-cdr (0),
pr-sub1 (1),
ev (1, list $(list, len)$, va, FA, n),
$n - 1$) **endif**

THEOREM: g-var-list-eval
ev (0, g-var-list $(list, len)$, va, FA, n)
$=$ g-var-listn (ev (0, $list$, va, FA, n), ev (0, len, va, FA, n), n)

DEFINITION:
var-listn-ind $(list, len, n)$
$=$ **if** $n \simeq 0$ **then t**
elseif $(len \simeq 0) \vee (list \simeq$ **nil**$)$ **then t**
else var-listn-ind (cdr $(list)$, $len - 1$, $n - 1$) **endif**

THEOREM: g-var-listn-gcode
$(len < n)$
$\rightarrow$ (g-var-listn (gcode $(list)$, gcode (len), n)
$=$ bool-fix (var-list $(list, len)$))

EVENT: Enable g-var-listn; name this event 'g1141'.

EVENT: Enable g-var-list; name this event 'g1142'.

EVENT: Enable g-var-list-defn; name this event 'g1143'.

THEOREM: pr-equaln-reduc
pr-equaln (gcode (x), gcode (y)) $=$ bool-fix (eql (x, y))

EVENT: Enable pr-equaln; name this event 'g1147'.

EVENT: Enable pr-orn; name this event 'g1150'.

`;;;;;;;Correspondence between G-SYMB, G-SYMBN, and SYMB.`

DEFINITION:
g-symbn (x, *symbols*, n)
= pr-orn (pr-equaln (x, g-consn (gcode (3), g-consn (gcode (0), gcode (2)))),
g-membn (x, *symbols*, n))

EVENT: Disable g-symb; name this event 'g1151'.

THEOREM: g-symb-eval
ev (0, g-symb (x, *symbols*), *va*, FA, n)
= g-symbn (ev (0, x, *va*, FA, n), ev (0, *symbols*, *va*, FA, n), n)

EVENT: Disable p; name this event 'g1152'.

EVENT: Disable bool-fix; name this event 'g1153'.

THEOREM: gcode-p
bool-fix (x = cons (3, cons (0, 2)))
= bool-fix (gcode (x) = gcode (cons (3, cons (0, 2))))

EVENT: Enable bool-fix; name this event 'g1154'.

THEOREM: g-symbn-gcode
(length (*symbols*) $<$ n)
$\rightarrow$ (g-symbn (gcode (x), gcode (*symbols*), n) = bool-fix (symb (x, *symbols*)))

EVENT: Enable p; name this event 'g1155'.

EVENT: Enable g-symb; name this event 'g1156'.

EVENT: Enable g-symbn; name this event 'g1157'.

`;;;;;;;Correspondence between G-LENGTH, G-LENGTHN, and LENGTH.`

DEFINITION:
g-lengthn (x, n)

$=$ **if** btmp $(x) \vee (n \simeq \mathtt{0})$ **then** BTM
else pr-ifn (g-typepn $(x,\ \mathtt{1})$,
pr-add1n (g-lengthn (g-cdrn $(x),\ n - 1)$),
gcode $(\mathtt{0})$) **endif**

EVENT: Disable g-length-defn; name this event 'g1158'.

EVENT: Disable g-length; name this event 'g1159'.

DEFINITION:
g-length-ind $(x,\ va,\ n)$
$=$ **if** $n \simeq \mathtt{0}$ **then t**
else g-length-ind (g-cdr $(\mathtt{0})$, ev $(\mathtt{1}$, list $(x),\ va$, FA, $n),\ n - 1)$ **endif**

THEOREM: eval-g-length
ev $(\mathtt{0}$, g-length $(x),\ va$, FA, $n)$ = g-lengthn (ev $(\mathtt{0},\ x,\ va$, FA, $n),\ n)$

THEOREM: add1-gcode
$(1 +$ gcode $(x))$ = gcode $(1 + x)$

THEOREM: g-lengthn-gcode
(length $(x) < n) \rightarrow$ (g-lengthn (gcode $(x),\ n)$ = gcode (length (x)))

EVENT: Enable g-length; name this event 'g1160'.

EVENT: Enable g-lengthn; name this event 'g1161'.

EVENT: Enable g-length-defn; name this event 'g1162'.

```
;;;;;;;Correspondence between G-TERMP, G-TERMPN, and TERMP.
```

DEFINITION:
g-termpn $(exp,\ flg,\ symbols,\ n)$
$=$ **if** btmp $(exp) \vee$ btmp $(flg) \vee$ btmp $(symbols) \vee (n \simeq \mathtt{0})$ **then** BTM
else pr-ifn (pr-zeropn (flg),
pr-ifn (g-typepn $(exp,\ \mathtt{1})$,
pr-andn (g-functionn (g-carn (exp)),
pr-andn (g-symbn (g-carn (exp),
$symbols$,
$n - 1)$,
pr-andn (g-termpn (g-cdrn (exp),
gcode $(\mathtt{1})$,
$symbols$,

$n-1$),
pr-equaln (g-lengthn (g-cdrn (*exp*),
$n-1$),
g-fixn (g-cdrn (g-cdrn (g-carn (*exp*)))))))),
g-variablen (*exp*)),
pr-ifn (g-typepn (*exp*, 1),
pr-andn (g-termpn (g-carn (*exp*),
gcode (0),
symbols,
$n-1$),
g-termpn (g-cdrn (*exp*),
gcode (1),
symbols,
$n-1$)),
bool-fix (**t**))) **endif**

EVENT: Disable g-termp; name this event 'g1163'.

EVENT: Disable g-termp-defn; name this event 'g1164'.

DEFINITION:
g-termp-ind (*exp*, *flg*, *symbols*, *va*, *n*)
= **if** $n \simeq 0$ **then t**
else g-termp-ind (g-cdr (0),
pr-quote (1),
2,
ev (1, list (*exp*, *flg*, *symbols*), *va*, FA, *n*),
$n-1$)
$\land$ g-termp-ind (g-car (0),
pr-quote (0),
2,
ev (1, list (*exp*, *flg*, *symbols*), *va*, FA, *n*),
$n-1$) **endif**

THEOREM: g-termp-eval
ev (0, g-termp (*exp*, *flg*, *symbols*), *va*, FA, *n*)
= g-termpn (ev (0, *exp*, *va*, FA, *n*), ev (0, *flg*, *va*, FA, *n*), ev (0, *symbols*, *va*, FA, *n*), *n*)

DEFINITION:
termp-ind (*exp*, *flg*, *symbols*, *n*)
= **if** $n \simeq 0$ **then t**
elseif $flg \simeq 0$
then if listp (*exp*) **then** termp-ind (cdr (*exp*), 1, *symbols*, $n-1$)
else t endif

elseif listp (exp)
then termp-ind (car (exp), 0, $symbols$, $n - 1$)
$\wedge$ termp-ind (cdr (exp), 1, $symbols$, $n - 1$)
else t endif

THEOREM: length-listp-cdr
listp $(x) \rightarrow$ (length (cdr (x)) $<$ length (x))

THEOREM: not-length-cdr
length $(x) \not< $ length (cdr (x))

DEFINITION:
termp-cnt $(exp, flg, symbols)$
$=$ **if** $flg \simeq 0$
then if listp (exp) **then** $1 +$ termp-cnt (cdr (exp), 1, $symbols$)
else $1 + (1 +$ length $(symbols))$ **endif**
elseif listp (exp)
then $1 +$ (termp-cnt (car (exp), 0, $symbols$)
$+$ termp-cnt (cdr (exp), 1, $symbols$))
else $1 + (1 +$ length $(symbols))$ **endif**

THEOREM: lessp-symbols-termp-cnt
length $(symbols) <$ (termp-cnt $(exp, flg, symbols) - 1$)

THEOREM: lessp-length-termp-cnt
length $(exp) <$ termp-cnt $(exp, flg, symbols)$

EVENT: Disable degree; name this event ‘g1165’.

THEOREM: numberp-gcode-equal
$((x \in \mathbf{N}) \vee (y \in \mathbf{N})) \rightarrow (($gcode $(x) =$ gcode $(y)) = (x = y))$

THEOREM: g-termpn-gcode
(termp-cnt $(exp, flg, symbols) < n$)
$\rightarrow$ (g-termpn (gcode (exp), gcode (flg), gcode $(symbols)$, n)
$=$ bool-fix (termp $(exp, flg, symbols)$))

EVENT: Enable g-termp; name this event ‘g1166’.

EVENT: Enable g-termpn; name this event ‘g1167’.

EVENT: Enable g-termp-defn; name this event ‘g1168’.

EVENT: Enable termp-cnt; name this event ‘g1169’.

`;;;;;;;Correspondence between G-ATOMP, G-ATOMPN, and ATOMP.`

DEFINITION:
g-atompn (*exp*, *symbols*, *n*)
= pr-andn (g-predicaten (g-carn (*exp*)),
pr-andn (g-symbn (g-carn (*exp*), *symbols*, *n*),
pr-andn (pr-equaln (g-lengthn (g-cdrn (*exp*), *n*),
g-fixn (g-cdrn (g-cdrn (g-carn (*exp*))))),
g-termpn (g-cdrn (*exp*), gcode (`1`), *symbols*, *n*))))

EVENT: Disable g-atomp; name this event 'g1170'.

THEOREM: g-atomp-eval
ev (`0`, g-atomp (*exp*, *symbols*), *va*, FA, *n*)
= g-atompn (ev (`0`, *exp*, *va*, FA, *n*), ev (`0`, *symbols*, *va*, FA, *n*), *n*)

EVENT: Disable pr-andn; name this event 'g1171'.

EVENT: Disable bool-fix; name this event 'g1172'.

THEOREM: lazy-pr-andn-reduc
pr-andn (bool-fix (**f**), *y*) = bool-fix (**f**)

THEOREM: lazy-pr-andn-reduc2
pr-andn (bool-fix (**t**), bool-fix (*y*)) = bool-fix (*y*)

EVENT: Enable pr-andn; name this event 'g1173'.

EVENT: Enable bool-fix; name this event 'g1174'.

THEOREM: g-atompn-gcode
(termp-cnt (cdr (*exp*), `1`, *symbols*) < *n*)
→ (g-atompn (gcode (*exp*), gcode (*symbols*), *n*)
= bool-fix (atomp (*exp*, *symbols*)))

EVENT: Enable g-atomp; name this event 'g1175'.

EVENT: Enable g-atompn; name this event 'g1176'.

`;;;;;;;Correspondence between G-SUBST, G-SUBSTN, and SUBST.`

DEFINITION:

```
g-substn (exp, var, term, flg, n)
=   if btmp (exp) ∨ btmp (var) ∨ btmp (term) ∨ btmp (flg) ∨ (n ≃ 0)
    then BTM
    else pr-ifn (pr-zeropn (flg),
                 pr-ifn (g-variablen (exp),
                         pr-ifn (pr-equaln (exp, var), term, exp),
                         pr-ifn (g-typepn (exp, 2),
                                 g-f-notn (g-substn (g-argn (exp, 0, 2),
                                                     var,
                                                     term,
                                                     gcode (0),
                                                     n − 1)),
                                 pr-ifn (g-typepn (exp, 3),
                                         g-f-orn (g-substn (g-argn (exp,
                                                                    0,
                                                                    3),
                                                            var,
                                                            term,
                                                            gcode (0),
                                                            n − 1),
                                                  g-substn (g-argn (exp,
                                                                    1,
                                                                    3),
                                                            var,
                                                            term,
                                                            gcode (0),
                                                            n − 1)),
                                         pr-ifn (g-typepn (exp, 4),
                                                 pr-ifn (pr-equaln (g-argn (exp,
                                                                            0,
                                                                            4),
                                                                    var),
                                                         exp,
                                                         g-forsomen (g-argn (exp,
                                                                             0,
                                                                             4),
                                                                     g-substn (g-argn (exp,
                                                                                       1,
                                                                                       4),
                                                                               var,
                                                                               term,
                                                                               gcode (0),
                                                                               n − 1))),
```

```
                                    pr-ifn (g-typepn (exp, 1),
                                            g-consn (g-carn (exp),
                                                     g-substn (g-cdrn (exp),
                                                               var,
                                                               term,
                                                               gcode (1),
                                                               n − 1)),
                                            exp))))),
                  pr-ifn (g-typepn (exp, 1),
                          g-consn (g-substn (g-carn (exp),
                                             var,
                                             term,
                                             gcode (0),
                                             n − 1),
                                   g-substn (g-cdrn (exp),
                                             var,
                                             term,
                                             gcode (1),
                                             n − 1)),
                          exp)) endif
```

EVENT: Disable g-subst; name this event ‘g1177’.

EVENT: Disable g-subst-defn; name this event ‘g1178’.

DEFINITION:

```
g-subst-ind (exp, var, term, flg, va, n)
=   if n ≃ 0 then t
    else g-subst-ind (g-arg (0, pr-quote (0), 2),
                      1,
                      2,
                      pr-quote (gcode (0)),
                      ev (1, list (exp, var, term, flg), va, FA, n),
                      n − 1)
         ∧  g-subst-ind (g-arg (0, pr-quote (0), 3),
                         1,
                         2,
                         pr-quote (gcode (0)),
                         ev (1, list (exp, var, term, flg), va, FA, n),
                         n − 1)
         ∧  g-subst-ind (g-arg (0, pr-quote (1), 3),
                         1,
                         2,
```

```
                              pr-quote (gcode (0)),
                              ev (1, list (exp, var, term, flg), va, FA, n),
                              n − 1)
        ∧   g-subst-ind (g-arg (0, pr-quote (1), 4),
                              1,
                              2,
                              pr-quote (gcode (0)),
                              ev (1, list (exp, var, term, flg), va, FA, n),
                              n − 1)
        ∧   g-subst-ind (g-arg (0, pr-quote (1), 1),
                              1,
                              2,
                              pr-quote (gcode (1)),
                              ev (1, list (exp, var, term, flg), va, FA, n),
                              n − 1)
        ∧   g-subst-ind (g-arg (0, pr-quote (0), 1),
                              1,
                              2,
                              pr-quote (gcode (0)),
                              ev (1, list (exp, var, term, flg), va, FA, n),
                              n − 1) endif
```

THEOREM: g-subst-eval

```
 ev (0, g-subst (exp, var, term, flg), va, FA, n)
 =   g-substn (ev (0, exp, va, FA, n),
               ev (0, var, va, FA, n),
               ev (0, term, va, FA, n),
               ev (0, flg, va, FA, n),
               n)
```

DEFINITION:

```
subst-ind (exp, var, term, flg, n)
=   if flg ≃ 0
    then if variable (exp)  then t
         elseif f-notp (exp)  then subst-ind (arg (exp), var, term, 0, n − 1)
         elseif f-orp (exp)
         then subst-ind (arg1 (exp), var, term, 0, n − 1)
              ∧   subst-ind (arg2 (exp), var, term, 0, n − 1)
         elseif forsomep (exp)  then subst-ind (body (exp), var, term, 0, n − 1)
         elseif listp (exp)  then subst-ind (cdr (exp), var, term, 1, n − 1)
         else t endif
    elseif listp (exp)
    then subst-ind (car (exp), var, term, 0, n − 1)
         ∧   subst-ind (cdr (exp), var, term, 1, n − 1)
    else t endif
```

THEOREM: g-substn-gcode
$(\text{exp-count}(exp) < n)$
$\rightarrow \quad (\text{g-substn}(\text{gcode}(exp), \text{gcode}(var), \text{gcode}(term), \text{gcode}(flg), n)$
$\quad = \quad \text{gcode}(\text{subst}(exp, var, term, flg)))$

EVENT: Enable g-subst; name this event ‘g1179’.

EVENT: Enable g-substn; name this event ‘g1180’.

EVENT: Enable g-subst-defn; name this event ‘g1181’.

```
;;;;;;;Correspondence between G-SUB, G-SUBN, and SUB.
```

DEFINITION:
g-subn (x, y, n)
$=$ **if** btmp $(x) \vee$ btmp $(y) \vee (n \simeq 0)$ **then** BTM
else pr-ifn (g-typepn $(x, 1)$,
pr-andn (g-membn (g-carn (x), y, $n - 1$),
g-subn (g-cdrn (x), y, $n - 1$)),
bool-fix (**t**)) **endif**

EVENT: Disable g-sub; name this event ‘g1182’.

EVENT: Disable g-sub-defn; name this event ‘g1183’.

DEFINITION:
g-sub-ind (x, y, va, n)
$=$ **if** $n \simeq 0$ **then t**
else g-sub-ind (g-cdr (0), 1, ev (1, list (x, y), va, FA, n), $n - 1$) **endif**

THEOREM: g-sub-eval
ev (0, g-sub (x, y), va, FA, n) = g-subn (ev (0, x, va, FA, n), ev (0, y, va, FA, n), n)

THEOREM: g-subn-gcode
$((\text{length}(x) + \text{length}(y)) < n)$
$\rightarrow \quad (\text{g-subn}(\text{gcode}(x), \text{gcode}(y), n) = \text{bool-fix}(\text{sub}(x, y)))$

EVENT: Enable g-sub; name this event ‘g1184’.

EVENT: Enable g-subn; name this event ‘g1185’.

EVENT: Enable g-sub-defn; name this event ‘g1186’.

`;;;;;;Correspondence between G-FORMULA, G-FORMULAN, and FORMULA.`

DEFINITION:
g-formulan (*exp*, *symbols*, *n*)
= **if** btmp (*exp*) $\vee$ btmp (*symbols*) $\vee$ ($n \simeq$ `0`) **then** BTM
else pr-ifn (g-typepn (*exp*, `2`),
g-formulan (g-argn (*exp*, `0`, `2`), *symbols*, $n - 1$),
pr-ifn (g-typepn (*exp*, `3`),
pr-andn (g-formulan (g-argn (*exp*, `0`, `3`),
symbols,
$n - 1$),
g-formulan (g-argn (*exp*, `1`, `3`),
symbols,
$n - 1$)),
pr-ifn (g-typepn (*exp*, `4`),
g-formulan (g-argn (*exp*, `1`, `4`),
symbols,
$n - 1$),
g-atompn (*exp*, *symbols*, $n - 1$)))) **endif**

EVENT: Disable g-formula; name this event 'g1187'.

EVENT: Disable g-formula-defn; name this event 'g1188'.

DEFINITION:
g-formula-ind (*exp*, *symbols*, *va*, *n*)
= **if** $n \simeq$ `0` **then** **t**
else g-formula-ind (g-arg (`0`, pr-quote (`0`), `2`),
`1`,
ev (`1`, list (*exp*, *symbols*), *va*, FA, *n*),
$n - 1$)
$\wedge$ g-formula-ind (g-arg (`0`, pr-quote (`0`), `3`),
`1`,
ev (`1`, list (*exp*, *symbols*), *va*, FA, *n*),
$n - 1$)
$\wedge$ g-formula-ind (g-arg (`0`, pr-quote (`1`), `3`),
`1`,
ev (`1`, list (*exp*, *symbols*), *va*, FA, *n*),
$n - 1$)
$\wedge$ g-formula-ind (g-arg (`0`, pr-quote (`1`), `4`),
`1`,
ev (`1`, list (*exp*, *symbols*), *va*, FA, *n*),
$n - 1$) **endif**

EVENT: Disable difference-sub1-n; name this event 'g1189'.

EVENT: Disable difference-lessp2; name this event 'g1190'.

EVENT: Disable fa; name this event 'g1191'.

EVENT: Disable get-sub1-n-nzerop; name this event 'g1192'.

EVENT: Disable difference-lessp; name this event 'g1193'.

THEOREM: get-g-formula-defn
get (sub1-n (22, 11), FA) = G-FORMULA-DEFN

EVENT: Enable difference-sub1-n; name this event 'g1194'.

EVENT: Enable difference-lessp2; name this event 'g1195'.

EVENT: Enable fa; name this event 'g1196'.

EVENT: Enable get-sub1-n-nzerop; name this event 'g1197'.

EVENT: Enable difference-lessp; name this event 'g1198'.

THEOREM: g-formula-eval
ev (0, g-formula (*exp*, *symbols*), *va*, FA, *n*)
= g-formulan (ev (0, *exp*, *va*, FA, *n*), ev (0, *symbols*, *va*, FA, *n*), *n*)

DEFINITION:
formula-ind (*exp*, *symbols*, *n*)
= **if** $n \simeq 0$ **then t**
elseif f-notp (*exp*) **then** formula-ind (arg (*exp*), *symbols*, $n - 1$)
elseif f-orp (*exp*)
then formula-ind (arg1 (*exp*), *symbols*, $n - 1$)
$\wedge$ formula-ind (arg2 (*exp*), *symbols*, $n - 1$)
elseif forsomep (*exp*) **then** formula-ind (body (*exp*), *symbols*, $n - 1$)
else t endif

DEFINITION:
formula-cnt (*exp*, *symbols*)

= **if** f-notp (*exp*) **then** 1 + formula-cnt (arg (*exp*), *symbols*)
elseif f-orp (*exp*)
then 1 + (formula-cnt (arg1 (*exp*), *symbols*)
+ formula-cnt (arg2 (*exp*), *symbols*))
elseif forsomep (*exp*) **then** 1 + formula-cnt (body (*exp*), *symbols*)
else 1 + termp-cnt (cdr (*exp*), `1`, *symbols*) **endif**

THEOREM: g-formulan-gcode
(formula-cnt (*exp*, *symbols*) $<$ *n*)
$\rightarrow$ (g-formulan (gcode (*exp*), gcode (*symbols*), *n*)
= bool-fix (formula (*exp*, *symbols*)))

EVENT: Enable g-formula; name this event 'g1199'.

EVENT: Enable g-formulan; name this event 'g1200'.

EVENT: Enable g-formula-defn; name this event 'g1201'.

EVENT: Enable formula-cnt; name this event 'g1202'.

```
;;;;;;;Correspondence between G-PAIREQUALS, G-PAIREQUALSN, and PAIREQUALS.
```

DEFINITION:
g-pairequalsn (*vars1*, *vars2*, *exp*, *n*)
= **if** btmp (*vars1*) $\vee$ btmp (*vars2*) $\vee$ btmp (*exp*) $\vee$ ($n \simeq$ `0`) **then** BTM
else pr-ifn (pr-andn (g-typepn (*vars1*, `1`), g-typepn (*vars2*, `1`)),
g-f-impliesn (g-f-equaln (g-carn (*vars1*), g-carn (*vars2*)),
g-pairequalsn (g-cdrn (*vars1*),
g-cdrn (*vars2*),
exp,
$n - 1$)),
exp) **endif**

EVENT: Disable g-pairequals; name this event 'g1203'.

EVENT: Disable g-pairequals-defn; name this event 'g1204'.

DEFINITION:
g-pairequals-ind (*vars1*, *vars2*, *exp*, *va*, *n*)
= **if** $n \simeq$ `0` **then t**
else g-pairequals-ind (g-cdr (`0`),
g-cdr (`1`),

2,
ev (1, list (*vars1*, *vars2*, *exp*), *va*, FA, *n*),
$n - 1$) **endif**

THEOREM: g-pairequals-eval
ev (0, g-pairequals (*vars1*, *vars2*, *exp*), *va*, FA, *n*)
= g-pairequalsn (ev (0, *vars1*, *va*, FA, *n*),
ev (0, *vars2*, *va*, FA, *n*),
ev (0, *exp*, *va*, FA, *n*),
n)

EVENT: Enable pairequals-reduc; name this event 'g1205'.

DEFINITION:
pairequals-ind (*vars1*, *vars2*, *exp*, *n*)
= **if** $n \simeq 0$ **then t**
elseif listp (*vars1*) $\wedge$ listp (*vars2*)
then pairequals-ind (cdr (*vars1*), cdr (*vars2*), *exp*, $n - 1$)
else t endif

THEOREM: g-pairequalsn-gcode
(length (*vars1*) $< n$)
$\rightarrow$ (g-pairequalsn (gcode (*vars1*), gcode (*vars2*), gcode (*exp*), *n*)
= gcode (pairequals (*vars1*, *vars2*, *exp*)))

EVENT: Enable g-pairequals; name this event 'g1206'.

EVENT: Enable g-pairequalsn; name this event 'g1207'.

EVENT: Enable g-pairequals-defn; name this event 'g1208'.

DEFINITION: g-cadrn (x) = g-carn (g-cdrn (x))

DEFINITION: g-caddrn (x) = g-carn (g-cdrn (g-cdrn (x)))

DEFINITION: g-cadddrn (x) = g-carn (g-cdrn (g-cdrn (g-cdrn (x))))

`;;;;;;;SUBST-AXIOM correspondence.`

DEFINITION:
g-subst-axiom-proofn (*conc*, *hints*, *symbols*, *n*)
= pr-andn (g-formulan (g-cadrn (*hints*), *symbols*, *n*),
pr-andn (g-variablen (g-caddrn (*hints*)),

```
            pr-andn (g-termpn (g-cadddrn (hints), gcode (0), symbols, n),
                     pr-andn (g-free-forn (g-cadrn (hints),
                                           g-caddrn (hints),
                                           g-cadddrn (hints),
                                           gcode (0),
                                           n),
                              pr-equaln (conc,
                                         g-f-impliesn (g-substn (g-cadrn (hints),
                                                                 g-caddrn (hints),
                                                                 g-cadddrn (hints),
                                                                 gcode (0),
                                                                 n),
                                                       g-forsomen (g-caddrn (hints),
                                                                   g-cadrn (hints))))))))
```

DEFINITION: g-degreen (x) = g-fixn (g-cdrn (g-cdrn (x)))

```
;;;;;;;Function equality axiom correspondence.
```

DEFINITION:

```
g-fun-equal-axiom-proofn (conc, hints, symbols, n)
=   pr-andn (g-termpn (g-caddrn (hints), gcode (1), symbols, n),
             pr-andn (g-termpn (g-cadddrn (hints), gcode (1), symbols, n),
                      pr-andn (g-functionn (g-cadrn (hints)),
                               pr-andn (pr-equaln (g-lengthn (g-caddrn (hints), n),
                                                   g-degreen (g-cadrn (hints))),
                                        pr-andn (pr-equaln (g-lengthn (g-cadddrn (hints),
                                                                       n),
                                                            g-degreen (g-cadrn (hints))),
                                                 pr-equaln (conc,
                                                            g-pairequalsn (g-caddrn (hints),
                                                                           g-cadddrn (hints),
                                                                           g-f-equaln (g-consn (g-cadrn (hints),
                                                                                                g-caddrn (hints)),
                                                                                       g-consn (g-cadrn (hints),
                                                                                                g-cadddrn (hints))),
                                                                           n)))))))
```

```
;;;;;;;Predicate equality axiom correspondence.
```

DEFINITION:

```
g-pred-equal-axiom-proofn (conc, hints, symbols, n)
=   pr-andn (g-termpn (g-caddrn (hints), gcode (1), symbols, n),
             pr-andn (g-termpn (g-cadddrn (hints), gcode (1), symbols, n),
                      pr-andn (g-predicaten (g-cadrn (hints)),
```

```
pr-andn (pr-equaln (g-lengthn (g-caddrn (hints), n),
                    g-degreen (g-cadrn (hints))),
         pr-andn (pr-equaln (g-lengthn (g-cadddrn (hints),
                                        n),
                             g-degreen (g-cadrn (hints))),
                  pr-equaln (conc,
                             g-pairequalsn (g-caddrn (hints),
                                            g-cadddrn (hints),
                                            g-f-impliesn (g-consn (g-cadrn (hints),
                                                                   g-caddrn (hints)),
                                                          g-consn (g-cadrn (hints),
                                                                   g-cadddrn (hints))),
                                            n)))))))
```

```
;;;;;;;Correspondence for FOL-AXIOM-PROOF.
```

DEFINITION:

```
g-fol-axiom-proofn (conc, hints, symbols, n)
=   pr-ifn (pr-equaln (g-carn (hints), gcode (1)),
            pr-andn (g-formulan (g-cadrn (hints), symbols, n),
                     pr-equaln (conc,
                                g-f-orn (g-f-notn (g-cadrn (hints)),
                                         g-cadrn (hints)))),
            pr-ifn (pr-equaln (g-carn (hints), gcode (2)),
                    g-subst-axiom-proofn (conc, hints, symbols, n),
                    pr-ifn (pr-equaln (g-carn (hints), gcode (3)),
                            pr-andn (g-termpn (g-cadrn (hints),
                                               gcode (0),
                                               symbols,
                                               n),
                                     pr-equaln (conc,
                                                g-f-equaln (g-cadrn (hints),
                                                            g-cadrn (hints)))),
                            pr-ifn (pr-equaln (g-carn (hints), gcode (4)),
                                    g-fun-equal-axiom-proofn (conc,
                                                              hints,
                                                              symbols,
                                                              n),
                                    pr-ifn (pr-equaln (g-carn (hints), gcode (5)),
                                            g-pred-equal-axiom-proofn (conc,
                                                                       hints,
                                                                       symbols,
                                                                       n),
                                            bool-fix (f))))))
```

THEOREM: g-fol-axiom-proof-eval

$$\begin{array}{l} \text{ev}\,(\texttt{0},\ \text{g-fol-axiom-proof}\,(conc,\ hints,\ symbols),\ va,\ \text{FA},\ n) \\ = \quad \text{g-fol-axiom-proofn}\,(\text{ev}\,(\texttt{0},\ conc,\ va,\ \text{FA},\ n), \\ \qquad\qquad \text{ev}\,(\texttt{0},\ hints,\ va,\ \text{FA},\ n), \\ \qquad\qquad \text{ev}\,(\texttt{0},\ symbols,\ va,\ \text{FA},\ n), \\ \qquad\qquad n) \end{array}$$

DEFINITION:

$$\begin{array}{l} \text{fol-axiom-cnt}\,(hints,\ symbols) \\ = \quad (1 + (\text{formula-cnt}\,(\text{second}\,(hints),\ symbols) \\ \qquad + \quad \text{termp-cnt}\,(\text{fourth}\,(hints),\ \texttt{0},\ symbols) \\ \qquad + \quad \text{exp-count}\,(\text{second}\,(hints)) \\ \qquad + \quad \text{exp-count}\,(\text{fourth}\,(hints)) \\ \qquad + \quad \text{termp-cnt}\,(\text{second}\,(hints),\ \texttt{0},\ symbols) \\ \qquad + \quad \text{termp-cnt}\,(\text{third}\,(hints),\ \texttt{1},\ symbols) \\ \qquad + \quad \text{termp-cnt}\,(\text{fourth}\,(hints),\ \texttt{1},\ symbols) \\ \qquad + \quad \text{length}\,(\text{third}\,(hints)) \\ \qquad + \quad \text{length}\,(\text{fourth}\,(hints)))) \end{array}$$

EVENT: Disable fol-axiom-proof; name this event ‘g1213’.

THEOREM: g-fol-axiom-proofn-gcode

$$\begin{array}{l} (\text{fol-axiom-cnt}\,(hints,\ symbols) < n) \\ \to \quad (\text{g-fol-axiom-proofn}\,(\text{gcode}\,(conc),\ \text{gcode}\,(hints),\ \text{gcode}\,(symbols),\ n) \\ \qquad = \quad \text{bool-fix}\,(\text{fol-axiom-proof}\,(conc,\ hints,\ symbols))) \end{array}$$

EVENT: Enable g-fol-axiom-proof; name this event ‘g1214’.

EVENT: Enable g-fol-axiom-proofn; name this event ‘g1215’.

EVENT: Enable fol-axiom-proof; name this event ‘g1216’.

DEFINITION:

$$\begin{array}{l} \text{g-expan-rulen}\,(conc,\ hints,\ sub\text{-}goals,\ symbols,\ n) \\ = \quad \text{pr-andn}\,(\text{g-formulan}\,(\text{g-cadrn}\,(hints),\ symbols,\ n), \\ \qquad\qquad \text{pr-andn}\,(\text{g-formulan}\,(\text{g-carn}\,(sub\text{-}goals),\ symbols,\ n), \\ \qquad\qquad\qquad \text{pr-equaln}\,(conc, \\ \qquad\qquad\qquad\qquad \text{g-f-orn}\,(\text{g-cadrn}\,(hints),\ \text{g-carn}\,(sub\text{-}goals))))) \end{array}$$

DEFINITION:

$$\begin{array}{l} \text{g-contrac-rulen}\,(conc,\ hints,\ sub\text{-}goals,\ symbols,\ n) \\ = \quad \text{pr-andn}\,(\text{g-formulan}\,(\text{g-cadrn}\,(hints),\ symbols,\ n), \\ \qquad\qquad \text{pr-andn}\,(\text{pr-equaln}\,(sub\text{-}goals, \end{array}$$

g-consn (g-f-orn (g-cadrn (*hints*),
g-cadrn (*hints*)),
gcode (**nil**))),
pr-equaln (*conc*, g-cadrn (*hints*))))

Definition:
g-assoc-rulen (*conc*, *hints*, *sub-goals*, *symbols*, *n*)
= pr-andn (g-formulan (g-cadrn (*hints*), *symbols*, *n*),
pr-andn (g-formulan (g-caddrn (*hints*), *symbols*, *n*),
pr-andn (g-formulan (g-cadddrn (*hints*), *symbols*, *n*),
pr-andn (pr-equaln (*sub-goals*,
g-consn (g-f-orn (g-cadrn (*hints*),
g-f-orn (g-caddrn (*hints*),
g-cadddrn (*hints*))),
gcode (**nil**))),
pr-equaln (*conc*,
g-f-orn (g-f-orn (g-cadrn (*hints*),
g-caddrn (*hints*)),
g-cadddrn (*hints*)))))))

Definition:
g-cut-rulen (*conc*, *hints*, *sub-goals*, *symbols*, *n*)
= pr-andn (g-formulan (g-cadrn (*hints*), *symbols*, *n*),
pr-andn (g-formulan (g-caddrn (*hints*), *symbols*, *n*),
pr-andn (g-formulan (g-cadddrn (*hints*), *symbols*, *n*),
pr-andn (pr-equaln (*sub-goals*,
g-consn (g-f-orn (g-cadrn (*hints*),
g-caddrn (*hints*)),
g-consn (g-f-orn (g-f-notn (g-cadrn (*hints*)),
g-cadddrn (*hints*)),
gcode (**nil**)))),
pr-equaln (*conc*,
g-f-orn (g-caddrn (*hints*),
g-cadddrn (*hints*)))))))

Definition:
g-forsome-intro-rulen (*conc*, *hints*, *sub-goals*, *symbols*, *n*)
= pr-andn (g-variablen (g-cadrn (*hints*)),
pr-andn (g-formulan (g-caddrn (*hints*), *symbols*, *n*),
pr-andn (g-formulan (g-cadddrn (*hints*), *symbols*, *n*),
pr-andn (pr-notn (g-membn (g-cadrn (*hints*),
g-collect-freen (g-cadddrn (*hints*),
gcode (0),
n),
n)),

```
pr-andn (pr-equaln (sub-goals,
                    g-consn (g-f-impliesn (g-caddrn (hints),
                                           g-cadddrn (hints)),
                             gcode (nil))),
         pr-equaln (conc,
                    g-f-impliesn (g-forsomen (g-cadrn (hints),
                                              g-caddrn (hints)),
                                  g-cadddrn (hints)))))))
```

```
;;;;;;;Correspondence for FOL-RULES.
```

DEFINITION:
g-fol-rulesn (*conc*, *hints*, *sub-goals*, *symbols*, *n*)

```
=   pr-ifn (pr-equaln (g-carn (hints), gcode (1)),
        g-expan-rulen (conc, hints, sub-goals, symbols, n),
        pr-ifn (pr-equaln (g-carn (hints), gcode (2)),
            g-contrac-rulen (conc, hints, sub-goals, symbols, n),
            pr-ifn (pr-equaln (g-carn (hints), gcode (3)),
                g-assoc-rulen (conc, hints, sub-goals, symbols, n),
                pr-ifn (pr-equaln (g-carn (hints), gcode (4)),
                    g-cut-rulen (conc,
                                 hints,
                                 sub-goals,
                                 symbols,
                                 n),
                    pr-ifn (pr-equaln (g-carn (hints), gcode (5)),
                        g-forsome-intro-rulen (conc,
                                               hints,
                                               sub-goals,
                                               symbols,
                                               n),
                        bool-fix (f))))))
```

EVENT: Disable g-fol-rules; name this event ‘g1217’.

THEOREM: g-fol-rules-eval
ev (0, g-fol-rules (*conc*, *hints*, *sub-goals*, *symbols*), *va*, FA, *n*)

```
=   g-fol-rulesn (ev (0, conc, va, FA, n),
                  ev (0, hints, va, FA, n),
                  ev (0, sub-goals, va, FA, n),
                  ev (0, symbols, va, FA, n),
                  n)
```

EVENT: Enable g-fol-rules; name this event ‘g1218’.

EVENT: Disable fol-rules; name this event 'g1219'.

THEOREM: g-fol-rulesn-gcode

$$
\begin{array}{l}
((\text{formula-cnt}\,(\text{second}\,(\textit{hints}),\ \textit{symbols}) \\
\quad +\ \ \text{formula-cnt}\,(\text{third}\,(\textit{hints}),\ \textit{symbols}) \\
\quad +\ \ \text{formula-cnt}\,(\text{fourth}\,(\textit{hints}),\ \textit{symbols}) \\
\quad +\ \ \text{formula-cnt}\,(\text{car}\,(\textit{sub-goals}),\ \textit{symbols}) \\
\quad +\ \ \text{exp-count}\,(\text{fourth}\,(\textit{hints}))) \\
\ \ <\ \ n) \\
\to\ \ (\text{g-fol-rulesn}\,(\text{gcode}\,(\textit{conc}), \\
\qquad\qquad\qquad \text{gcode}\,(\textit{hints}), \\
\qquad\qquad\qquad \text{gcode}\,(\textit{sub-goals}), \\
\qquad\qquad\qquad \text{gcode}\,(\textit{symbols}), \\
\qquad\qquad\qquad n) \\
\qquad =\ \ \text{bool-fix}\,(\text{fol-rules}\,(\textit{conc},\ \textit{hints},\ \textit{sub-goals},\ \textit{symbols})))
\end{array}
$$

EVENT: Enable fol-rules; name this event 'g1220'.

EVENT: Enable g-fol-rules; name this event 'g1221'.

EVENT: Enable g-fol-rulesn; name this event 'g1222'.

DEFINITION:

g-isinn $(x,\ y)$

$$
\begin{array}{l}
=\ \ \text{g-consn}\,(\text{g-consn}\,(\text{gcode}\,(3),\ \text{g-consn}\,(\text{gcode}\,(1),\ \text{gcode}\,(2))), \\
\qquad\qquad \text{g-consn}\,(x,\ \text{g-consn}\,(y,\ \text{gcode}\,(\mathbf{nil}))))
\end{array}
$$

DEFINITION:

G-PHIN

$$
=\ \ \text{g-consn}\,(\text{g-consn}\,(\text{gcode}\,(2),\ \text{g-consn}\,(\text{gcode}\,(0),\ \text{gcode}\,(0))),\ \text{gcode}\,(\mathbf{nil}))
$$

DEFINITION:

g-z-pairn $(x,\ y)$

$$
\begin{array}{l}
=\ \ \text{g-consn}\,(\text{g-consn}\,(\text{gcode}\,(2),\ \text{g-consn}\,(\text{gcode}\,(1),\ \text{gcode}\,(2))), \\
\qquad\qquad \text{g-consn}\,(x,\ \text{g-consn}\,(y,\ \text{gcode}\,(\mathbf{nil}))))
\end{array}
$$

DEFINITION:

g-z-unionn $(x,\ y)$

$$
\begin{array}{l}
=\ \ \text{g-consn}\,(\text{g-consn}\,(\text{gcode}\,(2),\ \text{g-consn}\,(\text{gcode}\,(2),\ \text{gcode}\,(2))), \\
\qquad\qquad \text{g-consn}\,(x,\ \text{g-consn}\,(y,\ \text{gcode}\,(\mathbf{nil}))))
\end{array}
$$

DEFINITION:

g-z-intn (x)

$=$ g-consn (g-consn (gcode (3), g-consn (gcode (2), gcode (1))),
g-consn (x, gcode (**nil**)))

Definition:
g-z-succn (x)
$=$ g-consn (g-consn (gcode (2), g-consn (gcode (3), gcode (1))),
g-consn (x, gcode (**nil**)))

Definition:
g-f-iffn (x, y) $=$ g-f-andn (g-f-impliesn (x, y), g-f-impliesn (y, x))

Theorem: g-f-iff-eval
ev (0, g-f-iff (x, y), va, fa, n) $=$ g-f-iffn (ev (0, x, va, fa, n), ev (0, y, va, fa, n))

Event: Disable f-iff; name this event 'g1223'.

Theorem: g-f-iffn-gcode
g-f-iffn (gcode (x), gcode (y)) $=$ gcode (f-iff (x, y))

Event: Enable f-iff; name this event 'g1224'.

Event: Enable g-f-iff; name this event 'g1225'.

Event: Enable g-f-iffn; name this event 'g1226'.

```
;;;;;;;Correspondence for Z2-AXIOMS.
```

```
;;;;;;;Correspondence for Z2-AXIOMS.
(DEFN
 G-Z2-AXIOMSN
 (CONC HINTS SYMBOLS N)
 (PR-IFN
  (PR-EQUALN (G-CARN HINTS) (GCODE 1))
  (PR-ANDN
   (G-VARIABLEN (G-CADDDRN HINTS))
   (PR-ANDN
    (G-TERMPN (G-CADRN HINTS) (GCODE 0) SYMBOLS N)
    (PR-ANDN
     (G-TERMPN (G-CADDRN HINTS) (GCODE 0) SYMBOLS N)
     (PR-ANDN
      (PR-NOTN (G-MEMBN (G-CADDDRN HINTS) (G-COLLECT-FREEN (G-CADRN HINTS) (GCODE 0) N) N))
      (PR-ANDN (PR-NOTN (G-MEMBN (G-CADDDRN HINTS)
```

```
                              (G-COLLECT-FREEN (G-CADDRN HINTS) (GCODE 0) N)
                              N))
                  (PR-EQUALN CONC
                             (G-F-IFFN (G-F-EQUALN (G-CADRN HINTS) (G-CADDRN HINTS))
                                       (G-FORALLN (G-CADDDRN HINTS)
                                                  (G-F-IFFN (G-ISINN (G-CADDDRN HINTS)
                                                                     (G-CADRN HINTS))
                                                            (G-ISINN (G-CADDDRN HINTS)
                                                                     (G-CADDRN HINTS))))))))))
(PR-IFN
 (PR-EQUALN (G-CARN HINTS) (GCODE 2))
 (PR-ANDN (G-TERMPN (G-CADRN HINTS) (GCODE 0) SYMBOLS N)
          (PR-EQUALN CONC (G-F-NOTN (G-ISINN (G-CADRN HINTS) (G-PHIN)))))
 (PR-IFN
  (PR-EQUALN (G-CARN HINTS) (GCODE 3))
  (PR-ANDN (G-TERMPN (G-CONSN (G-CADRN HINTS)
                              (G-CONSN (G-CADDRN HINTS)
                                       (G-CONSN (G-CADDDRN HINTS) (GCODE NIL))))
                     (GCODE 1)
                     SYMBOLS
                     N)
           (PR-EQUALN CONC
                      (G-F-IFFN (G-ISINN (G-CADDDRN HINTS)
                                         (G-Z-PAIRN (G-CADRN HINTS) (G-CADDRN HINTS)))
                                (G-F-ORN (G-F-EQUALN (G-CADDDRN HINTS) (G-CADRN HINTS))
                                         (G-F-EQUALN (G-CADDDRN HINTS) (G-CADDRN HINTS))))))
  (PR-IFN
   (PR-EQUALN (G-CARN HINTS) (GCODE 4))
   (PR-ANDN (G-TERMPN (G-CONSN (G-CADRN HINTS)
                               (G-CONSN (G-CADDRN HINTS)
                                        (G-CONSN (G-CADDDRN HINTS) (GCODE NIL))))
                      (GCODE 1)
                      SYMBOLS
                      N)
            (PR-EQUALN CONC
                       (G-F-IFFN (G-ISINN (G-CADDDRN HINTS)
                                          (G-Z-UNIONN (G-CADRN HINTS) (G-CADDRN HINTS)))
                                 (G-F-ORN (G-ISINN (G-CADDDRN HINTS) (G-CADRN HINTS))
                                          (G-ISINN (G-CADDDRN HINTS) (G-CADDRN HINTS))))))
   (PR-IFN
    (PR-EQUALN (G-CARN HINTS) (GCODE 5))
    (PR-ANDN
     (G-VARIABLEN (G-CADDRN HINTS))
     (PR-ANDN
```

```
        (G-FORMULAN (G-CADRN HINTS) SYMBOLS N)
        (PR-EQUALN
         CONC
         (G-F-IMPLIESN
          (G-F-ANDN (G-SUBSTN (G-CADRN HINTS) (G-CADDRN HINTS) (G-PHIN) (GCODE 0) N)
                    (G-FORALLN (G-CADDRN HINTS)
                               (G-F-IMPLIESN (G-F-ANDN (G-Z-INTN (G-CADDRN HINTS))
                                                       (G-CADRN HINTS))
                                             (G-SUBSTN (G-CADRN HINTS)
                                                       (G-CADDRN HINTS)
                                                       (G-Z-SUCCN (G-CADDRN HINTS))
                                                       (GCODE 0)
                                                       N))))
          (G-FORALLN (G-CADDRN HINTS)
                     (G-F-IMPLIESN (G-Z-INTN (G-CADDRN HINTS)) (G-CADRN HINTS))))))
      (BOOL-FIX F)))))))
```

EVENT: Disable g-z2-axioms; name this event 'g1238'.

THEOREM: g-z2-axioms-eval
ev (0, g-z2-axioms(*conc*, *hints*, *symbols*), *va*, FA, *n*)
= g-z2-axiomsn(ev (0, *conc*, *va*, FA, *n*),
ev (0, *hints*, *va*, FA, *n*),
ev (0, *symbols*, *va*, FA, *n*),
n)

EVENT: Disable p; name this event 'g1239'.

EVENT: Disable fn; name this event 'g1240'.

EVENT: Disable isin; name this event 'g1241'.

EVENT: Disable phi; name this event 'g1242'.

EVENT: Disable z-pair; name this event 'g1243'.

EVENT: Disable z-union; name this event 'g1244'.

EVENT: Disable z-int; name this event 'g1245'.

EVENT: Disable z-succ; name this event 'g1246'.

EVENT: Enable g-z2-axioms; name this event ‘g1247’.

DEFINITION:
z2-axioms-cnt (*hints*, *symbols*)
= (termp-cnt (second (*hints*), 0, *symbols*)
    + termp-cnt (third (*hints*), 0, *symbols*)
    + exp-count (second (*hints*))
    + exp-count (third (*hints*))
    + termp-cnt (list (second (*hints*), third (*hints*), fourth (*hints*)),
                1,
                symbols)
    + formula-cnt (second (*hints*), *symbols*))

THEOREM: g-z2-axiomsn-gcode
((termp-cnt (second (*hints*), 0, *symbols*)
  + termp-cnt (third (*hints*), 0, *symbols*)
  + exp-count (second (*hints*))
  + exp-count (third (*hints*))
  + termp-cnt (list (second (*hints*), third (*hints*), fourth (*hints*)), 1, *symbols*)
  + formula-cnt (second (*hints*), *symbols*))
 < *n*)
→ (g-z2-axiomsn (gcode (*conc*), gcode (*hints*), gcode (*symbols*), *n*)
    = bool-fix (z2-axioms (*conc*, *hints*, *symbols*)))

EVENT: Enable fn; name this event ‘g1248’.

EVENT: Enable isin; name this event ‘g1249’.

EVENT: Enable phi; name this event ‘g1250’.

EVENT: Enable z-pair; name this event ‘g1251’.

EVENT: Enable z-union; name this event ‘g1252’.

EVENT: Enable z-int; name this event ‘g1253’.

EVENT: Enable z-succ; name this event ‘g1254’.

EVENT: Enable g-z2-axioms; name this event ‘g1255’.

EVENT: Enable g-z2-axiomsn; name this event 'g1256'.

EVENT: Enable z2-axioms; name this event 'g1257'.

```
;;;;;;;Correspondence for SET.
```

DEFINITION:
g-setn $(x,\ n)$
$=$ **if** btmp $(x) \lor (n \simeq 0)$ **then** BTM
else pr-ifn (g-typepn $(x,\ 1)$,
pr-andn (pr-notn (g-membn (g-carn (x), g-cdrn (x), $n - 1)$),
g-setn (g-cdrn (x), $n - 1)$),
bool-fix (**t**)) **endif**

EVENT: Disable g-set; name this event 'g1258'.

EVENT: Disable g-set-defn; name this event 'g1259'.

DEFINITION:
g-set-ind $(x,\ va,\ n)$
$=$ **if** $n \simeq 0$ **then t**
else g-set-ind (g-cdr (0), ev (1, list (x), va, FA, n), $n - 1$) **endif**

THEOREM: g-set-eval
ev (0, g-set (x), va, FA, n) = g-setn (ev (0, x, va, FA, n), n)

THEOREM: g-setn-gcode
(length $(x) < n$) $\rightarrow$ (g-setn (gcode (x), n) = bool-fix (set (x)))

EVENT: Enable g-set; name this event 'g1260'.

EVENT: Enable g-setn; name this event 'g1261'.

EVENT: Enable g-set-defn; name this event 'g1262'.

```
;;;;;;;Correspondence for ASSOC.
```

DEFINITION:
g-assocn $(x,\ y,\ n)$
$=$ **if** btmp $(x) \lor$ btmp $(y) \lor (n \simeq 0)$ **then** BTM
else pr-ifn (g-typepn $(y,\ 1)$,
pr-ifn (pr-equaln (g-carn (g-carn (y)), x),

g-cdrn (g-carn (y)),
g-assocn (x, g-cdrn (y), $n - 1$)),
gcode (0)) **endif**

EVENT: Disable g-assoc-defn; name this event 'g1265'.

DEFINITION:
g-assoc-ind (x, y, va, n)
= **if** $n \simeq 0$ **then t**
else g-assoc-ind (0, g-cdr (1), ev (1, list (x, y), va, FA, n), $n - 1$) **endif**

THEOREM: g-assoc-eval
ev (0, g-assoc (x, y), va, FA, n) = g-assocn (ev (0, x, va, FA, n), ev (0, y, va, FA, n), n)

THEOREM: g-assocn-gcode
(length (y) $<$ n) $\rightarrow$ (g-assocn (gcode (x), gcode (y), n) = gcode (assoc (x, y)))

EVENT: Enable g-assoc; name this event 'g1266'.

EVENT: Enable g-assocn; name this event 'g1267'.

EVENT: Enable g-assoc-defn; name this event 'g1268'.

```
;;;;;;;Correspondence for CONC.
```

DEFINITION:
g-concn (pf, flg, n)
= **if** btmp (pf) $\vee$ btmp (flg) $\vee$ ($n \simeq 0$) **then** BTM
else pr-ifn (pr-zeropn (flg),
g-caddrn (pf),
pr-ifn (g-typepn (pf, 1),
g-consn (g-concn (g-carn (pf), gcode (0), $n - 1$),
g-concn (g-cdrn (pf), gcode (1), $n - 1$)),
gcode (**nil**))) **endif**

EVENT: Disable g-conc; name this event 'g1269'.

EVENT: Disable g-conc-defn; name this event 'g1270'.

DEFINITION:
g-conc-ind (pf, flg, va, n)
= **if** $n \simeq 0$ **then t**

else g-conc-ind (g-car (0),
pr-quote (0),
ev (1, list (pf, flg), va, FA, n),
$n - 1$)
$\land$ g-conc-ind (g-cdr (0),
pr-quote (1),
ev (1, list (pf, flg), va, FA, n),
$n - 1$) **endif**

THEOREM: g-conc-eval
ev (0, g-conc (pf, flg), va, FA, n)
= g-concn (ev (0, pf, va, FA, n), ev (0, flg, va, FA, n), n)

DEFINITION:
conc-ind (pf, flg, n)
= **if** $n \simeq 0$ **then t**
elseif $flg \simeq 0$ **then t**
else conc-ind (car (pf), 0, $n - 1$) $\land$ conc-ind (cdr (pf), 1, $n - 1$) **endif**

THEOREM: g-concn-gcode
(exp-count (pf) $< n$)
$\rightarrow$ (g-concn (gcode (pf), gcode (flg), n) = gcode (conc (pf, flg)))

EVENT: Enable g-conc; name this event ‘g1271’.

EVENT: Enable g-concn; name this event ‘g1272’.

EVENT: Enable g-conc-defn; name this event ‘g1273’.

```
;;;;;;;Correspondence for REST-OF.
```

DEFINITION:
g-rest-ofn (sym, $symbols$, n)
= **if** btmp (sym) $\lor$ btmp ($symbols$) $\lor$ ($n \simeq 0$) **then** BTM
else pr-ifn (g-typepn ($symbols$, 1),
pr-ifn (pr-andn (pr-equaln (g-carn ($symbols$), sym),
pr-notn (g-membn (sym,
g-cdrn ($symbols$),
$n - 1$))),
g-cdrn ($symbols$),
g-rest-ofn (sym, g-cdrn ($symbols$), $n - 1$)),
gcode (**nil**)) **endif**

EVENT: Disable g-rest-of; name this event 'g1274'.

EVENT: Disable g-rest-of-defn; name this event 'g1275'.

DEFINITION:
g-rest-of-ind (sym, $symbols$, va, n)
$=$ **if** $n \simeq 0$ **then t**
else g-rest-of-ind (0,
g-cdr (1),
ev (1, list (sym, $symbols$), va, FA, n),
$n - 1$) **endif**

THEOREM: g-rest-of-eval
ev (0, g-rest-of (sym, $symbols$), va, FA, n)
$=$ g-rest-ofn (ev (0, sym, va, FA, n), ev (0, $symbols$, va, FA, n), n)

EVENT: Disable rest-of; name this event 'g1276'.

THEOREM: g-rest-ofn-gcode
((1 + length ($symbols$)) $<$ n)
$\rightarrow$ (g-rest-ofn (gcode (sym), gcode ($symbols$), n)
$=$ gcode (rest-of (sym, $symbols$)))

EVENT: Enable rest-of; name this event 'g1277'.

EVENT: Enable g-rest-of; name this event 'g1278'.

EVENT: Enable g-rest-ofn; name this event 'g1279'.

EVENT: Enable g-rest-of-defn; name this event 'g1280'.

EVENT: Disable g-symb-defn-proof; name this event 'g1281'.

```
;;;;;;;Correspondence for SYMB-DEFN-PROOF.
```

```
;;;;;;;Correspondence for SYMB-DEFN-PROOF.
(DEFN
 G-SYMB-DEFN-PROOFN
 (CONC HINTS SUB-GOALS DEFN SYMBOLS N)
 (PR-IFN
```

```
(G-PREDICATEN (G-CARN (G-CARN HINTS)))
(PR-ANDN (G-SETN (G-CDRN (G-CARN HINTS)) N)
         (PR-ANDN (G-VAR-LISTN (G-CDRN (G-CARN HINTS)) (G-DEGREEN (G-CARN (G-CARN HINTS))) N)
                  (PR-ANDN (PR-EQUALN CONC (G-F-IFFN (G-CARN HINTS) DEFN))
                           (PR-ANDN (G-SUBN (G-COLLECT-FREEN DEFN (GCODE 0) N)
                                            (G-CDRN (G-CARN HINTS))
                                            N)
                                    (G-FORMULAN DEFN
                                                (G-REST-OFN (G-CARN (G-CARN HINTS))
                                                            SYMBOLS
                                                            N)
                                                N)))))
(PR-IFN
 (G-FUNCTIONN (G-CARN (G-CARN HINTS)))
 (PR-ANDN
  (G-SETN (G-CONSN (G-CADDDRN HINTS) (G-CONSN (G-CADRN HINTS) (G-CDRN (G-CARN HINTS)))) N)
  (PR-ANDN
   (G-VAR-LISTN (G-CDRN (G-CARN HINTS)) (G-DEGREEN (G-CARN (G-CARN HINTS))) N)
   (PR-ANDN
    (G-VARIABLEN (G-CADRN HINTS))
    (PR-ANDN
     (G-VARIABLEN (G-CADDDRN HINTS))
     (PR-ANDN
      (G-FORMULAN (G-CADDRN HINTS) (G-REST-OFN (G-CARN (G-CARN HINTS)) SYMBOLS N) N)
      (PR-ANDN
       (G-SUBN (G-COLLECT-FREEN (G-CADDRN HINTS) (GCODE 0) N)
               (G-CONSN (G-CADRN HINTS) (G-CDRN (G-CARN HINTS)))
               N)
       (PR-ANDN
        (G-FREE-FORN (G-CADDRN HINTS) (G-CADRN HINTS) (G-CARN HINTS) (GCODE 0) N)
        (PR-ANDN
         (G-FREE-FORN (G-CADDRN HINTS) (G-CADRN HINTS) (G-CADDDRN HINTS) (GCODE 0) N)
         (PR-ANDN
          (PR-EQUALN CONC
                     (G-SUBSTN (G-CADDRN HINTS)
                               (G-CADRN HINTS)
                               (G-CARN HINTS)
                               (GCODE 0)
                               N))
          (PR-ANDN
           (PR-EQUALN CONC DEFN)
           (PR-EQUALN
            SUB-GOALS
            (G-CONSN (G-FORSOMEN (G-CADRN HINTS) (G-CADDRN HINTS))
```

```
                        (G-CONSN (G-F-IMPLIESN (G-F-ANDN (G-CADDRN HINTS)
                                                         (G-SUBSTN (G-CADDRN HINTS)
                                                                   (G-CADRN HINTS)
                                                                   (G-CADDDRN HINTS)
                                                                   (GCODE 0)
                                                                   N))
                                               (G-F-EQUALN (G-CADRN HINTS)
                                                           (G-CADDDRN HINTS)))
                                 (GCODE NIL)))))))))))))
  (BOOL-FIX F))))
```

THEOREM: g-symb-defn-proof-eval
ev (0, g-symb-defn-proof (*conc*, *hints*, *sub-goals*, *defn*, *symbols*), *va*, FA, n)
= g-symb-defn-proofn (ev (0, *conc*, *va*, FA, n),
ev (0, *hints*, *va*, FA, n),
ev (0, *sub-goals*, *va*, FA, n),
ev (0, *defn*, *va*, FA, n),
ev (0, *symbols*, *va*, FA, n),
n)

THEOREM: lazy-pr-andn-reduc3
pr-andn (bool-fix (x), bool-fix (**f**)) = bool-fix (**f**)

DEFINITION:
g-symb-count (*hints*, *defn*, *symbols*)
= **if** predicate (caar (*hints*))
then length (cdar (*hints*))
+ cddaar (*hints*)
+ exp-count (*defn*)
+ (1 + length (*symbols*))
+ formula-cnt (*defn*, rest-of (caar (*hints*), *symbols*))
else length (cons (fourth (*hints*), cons (second (*hints*), cdar (*hints*))))
+ (1 + length (*symbols*))
+ formula-cnt (third (*hints*),
rest-of (caar (*hints*), *symbols*))
+ cddaar (*hints*)
+ (1 + exp-count (third (*hints*)))
+ (1 + exp-count (second (*hints*)))
+ (1 + exp-count (car (*hints*)))
+ (1 + exp-count (fourth (*hints*))) **endif**

THEOREM: g-symb-defn-proofn-gcode
(g-symb-count (*hints*, *defn*, *symbols*) $<$ n)
$\rightarrow$ (g-symb-defn-proofn (gcode (*conc*),
gcode (*hints*),

gcode (*sub-goals*),
gcode (*defn*),
gcode (*symbols*),
n)

= bool-fix (symb-defn-proof (*conc*, *hints*, *sub-goals*, *defn*, *symbols*)))

EVENT: Enable symb-defn-proof; name this event 'g1290'.

EVENT: Enable g-symb-defn-proof; name this event 'g1291'.

EVENT: Enable g-symb-defn-proofn; name this event 'g1292'.

```
;;;;;;;Correspondence for PRF.
```



```
;;;;;;;Correspondence for PRF.
```



```
(DEFN
 G-PRFN
 (PF GIVEN DEFNS FLG SYMBOLS N)
 (IF
  (OR (ZEROP N)
      (BTMP PF)
      (BTMP GIVEN)
      (BTMP DEFNS)
      (BTMP FLG)
      (BTMP SYMBOLS))
  (BTM)
  (PR-IFN
   (PR-ZEROPN FLG)
   (PR-IFN
    (G-TYPEPN PF 1)
    (PR-IFN
     (PR-NOTN (G-FORMULAN (G-CONCN PF (GCODE 0) (SUB1 N)) SYMBOLS (SUB1 N)))
     (BOOL-FIX F)
     (PR-IFN
      (PR-EQUALN (G-CARN PF) (GCODE 1))
      (G-FOL-AXIOM-PROOFN (G-CONCN PF (GCODE 0) (SUB1 N)) (G-CADRN PF) SYMBOLS (SUB1 N))
      (PR-IFN
       (PR-EQUALN (G-CARN PF) (GCODE 2))
       (PR-ANDN (G-FOL-RULESN (G-CONCN PF (GCODE 0) (SUB1 N))
                              (G-CADRN PF)
                              (G-CONCN (G-CADDDRN PF) (GCODE 1) (SUB1 N))
                              SYMBOLS
```

```
                            (SUB1 N))
              (G-PRFN (G-CADDDRN PF) GIVEN DEFNS (GCODE 1) SYMBOLS (SUB1 N)))
      (PR-IFN (PR-EQUALN (G-CARN PF) (GCODE 3))
              (G-Z2-AXIOMSN (G-CONCN PF (GCODE 0) (SUB1 N)) (G-CADRN PF) SYMBOLS (SUB1 N))
              (PR-IFN (PR-EQUALN (G-CARN PF) (GCODE 4))
                      (PR-ANDN (G-SYMB-DEFN-PROOFN (G-CONCN PF (GCODE 0) (SUB1 N))
                                                   (G-CADRN PF)
                                                   (G-CONCN (G-CADDDRN PF)
                                                            (GCODE 1)
                                                            (SUB1 N))
                                                   (G-ASSOCN (G-CARN (G-CARN (G-CADRN PF)))
                                                             DEFNS
                                                             (SUB1 N))
                                                   SYMBOLS
                                                   (SUB1 N))
                               (PR-IFN (G-FUNCTIONN (G-CARN (G-CARN (G-CADRN PF))))
                                       (G-PRFN (G-CADDDRN PF)
                                               GIVEN
                                               DEFNS
                                               (GCODE 1)
                                               (G-REST-OFN (G-CARN (G-CARN (G-CADRN PF)))
                                                           SYMBOLS
                                                           (SUB1 N))
                                               (SUB1 N))
                                       (BOOL-FIX T)))
                      (PR-IFN (PR-EQUALN (G-CARN PF) (GCODE 5))
                              (G-MEMBN (G-CONCN PF (GCODE 0) (SUB1 N)) GIVEN (SUB1 N))
                              (BOOL-FIX F)))))))
   (BOOL-FIX F))
  (PR-IFN (G-TYPEPN PF 1)
          (PR-ANDN (G-PRFN (G-CARN PF) GIVEN DEFNS (GCODE 0) SYMBOLS (SUB1 N))
                   (G-PRFN (G-CDRN PF) GIVEN DEFNS (GCODE 1) SYMBOLS (SUB1 N)))
          (BOOL-FIX T)))))
```

EVENT: Disable g-prf-defn; name this event 'g1293'.

DEFINITION:
g-prf-ind (*pf*, *given*, *defns*, *flg*, *symbols*, *va*, *n*)
= **if** $n \simeq 0$ **then** **t**
else g-prf-ind (g-cadddr (0),
1,
2,
pr-quote (1),
4,

ev (1, list (*pf*, *given*, *defns*, *flg*, *symbols*), *va*, FA, *n*),
$n - 1$)
$\land$ g-prf-ind (g-cadddr (0),
1,
2,
pr-quote (1),
g-rest-of (g-car (g-car (g-cadr (0))), 4),
ev (1,
list (*pf*, *given*, *defns*, *flg*, *symbols*),
va,
FA,
n),
$n - 1$)
$\land$ g-prf-ind (g-car (0),
1,
2,
pr-quote (0),
4,
ev (1,
list (*pf*, *given*, *defns*, *flg*, *symbols*),
va,
FA,
n),
$n - 1$)
$\land$ g-prf-ind (g-cdr (0),
1,
2,
pr-quote (1),
4,
ev (1,
list (*pf*, *given*, *defns*, *flg*, *symbols*),
va,
FA,
n),
$n - 1$) **endif**

THEOREM: g-prf-eval
ev (0, g-prf (*pf*, *given*, *defns*, *flg*, *symbols*), *va*, FA, *n*)
= g-prfn (ev (0, *pf*, *va*, FA, *n*),
ev (0, *given*, *va*, FA, *n*),
ev (0, *defns*, *va*, FA, *n*),
ev (0, *flg*, *va*, FA, *n*),
ev (0, *symbols*, *va*, FA, *n*),
n)

EVENT: Disable prf; name this event 'g1294'.

EVENT: Enable fol-axiom-cnt; name this event 'g1296'.

EVENT: Enable fol-axiom-cnt; name this event 'g1297'.

DEFINITION:
fol-rules-cnt (*hints*, *sub-goals*, *symbols*)
$=$ (formula-cnt (second (*hints*), *symbols*)
 $+$ formula-cnt (third (*hints*), *symbols*)
 $+$ formula-cnt (fourth (*hints*), *symbols*)
 $+$ formula-cnt (car (*sub-goals*), *symbols*)
 $+$ exp-count (fourth (*hints*)))

THEOREM: g-fol-rulesn-gcode1
(fol-rules-cnt (*hints*, *sub-goals*, *symbols*) $<$ *n*)
$\rightarrow$ (g-fol-rulesn (gcode (*conc*),
 gcode (*hints*),
 gcode (*sub-goals*),
 gcode (*symbols*),
 n)
 $=$ bool-fix (fol-rules (*conc*, *hints*, *sub-goals*, *symbols*)))

DEFINITION:
prf-cnt-part (*pf*, *symbols*)
$=$ ((1 + formula-cnt (conc (*pf*, 0), *symbols*)) + (1 + exp-count (*pf*)))

EVENT: Enable fol-rules-cnt; name this event 'g1298'.

DEFINITION:
prf-ind (*pf*, *given*, *defns*, *flg*, *symbols*, *n*)
$=$ **if** $n \simeq 0$ **then t**
 elseif $flg \simeq 0$
 then if listp (*pf*)
  then if $\neg$ formula (conc (*pf*, 0), *symbols*) **then t**
   elseif car (*pf*) $=$ 1 **then t**
   elseif car (*pf*) $=$ 2
   then prf-ind (cadddr (*pf*), *given*, *defns*, 1, *symbols*, $n - 1$)
   elseif car (*pf*) $=$ 3 **then t**
   elseif car (*pf*) $=$ 4
   then prf-ind (cadddr (*pf*),
    given,

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{defns}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{1}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{rest-of}\,(\text{caar}\,(\text{cadr}\,(\textit{pf})),\ \textit{symbols}), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad n - 1) \\
\qquad\qquad\qquad\qquad \textbf{else t endif} \\
\qquad\qquad \textbf{else t endif} \\
\textbf{elseif}\ \text{listp}\,(\textit{pf}) \\
\textbf{then}\ \text{prf-ind}\,(\text{car}\,(\textit{pf}),\ \textit{given},\ \textit{defns},\ \mathtt{0},\ \textit{symbols},\ n - 1) \\
\qquad \land \quad \text{prf-ind}\,(\text{cdr}\,(\textit{pf}),\ \textit{given},\ \textit{defns},\ \mathtt{1},\ \textit{symbols},\ n - 1) \\
\textbf{else t endif}
\end{array}
$$

EVENT: Enable g-symb-count; name this event ‘g1299’.

THEOREM: g-z2-axiomsn-gcode1

$$
\begin{array}{l}
(\text{z2-axioms-cnt}\,(\textit{hints},\ \textit{symbols}) < n) \\
\rightarrow \quad (\text{g-z2-axiomsn}\,(\text{gcode}\,(\textit{conc}),\ \text{gcode}\,(\textit{hints}),\ \text{gcode}\,(\textit{symbols}),\ n) \\
\qquad = \quad \text{bool-fix}\,(\text{z2-axioms}\,(\textit{conc},\ \textit{hints},\ \textit{symbols})))
\end{array}
$$

EVENT: Enable z2-axioms-cnt; name this event ‘g1300’.

EVENT: Enable g-z2-axiomsn-gcode; name this event ‘g1301’.

EVENT: Enable g-fol-rulesn-gcode; name this event ‘g1302’.

DEFINITION:

$$
\begin{array}{l}
\text{prf-cnt}\,(\textit{pf},\ \textit{given},\ \textit{defns},\ \textit{flg},\ \textit{symbols}) \\
= \quad \textbf{if}\ \textit{flg} \simeq \mathtt{0} \\
\quad\ \ \textbf{then if}\ \text{listp}\,(\textit{pf}) \\
\qquad\quad \textbf{then if}\ \neg\ \text{formula}\,(\text{conc}\,(\textit{pf},\ \mathtt{0}),\ \textit{symbols}) \\
\qquad\qquad \textbf{then}\ \text{prf-cnt-part}\,(\textit{pf},\ \textit{symbols}) \\
\qquad\qquad \textbf{elseif}\ \text{car}\,(\textit{pf}) = \mathtt{1} \\
\qquad\qquad \textbf{then}\ (1 + \text{fol-axiom-cnt}\,(\text{second}\,(\textit{pf}),\ \textit{symbols})) \\
\qquad\qquad\qquad + \quad \text{prf-cnt-part}\,(\textit{pf},\ \textit{symbols}) \\
\qquad\qquad \textbf{elseif}\ \text{car}\,(\textit{pf}) = \mathtt{2} \\
\qquad\qquad \textbf{then}\ \text{prf-cnt-part}\,(\textit{pf},\ \textit{symbols}) \\
\qquad\qquad\qquad + \quad (1 + \text{fol-rules-cnt}\,(\text{second}\,(\textit{pf}), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{conc}\,(\text{cadddr}\,(\textit{pf}),\ \mathtt{1}), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{symbols})) \\
\qquad\qquad\qquad + \quad (1 + \text{prf-cnt}\,(\text{cadddr}\,(\textit{pf}),\ \textit{given},\ \textit{defns},\ \mathtt{1},\ \textit{symbols})) \\
\qquad\qquad \textbf{elseif}\ \text{car}\,(\textit{pf}) = \mathtt{3} \\
\qquad\qquad \textbf{then}\ \text{prf-cnt-part}\,(\textit{pf},\ \textit{symbols}) \\
\qquad\qquad\qquad + \quad (1 + \text{z2-axioms-cnt}\,(\text{second}\,(\textit{pf}),\ \textit{symbols}))
\end{array}
$$

elseif car (pf) = `4`
then prf-cnt-part (pf, $symbols$)
+ (1 + g-symb-count (second (pf),
assoc (caar (cadr (pf)), $defns$),
$symbols$))
+ (1 + (1 + length ($symbols$)))
+ (1 + length ($defns$))
+ (1 + exp-count (fourth (pf)))
+ (1 + prf-cnt (cadddr (pf),
$given$,
$defns$,
`1`,
rest-of (caar (cadr (pf)), $symbols$)))
else prf-cnt-part (pf, $symbols$)
+ (1 + length ($given$)) **endif**
else prf-cnt-part (pf, $symbols$) **endif**
elseif listp (pf)
then 1 + ((1 + prf-cnt (car (pf), $given$, $defns$, `0`, $symbols$))
+ (1 + prf-cnt (cdr (pf), $given$, $defns$, `1`, $symbols$)))
else prf-cnt-part (pf, $symbols$) **endif**

THEOREM: g-prfn-gcode
(prf-cnt (pf, $given$, $defns$, flg, $symbols$) $<$ n)
$\rightarrow$ (g-prfn (gcode (pf),
gcode ($given$),
gcode ($defns$),
gcode (flg),
gcode ($symbols$),
n)
= bool-fix (prf (pf, $given$, $defns$, flg, $symbols$)))

EVENT: Enable g-prf; name this event 'g1303'.

EVENT: Enable g-prfn; name this event 'g1304'.

EVENT: Enable g-prf-defn; name this event 'g1305'.

EVENT: Enable prf-cnt; name this event 'g1306'.

EVENT: Enable prf; name this event 'g1307'.

`;;;;;;;Correspondence for PROVES.`

DEFINITION:
g-provesn (pf, exp, $given$, $defns$, $symbols$, n)
= pr-andn (pr-equaln (g-concn (pf, gcode (0), n), exp),
pr-andn (g-formulan (exp, $symbols$, n),
g-prfn (pf, $given$, $defns$, gcode (0), $symbols$, n)))

EVENT: Disable g-proves; name this event 'g1308'.

THEOREM: g-proves-eval
ev (0, g-proves (pf, exp, $given$, $defns$, $symbols$), va, FA, n)
= g-provesn (ev (0, pf, va, FA, n),
ev (0, exp, va, FA, n),
ev (0, $given$, va, FA, n),
ev (0, $defns$, va, FA, n),
ev (0, $symbols$, va, FA, n),
n)

EVENT: Disable proves; name this event 'g1309'.

THEOREM: g-provesn-gcode
((exp-count (pf)
+ formula-cnt (exp, $symbols$)
+ prf-cnt (pf, $given$, $defns$, 0, $symbols$))
< n)
→ (g-provesn (gcode (pf),
gcode (exp),
gcode ($given$),
gcode ($defns$),
gcode ($symbols$),
n)
= bool-fix (proves (pf, exp, $given$, $defns$, $symbols$)))

EVENT: Enable proves; name this event 'g1310'.

EVENT: Enable g-proves; name this event 'g1311'.

EVENT: Enable g-provesn; name this event 'g1312'.

;;;;;;;Correspondence for NUMERAL.

DEFINITION:
g-numeraln (x, n)

$=$ **if** btmp$(x) \lor (n \simeq 0)$ **then** BTM
else pr-ifn (pr-zeropn(x),
G-PHIN,
g-z-succn (g-numeraln (pr-sub1n(x), $n - 1$))) **endif**

EVENT: Disable g-numeral-defn; name this event 'g1313'.

DEFINITION:
g-numeral-ind(x, va, n)
$=$ **if** $n \simeq 0$ **then** **t**
else g-numeral-ind (pr-sub1 (0), ev (1, list(x), va, FA, n), $n - 1$) **endif**

THEOREM: g-numeral-eval
ev (0, g-numeral(x), va, FA, n) $=$ g-numeraln (ev (0, x, va, FA, n), n)

DEFINITION:
numeral-ind(x, n)
$=$ **if** $n \simeq 0$ **then** **t**
elseif $x \simeq 0$ **then** **t**
else numeral-ind$(x - 1, n - 1)$ **endif**

EVENT: Disable phi; name this event 'g1314'.

EVENT: Disable z-succ; name this event 'g1315'.

EVENT: Disable fn; name this event 'g1316'.

THEOREM: g-numeraln-gcode
$(x < n) \rightarrow$ (g-numeraln (gcode(x), n) $=$ gcode (numeral(x)))

EVENT: Enable phi; name this event 'g1317'.

EVENT: Enable z-succ; name this event 'g1318'.

EVENT: Enable g-numeral; name this event 'g1319'.

EVENT: Enable g-numeraln; name this event 'g1320'.

EVENT: Enable g-numeral-defn; name this event 'g1321'.

DEFINITION: g-z-singn(x) $=$ g-z-pairn(x, x)

DEFINITION:
g-z-opairn $(x,\ y)$ = g-z-pairn (g-z-singn (x), g-z-pairn $(x,\ y)$)

THEOREM: a-ev-good-proof-proves-again
(ev-hyps (*given*, *defns*, *symbols*)
 $\wedge$ variable (*trace*)
 $\wedge$ (*trace* $\notin$ list (6, 7, 8, 9, 10, 11, 12, 15, 16))
 $\wedge$ ($\neg$ btmp (ev (*flg*, *exp*, *va*, *fa*, *n*)))
 $\wedge$ (*concl* = f-implies (a-ev (ncode (*flg*),
 ncode (*exp*),
 ncode (*va*),
 ncode (*fa*),
 10,
 trace),
 f-equal (10, ncode (ev (*flg*, *exp*, *va*, *fa*, *n*))))))
$\rightarrow$ proves (a-ev-good-proof (*flg*, *exp*, *va*, *fa*, *n*, *trace*),
 concl,
 given,
 defns,
 symbols)

DEFINITION:
a-ev1 (*flg-bar*, *exp-bar*, *va-bar*, *fa-bar*, *val-bar*)
= forsome (1, a-ev (*flg-bar*, *exp-bar*, *va-bar*, *fa-bar*, *val-bar*, 1))

EVENT: Enable a-ev; name this event 'g1323'.

DEFINITION:
a-ev1-ok-proof (*flg*, *exp*, *va*, *fa*, *n*)
= eg-proof (a-ev (ncode (*flg*),
 ncode (*exp*),
 ncode (*va*),
 ncode (*fa*),
 ncode (ev (*flg*, *exp*, *va*, *fa*, *n*)),
 1),
 list (1),
 list (graph-trans (ev-graph (*flg*, *exp*, *va*, *fa*, *n*))),
 a-ev-ok-proof (*flg*, *exp*, *va*, *fa*, *n*))

THEOREM: formula-a-ev-defn-again
(variable (*trace*) $\wedge$ ev-hyps (*given*, *defns*, *symbols*))
$\rightarrow$ formula (a-ev-defn (*trace*), *symbols*)

EVENT: Disable a-ev-defn-part1; name this event 'g1324'.

THEOREM: parallel-subst-trace-a-ev-defn-part1
parallel-subst (a-ev-defn-part1 (6, 7, 8, 9, 10, 1), list (1), list (*trace*), 0)
= a-ev-defn-part1 (6, 7, 8, 9, 10, *trace*)

EVENT: Enable a-ev-defn-part1; name this event 'g1325'.

EVENT: Disable a-ev-defn; name this event 'g1326'.

THEOREM: parallel-subst-trace-a-ev-defn
parallel-subst (a-ev-defn (1), list (1), list (*trace*), 0) = a-ev-defn (*trace*)

THEOREM: a-ev1-ok-proof-proves
(ev-hyps (*given*, *defns*, *symbols*)
∧ (¬ btmp (ev (*flg*, *exp*, *va*, *fa*, *n*)))
∧ (*concl* = a-ev1 (ncode (*flg*),
ncode (*exp*),
ncode (*va*),
ncode (*fa*),
ncode (ev (*flg*, *exp*, *va*, *fa*, *n*)))))
→ proves (a-ev1-ok-proof (*flg*, *exp*, *va*, *fa*, *n*), *concl*, *given*, *defns*, *symbols*)

DEFINITION:
a-ev1-good-proof (*flg*, *exp*, *va*, *fa*, *n*)
= forsome-intro-proof (1,
a-ev (ncode (*flg*),
ncode (*exp*),
ncode (*va*),
ncode (*fa*),
10,
1),
f-equal (10, ncode (ev (*flg*, *exp*, *va*, *fa*, *n*))),
a-ev-good-proof (*flg*, *exp*, *va*, *fa*, *n*, 1))

THEOREM: a-ev1-good-proof-proves
(ev-hyps (*given*, *defns*, *symbols*)
∧ (¬ btmp (ev (*flg*, *exp*, *va*, *fa*, *n*)))
∧ (*concl* = f-implies (a-ev1 (ncode (*flg*),
ncode (*exp*),
ncode (*va*),
ncode (*fa*),
10),
f-equal (10, ncode (ev (*flg*, *exp*, *va*, *fa*, *n*))))))
→ proves (a-ev1-good-proof (*flg*, *exp*, *va*, *fa*, *n*), *concl*, *given*, *defns*, *symbols*)

Event: Enable a-ev1-ok-proof; name this event ‘g1327’.

Event: Enable a-ev1-good-proof; name this event ‘g1328’.

Event: Enable free-for; name this event ‘g1329’.

Definition:

```
a-ev1-not-good-proof (flg, exp, va, fa, n, val)
=   tautconseq-proof (list (f-not (f-equal (ncode (val),
                                           ncode (ev (flg, exp, va, fa, n)))),
                           f-implies (a-ev1 (ncode (flg),
                                             ncode (exp),
                                             ncode (va),
                                             ncode (fa),
                                             ncode (val)),
                                      f-equal (ncode (val),
                                               ncode (ev (flg, exp, va, fa, n))))),
                     f-not (a-ev1 (ncode (flg),
                                   ncode (exp),
                                   ncode (va),
                                   ncode (fa),
                                   ncode (val))),
                     list (ncode-nequal-proof (val, ev (flg, exp, va, fa, n)),
                           subst-rule-proof (f-implies (a-ev1 (ncode (flg),
                                                               ncode (exp),
                                                               ncode (va),
                                                               ncode (fa),
                                                               10),
                                                        f-equal (10,
                                                                 ncode (ev (flg,
                                                                            exp,
                                                                            va,
                                                                            fa,
                                                                            n)))),
                                             10,
                                             ncode (val),
                                             a-ev1-good-proof (flg,
                                                               exp,
                                                               va,
                                                               fa,
                                                               n))))
```

Theorem: a-ev1-not-good-proof-proves

$$
\begin{array}{l}
(\text{ev-hyps}\,(\mathit{given},\ \mathit{defns},\ \mathit{symbols}) \\
\ \wedge\quad (\neg\ \text{btmp}\,(\text{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n))) \\
\ \wedge\quad (\neg\ \text{ncode-equal}\,(\mathit{val},\ \text{ev}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n))) \\
\ \wedge\quad (\mathit{concl} = \text{f-not}\,(\text{a-ev1}\,(\text{ncode}\,(\mathit{flg}), \\
\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\mathit{exp}), \\
\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\mathit{va}), \\
\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\mathit{fa}), \\
\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\mathit{val}))))) \\
\rightarrow\quad \text{proves}\,(\text{a-ev1-not-good-proof}\,(\mathit{flg},\ \mathit{exp},\ \mathit{va},\ \mathit{fa},\ n,\ \mathit{val}), \\
\qquad\qquad \mathit{concl}, \\
\qquad\qquad \mathit{given}, \\
\qquad\qquad \mathit{defns}, \\
\qquad\qquad \mathit{symbols})
\end{array}
$$

DEFINITION:

$$
\begin{array}{l}
\text{a-ev2}\,(\mathit{flg},\ \mathit{exp},\ \mathit{fa},\ \mathit{val}) \\
=\quad \text{forsome}\,(\mathtt{1}, \\
\qquad\qquad \text{f-and}\,(\text{isin}\,(\text{z-list}\,(\text{list}\,(\text{ncode}\,(\mathit{val}), \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\mathit{flg}), \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\mathit{exp}), \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{z-opair}\,(\mathtt{6}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{z-opair}\,(\mathtt{7},\ \text{ncode}\,(\text{gcode}\,(\mathbf{nil})))), \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\mathit{fa}))), \\
\qquad\qquad\qquad\qquad \mathtt{1}), \\
\qquad\qquad\qquad \text{a-ev-defn}\,(\mathtt{1})))
\end{array}
$$

THEOREM: replace-va-a-ev2

$$
\begin{array}{l}
\text{subst}\,(\text{subst}\,(\text{a-ev2}\,(\mathit{flg},\ \mathit{exp},\ \mathit{fa},\ \mathit{val}),\ \mathtt{7},\ \text{ncode}\,(\mathit{va2}),\ \mathtt{0}),\ \mathtt{6},\ \text{ncode}\,(\mathit{va1}),\ \mathtt{0}) \\
=\quad \text{a-ev1}\,(\text{ncode}\,(\mathit{flg}), \\
\qquad\qquad \text{ncode}\,(\mathit{exp}), \\
\qquad\qquad \text{z-opair}\,(\text{ncode}\,(\mathit{va1}),\ \text{z-opair}\,(\text{ncode}\,(\mathit{va2}),\ \text{ncode}\,(\text{gcode}\,(\mathbf{nil})))), \\
\qquad\qquad \text{ncode}\,(\mathit{fa}), \\
\qquad\qquad \text{ncode}\,(\mathit{val}))
\end{array}
$$

THEOREM: replace-va-forall-a-ev2

$$
\begin{array}{l}
\text{subst}\,(\forall\ \mathtt{6}\ \text{a-ev2}\,(\mathit{flg},\ \mathit{exp},\ \mathit{fa},\ \mathit{val}),\ \mathtt{7},\ \text{ncode}\,(\mathit{va2}),\ \mathtt{0}) \\
=\quad \forall\ \mathtt{6}\ \text{a-ev1}\,(\text{ncode}\,(\mathit{flg}), \\
\qquad\qquad\quad \text{ncode}\,(\mathit{exp}), \\
\qquad\qquad\quad \text{z-opair}\,(\mathtt{6},\ \text{z-opair}\,(\text{ncode}\,(\mathit{va2}),\ \text{ncode}\,(\text{gcode}\,(\mathbf{nil})))), \\
\qquad\qquad\quad \text{ncode}\,(\mathit{fa}), \\
\qquad\qquad\quad \text{ncode}\,(\mathit{val}))
\end{array}
$$

THEOREM: replace-va1-a-ev2

$$
\begin{array}{l}
\text{subst}\,(\text{a-ev1}\,(\text{ncode}\,(\mathit{flg}), \\
\qquad\qquad\quad \text{ncode}\,(\mathit{exp}),
\end{array}
$$

z-opair (6, z-opair (ncode (*va2*), ncode (gcode (**nil**)))),
ncode (*fa*),
ncode (*val*)),
6,
ncode (*va1*),
0)
= a-ev1 (ncode (*flg*),
ncode (*exp*),
z-opair (ncode (*va1*), z-opair (ncode (*va2*), ncode (gcode (**nil**)))),
ncode (*fa*),
ncode (*val*))

EVENT: Enable a-ev1; name this event 'g1330'.

EVENT: Enable a-ev-defn; name this event 'g1331'.

EVENT: Enable a-ev; name this event 'g1332'.

DEFINITION:
a-ev2-inst-proof (*flg*, *exp*, *fa*, *val*, *va1*, *va2*, *pf*)
= subst-rule-proof (a-ev1 (ncode (*flg*),
ncode (*exp*),
z-opair (6,
z-opair (ncode (*va2*), ncode (gcode (**nil**)))),
ncode (*fa*),
ncode (*val*)),
6,
ncode (*va1*),
detach-proof ($\forall$ 6 a-ev1 (ncode (*flg*),
ncode (*exp*),
z-opair (6,
z-opair (ncode (*va2*),
ncode (gcode (**nil**)))),
ncode (*fa*),
ncode (*val*)),
a-ev1 (ncode (*flg*),
ncode (*exp*),
z-opair (6,
z-opair (ncode (*va2*),
ncode (gcode (**nil**)))),
ncode (*fa*),
ncode (*val*)),
pf,

all-elim-proof (6,
  a-ev1 (ncode (*flg*),
    ncode (*exp*),
    z-opair (6,
      z-opair (ncode (*va2*),
        ncode (gcode (**nil**)))),
    ncode (*fa*),
    ncode (*val*)))))

THEOREM: a-ev2-inst-proof-proves
(ev-hyps (*given*, *defns*, *symbols*)
∧ proves (*pf*,
  ∀ 6 a-ev1 (ncode (*flg*),
    ncode (*exp*),
    z-opair (6, z-opair (ncode (*va2*), ncode (gcode (**nil**)))),
    ncode (*fa*),
    ncode (*val*)),
  given,
  defns,
  symbols)
∧ (*concl* = a-ev1 (ncode (*flg*),
    ncode (*exp*),
    z-opair (ncode (*va1*),
      z-opair (ncode (*va2*), ncode (gcode (**nil**)))),
    ncode (*fa*),
    ncode (*val*))))
→ proves (a-ev2-inst-proof (*flg*, *exp*, *fa*, *val*, *va1*, *va2*, *pf*),
  concl,
  given,
  defns,
  symbols)

EVENT: Disable bool-fix; name this event ‘g1333’.

THEOREM: ncode-nequal-bool-fix-t-f
¬ ncode-equal (bool-fix (**f**), bool-fix (**t**))

EVENT: Enable bool-fix; name this event ‘g1334’.

```
;;;;;;;Exponentiation.
```

DEFINITION:
exp (x, y)
= **if** $y \simeq 0$ **then** 1
  else exp (x, $y - 1$) $*$ x **endif**

```
;;;;;;;Remainder.
```

DEFINITION:
rem (x, y)
$=$ **if** $y \simeq 0$ **then** $\text{fix}(x)$
elseif $x < y$ **then** $\text{fix}(x)$
else rem $(x - y, y)$ **endif**

```
;;;;;;;Development of number theory leading to Unique Prime factorization theorem.
;;;;;;;Taken from Boyer-Moore's "proveall".
```

THEOREM: plus-right-id2
$(y \notin \mathbf{N}) \rightarrow ((x + y) = \text{fix}(x))$

THEOREM: plus-add1
$(x + (1 + y))$
$=$ **if** $y \in \mathbf{N}$ **then** $1 + (x + y)$
else $1 + x$ **endif**

THEOREM: commutativity2-of-plus
$(x + (y + z)) = (y + (x + z))$

THEOREM: commutativity-of-plus
$(x + y) = (y + x)$

THEOREM: associativity-of-plus
$((x + y) + z) = (x + (y + z))$

THEOREM: plus-equal-0
$((a + b) = 0) = ((a \simeq 0) \wedge (b \simeq 0))$

THEOREM: difference-x-x
$(x - x) = 0$

THEOREM: difference-plus
$(((x + y) - x) = \text{fix}(y)) \wedge (((y + x) - x) = \text{fix}(y))$

THEOREM: plus-cancellation
$((a + b) = (a + c)) = (\text{fix}(b) = \text{fix}(c))$

THEOREM: difference-0
$(y \not< x) \rightarrow ((x - y) = 0)$

THEOREM: equal-difference-0
$(0 = (x - y)) = (y \not< x)$

THEOREM: difference-cancellation-0
$(x = (x - y)) = ((x \in \mathbf{N}) \land ((x = 0) \lor (y \simeq 0)))$

THEOREM: difference-cancellation-1
$((x - y) = (z - y))$
$= \;$ **if** $x < y$ **then** $y \not< z$
elseif $z < y$ **then** $y \not< x$
else $\mathrm{fix}(x) = \mathrm{fix}(z)$ **endif**

THEOREM: times-zero2
$(y \notin \mathbf{N}) \rightarrow ((x * y) = 0)$

THEOREM: distributivity-of-times-over-plus
$(x * (y + z)) = ((x * y) + (x * z))$

THEOREM: times-add1
$(x * (1 + y))$
$= \;$ **if** $y \in \mathbf{N}$ **then** $x + (x * y)$
else $\mathrm{fix}(x)$ **endif**

THEOREM: commutativity-of-times
$(x * y) = (y * x)$

THEOREM: commutativity2-of-times
$(x * (y * z)) = (y * (x * z))$

THEOREM: associativity-of-times
$((x * y) * z) = (x * (y * z))$

THEOREM: equal-times-0
$((x * y) = 0) = ((x \simeq 0) \lor (y \simeq 0))$

THEOREM: equal-lessp
$((x < y) = z)$
$= \;$ **if** $x < y$ **then** $\mathbf{t} = z$
else $\mathbf{f} = z$ **endif**

THEOREM: difference-elim
$((y \in \mathbf{N}) \land (y \not< x)) \rightarrow ((x + (y - x)) = y)$

THEOREM: rem-quotient
$(\mathrm{rem}(x, y) + (y * (x \div y))) = \mathrm{fix}(x)$

THEOREM: rem-wrt-1
$\mathrm{rem}(y, 1) = 0$

THEOREM: rem-wrt-12
$(x \notin \mathbf{N}) \rightarrow (\mathrm{rem}\,(y,\ x) = \mathrm{fix}\,(y))$

THEOREM: lessp-rem2
$(\mathrm{rem}\,(x,\ y) < y) = (y \not\simeq 0)$

THEOREM: rem-x-x
$\mathrm{rem}\,(x,\ x) = 0$

THEOREM: rem-quotient-elim
$((y \not\simeq 0) \wedge (x \in \mathbf{N})) \rightarrow ((\mathrm{rem}\,(x,\ y) + (y * (x \div y))) = x)$

THEOREM: lessp-times-1
$(i \not\simeq 0) \rightarrow ((i * j) \not< j)$

THEOREM: lessp-times-2
$(i \not\simeq 0) \rightarrow ((j * i) \not< j)$

THEOREM: lessp-quotient1
$((i \div j) < i) = ((i \not\simeq 0) \wedge ((j \simeq 0) \vee (j \neq 1)))$

THEOREM: lessp-rem1
$(\mathrm{rem}\,(x,\ y) < x) = ((y \not\simeq 0) \wedge (x \not\simeq 0) \wedge (x \not< y))$

DEFINITION:
$\mathrm{gcd}\,(x,\ y)$
$=$ **if** $x \simeq 0$ **then** $\mathrm{fix}\,(y)$
elseif $y \simeq 0$ **then** x
elseif $x < y$ **then** $\mathrm{gcd}\,(x,\ y - x)$
else $\mathrm{gcd}\,(x - y,\ y)$ **endif**

THEOREM: numberp-gcd
$\mathrm{gcd}\,(x,\ y) \in \mathbf{N}$

THEOREM: gcd-equal-0
$(\mathrm{gcd}\,(x,\ y) = 0) = ((x \simeq 0) \wedge (y \simeq 0))$

THEOREM: gcd-0
$\mathrm{gcd}\,(0,\ y) = \mathrm{fix}\,(y)$

THEOREM: commutativity-of-gcd
$\mathrm{gcd}\,(x,\ y) = \mathrm{gcd}\,(y,\ x)$

THEOREM: difference-plus1
$((x + y) - x) = \mathrm{fix}\,(y)$

THEOREM: difference-plus2
$((y + x) - x) = \text{fix}(y)$

THEOREM: difference-plus-cancelation
$((x + y) - (x + z)) = (y - z)$

THEOREM: times-difference
$(x * (c - w)) = ((c * x) - (w * x))$

DEFINITION: $\text{divides}(x, y) = (\text{rem}(y, x) \simeq \mathsf{0})$

THEOREM: divides-times
$\text{rem}(x * z, z) = \mathsf{0}$

THEOREM: difference-plus3
$((b + (a + c)) - a) = (b + c)$

THEOREM: difference-add1-cancellation
$((1 + (y + z)) - z) = (1 + y)$

THEOREM: rem-add1
$((y \not\simeq \mathsf{0}) \wedge (y \neq \mathsf{1})) \rightarrow (\text{rem}(1 + (x * y), y) \neq \mathsf{0})$

THEOREM: divides-plus-rewrite1
$((\text{rem}(x, z) = \mathsf{0}) \wedge (\text{rem}(y, z) = \mathsf{0})) \rightarrow (\text{rem}(x + y, z) = \mathsf{0})$

THEOREM: divides-plus-rewrite2
$((\text{rem}(x, z) = \mathsf{0}) \wedge (\text{rem}(y, z) \neq \mathsf{0})) \rightarrow (\text{rem}(x + y, z) \neq \mathsf{0})$

THEOREM: divides-plus-rewrite
$(\text{rem}(x, z) = \mathsf{0}) \rightarrow ((\text{rem}(x + y, z) = \mathsf{0}) = (\text{rem}(y, z) = \mathsf{0}))$

THEOREM: lessp-plus-cancelation
$((x + y) < (x + z)) = (y < z)$

THEOREM: divides-plus-rewrite-commuted
$(\text{rem}(x, z) = \mathsf{0}) \rightarrow ((\text{rem}(y + x, z) = \mathsf{0}) = (\text{rem}(y, z) = \mathsf{0}))$

THEOREM: euclid
$$\begin{array}{l} (\text{rem}(x, z) = \mathsf{0}) \\ \rightarrow \quad ((\text{rem}(y - x, z) = \mathsf{0}) \\ \qquad = \quad \textbf{if } x < y \textbf{ then } \text{rem}(y, z) = \mathsf{0} \\ \qquad\qquad \textbf{else t endif}) \end{array}$$

THEOREM: lessp-times-cancellation
$((x * z) < (y * z)) = ((z \not\simeq \mathsf{0}) \wedge (x < y))$

THEOREM: lessp-plus-cancellation3
$(y < (x + y)) = (x \not\simeq 0)$

THEOREM: distributivity-of-times-over-gcd
$\text{gcd}\,(x * z,\ y * z) = (z * \text{gcd}\,(x,\ y))$

THEOREM: gcd-divides-both
$(\text{rem}\,(x,\ \text{gcd}\,(x,\ y)) = 0) \land (\text{rem}\,(y,\ \text{gcd}\,(x,\ y)) = 0)$

THEOREM: gcd-is-the-greatest
$((x \not\simeq 0) \land (y \not\simeq 0) \land \text{divides}\,(z,\ x) \land \text{divides}\,(z,\ y))$
$\rightarrow \quad (z \leq \text{gcd}\,(x,\ y))$

DEFINITION:
$\text{prime1}\,(x,\ y)$
$= \quad$ **if** $y \simeq 0$ **then f**
elseif $y = 1$ **then t**
else $(\neg\ \text{divides}\,(y,\ x)) \land \text{prime1}\,(x,\ y - 1)$ **endif**

DEFINITION:
$\text{prime}\,(x) = ((x \not\simeq 0) \land (x \neq 1) \land \text{prime1}\,(x,\ x - 1))$

DEFINITION:
$\text{greatest-factor}\,(x,\ y)$
$= \quad$ **if** $(y \simeq 0) \lor (y = 1)$ **then** x
elseif $\text{divides}\,(y,\ x)$ **then** y
else $\text{greatest-factor}\,(x,\ y - 1)$ **endif**

THEOREM: greatest-factor-lessp
$((y < x)$
$\land \quad (\neg\ \text{prime1}\,(x,\ y))$
$\land \quad (x \not\simeq 0)$
$\land \quad ((x - 1) \neq 0)$
$\land \quad (y \not\simeq 0))$
$\rightarrow \quad (\text{greatest-factor}\,(x,\ y) < x)$

THEOREM: greatest-factor-divides
$((y < x) \land (\neg\ \text{prime1}\,(x,\ y)) \land (x \not\simeq 0) \land (x \neq 1) \land (y \not\simeq 0))$
$\rightarrow \quad (\text{rem}\,(x,\ \text{greatest-factor}\,(x,\ y)) = 0)$

THEOREM: greatest-factor-0
$(\text{greatest-factor}\,(x,\ y) = 0) = (((y \simeq 0) \lor (y = 1)) \land (x = 0))$

THEOREM: rem-0-crock
$\text{rem}\,(0,\ y) = 0$

THEOREM: greatest-factor-1
(greatest-factor $(x,\ y) = 1) = (x = 1)$

THEOREM: numberp-greatest-factor
(greatest-factor $(x,\ y) \in \mathbf{N}) = (\neg\ (((y \simeq 0) \vee (y = 1)) \wedge (x \notin \mathbf{N})))$

DEFINITION:
prime-factors (x)
$=$ **if** $(x \simeq 0) \vee ((x - 1) = 0)$ **then nil**
elseif prime1 $(x,\ x - 1)$ **then** cons $(x,$ **nil**$)$
else append (prime-factors (greatest-factor $(x,\ x - 1)$),
prime-factors $(x \div$ greatest-factor $(x,\ x - 1)))$ **endif**

DEFINITION:
prime-list (l)
$=$ **if** $l \simeq$ **nil then t**
else prime (car (l)) $\wedge$ prime-list (cdr (l)) **endif**

DEFINITION:
times-list (l)
$=$ **if** $l \simeq$ **nil then** 1
else car $(l) *$ times-list (cdr (l)) **endif**

THEOREM: times-list-append
times-list (append $(x,\ y)$) $=$ (times-list $(x) *$ times-list (y))

THEOREM: prime-list-append
prime-list (append $(x,\ y)$) $=$ (prime-list $(x) \wedge$ prime-list (y))

THEOREM: prime-list-prime-factors
prime-list (prime-factors (x))

THEOREM: quotient-times1
$((y \in \mathbf{N}) \wedge (x \in \mathbf{N}) \wedge (x \neq 0) \wedge$ divides $(x,\ y))$
$\rightarrow \quad ((x * (y \div x)) = y)$

THEOREM: quotient-lessp
$((x \not\simeq 0) \wedge (x < y)) \rightarrow ((y \div x) \neq 0)$

THEOREM: enough-factors
$(x \not\simeq 0) \rightarrow$ (times-list (prime-factors (x)) $= x$)

THEOREM: prime-factorization-existence
$(x \not\simeq 0)$
$\rightarrow$ ((times-list (prime-factors (x)) $= x$)
$\wedge$ prime-list (prime-factors (x)))

DEFINITION:
greatereqpr (w, z)
$=$ **if** $w \simeq \mathsf{0}$ **then** $z \simeq \mathsf{0}$
elseif $w = z$ **then** $\mathbf{t}$
else greatereqpr $(w - 1, z)$ **endif**

THEOREM: times-id-iff-1
$(z = (w * z)) = ((z \in \mathbf{N}) \wedge ((z = \mathsf{0}) \vee (w = \mathsf{1})))$

THEOREM: prime1-basic
$((z \neq \mathsf{1}) \wedge (z \neq \mathsf{0}) \wedge (z \in \mathbf{N}) \wedge \text{greatereqpr}(u, z))$
$\rightarrow \quad (\neg\ \text{prime1}(w * z, u))$

THEOREM: greatereqpr-lessp
$\text{greatereqpr}(x, y) = (x \not< y)$

THEOREM: greatereqpr-rem
$((z \neq (1 + v)) \wedge \text{divides}(z, 1 + v)) \rightarrow \text{greatereqpr}(v, z)$

THEOREM: prime-basic
$((z \neq \mathsf{1}) \wedge (z \neq x) \wedge (x \not\simeq \mathsf{0}) \wedge (x \neq \mathsf{1}) \wedge \text{divides}(z, x))$
$\rightarrow \quad (\neg\ \text{prime1}(x, x - 1))$

THEOREM: rem-gcd
$(\text{gcd}(b, x) = y) \rightarrow (\text{rem}(b, y) = \mathsf{0})$

THEOREM: rem-gcd-1
$(\text{rem}(b, x) \neq \mathsf{0}) \rightarrow (\text{gcd}(b, x) \neq x)$

THEOREM: divides-times1
$(a = (z * y)) \rightarrow (\text{rem}(a, z) = \mathsf{0})$

THEOREM: times-identity1
$((y \in \mathbf{N}) \wedge (y \neq \mathsf{1}) \wedge (y \neq \mathsf{0}) \wedge (x \neq \mathsf{0})) \rightarrow (x \neq (x * y))$

THEOREM: times-identity
$(x = (x * y)) = ((x = \mathsf{0}) \vee ((x \in \mathbf{N}) \wedge (y = \mathsf{1})))$

THEOREM: kludge-bridge
$(y = (k * x)) \rightarrow (\text{gcd}(y, a * x) = (x * \text{gcd}(a, k)))$

THEOREM: hack1
$((\neg\ \text{divides}(x, a)) \wedge (a = \text{gcd}(x * a, b * a)))$
$\rightarrow \quad ((k * x) \neq (b * a))$

THEOREM: prime-gcd
$((\neg\ \text{divides}(x, b)) \wedge (x \not\simeq \mathsf{0}) \wedge ((x - 1) \neq \mathsf{0}) \wedge \text{prime1}(x, x - 1))$
$\rightarrow \quad ((\text{gcd}(b, x) = \mathsf{1}) = \mathbf{t})$

THEOREM: gcd-distributes-over-an-opened-up-times
$((x \in \mathbf{N}) \land (x \neq 0) \land (\mathit{free} = (x * z)))$
$\rightarrow \quad (\mathrm{gcd}\,(b * z,\ \mathit{free}) = (z * \mathrm{gcd}\,(b,\ x)))$

THEOREM: prime-key
$((z \in \mathbf{N}) \land \mathrm{prime}\,(x) \land (\neg\ \mathrm{divides}\,(x,\ z)) \land (\neg\ \mathrm{divides}\,(x,\ b)))$
$\rightarrow \quad ((x * k) \neq (b * z))$

THEOREM: quotient-divides
$((y \in \mathbf{N}) \land ((x * (y \div x)) \neq y)) \rightarrow (\mathrm{rem}\,(y,\ x) \neq 0)$

THEOREM: little-step
$(\mathrm{prime}\,(x) \land (y \neq 1) \land (x \neq y)) \rightarrow (\mathrm{rem}\,(x,\ y) \neq 0)$

DEFINITION:
delete $(x,\ y)$
= **if** listp (y)
then if $x = \mathrm{car}\,(y)$ **then** cdr (y)
else cons (car (y), delete $(x$, cdr $(y)))$ **endif**
else y **endif**

THEOREM: delete-non-member
$(x \notin y) \rightarrow (\mathrm{delete}\,(x,\ y) = y)$

THEOREM: member-delete
$(x \in \mathrm{delete}\,(u,\ v)) \rightarrow (x \in v)$

THEOREM: lessp-count-delete
$(n \in l) \rightarrow (\mathrm{count}\,(\mathrm{delete}\,(n,\ l)) < \mathrm{count}\,(l))$

DEFINITION:
perm $(a,\ b)$
= **if** $a \simeq$ **nil** **then** $b \simeq$ **nil**
elseif car $(a) \in b$ **then** perm (cdr (a), delete (car (a), b))
else f endif

THEOREM: rem-times
$\mathrm{rem}\,(y * x,\ y) = 0$

THEOREM: prime-list-delete
prime-list $(l2) \rightarrow$ prime-list (delete $(x,\ l2)$)

THEOREM: divides-times-list
$((c \not\simeq 0) \land (c \in l)) \rightarrow (\mathrm{rem}\,(\text{times-list}\,(l),\ c) = 0)$

THEOREM: quotient-times
$((y * x) \div y)$
$= \quad$ **if** $y \simeq 0$ **then** 0
else $\text{fix}(x)$ **endif**

THEOREM: distributivity-of-divides
$((a \not\simeq 0) \land \text{divides}(a, w)) \rightarrow ((c * (w \div a)) = ((c * w) \div a))$

THEOREM: times-list-delete
$((c \not\simeq 0) \land (c \in l))$
$\rightarrow \quad (\text{times-list}(\text{delete}(c, l)) = (\text{times-list}(l) \div c))$

THEOREM: prime-list-times-list
$(\text{prime}(c) \land \text{prime-list}(l2) \land (c \notin l2))$
$\rightarrow \quad (\text{rem}(\text{times-list}(l2), c) \neq 0)$

THEOREM: if-times-then-divides
$((c \not\simeq 0) \land (\neg \text{divides}(c, x))) \rightarrow ((c * y) \neq x)$

THEOREM: times-equal-1
$((a * b) = 1)$
$= \quad ((a \neq 0)$
$\land \quad (b \neq 0)$
$\land \quad (a \in \mathbf{N})$
$\land \quad (b \in \mathbf{N})$
$\land \quad ((a - 1) = 0)$
$\land \quad ((b - 1) = 0))$

THEOREM: prime-member
$(((c * \text{times-list}(l1)) = \text{times-list}(l2)) \land \text{prime}(c) \land \text{prime-list}(l2))$
$\rightarrow \quad (c \in l2)$

THEOREM: divides-implies-times
$((a \not\simeq 0) \land (c \in \mathbf{N}) \land ((a * c) = b)) \rightarrow ((c = (b \div a)) = \mathbf{t})$

`;;;;;;;Two prime factorizations with the same product are permutations of one another.`

THEOREM: prime-factorization-uniqueness
$(\text{prime-list}(l1) \land \text{prime-list}(l2) \land (\text{times-list}(l1) = \text{times-list}(l2)))$
$\rightarrow \quad \text{perm}(l1, l2)$

`;;;;;;;NUM-TREEP checks if X is constructed from numbers using CONS.`

DEFINITION:
$\text{num-treep}(x)$
$= \quad$ **if** $\text{listp}(x)$ **then** $\text{num-treep}(\text{car}(x)) \land \text{num-treep}(\text{cdr}(x))$
else $x \in \mathbf{N}$ **endif**

```
;;;;;;;DOUBLE multiplies X by 2.  The recursive definition is there to provide an induction
;;;;;;;scheme.
```

DEFINITION:
double (x)
$=$ **if** $x \simeq 0$ **then** 0
else $1 + (1 + \text{double}(x - 1))$ **endif**

```
;;;;;;;HALVE integer-divides X by 2.
```

DEFINITION:
halve (x)
$=$ **if** $x \simeq 0$ **then** 0
elseif $(x - 1) \simeq 0$ **then** 0
else $1 + \text{halve}((x - 1) - 1)$ **endif**

THEOREM: odd-rem-two
$\text{rem}(1 + \text{double}(x), 2) = 1$

```
;;;;;;;NUMCODE is a one-to-one mapping from num-trees to numbers.  A number N is mapped
;;;;;;;to the odd number (2N + 1).  The list (CONS X Y) is mapped to ((x**2)(y**3)),
;;;;;;;if X is mapped to x, and Y to y.  The unique prime factorization theorem is
;;;;;;;needed to decipher the num-tree back from a NUMCODE.
```

DEFINITION:
numcode (x)
$=$ **if** listp (x) **then** $\text{exp}(2, \text{numcode}(\text{car}(x))) * \text{exp}(3, \text{numcode}(\text{cdr}(x)))$
else $1 + \text{double}(x)$ **endif**

DEFINITION:
y-x-list (x, y)
$=$ **if** $y \simeq 0$ **then nil**
else cons $(x, \text{y-x-list}(x, y - 1))$ **endif**

THEOREM: exp-times-list-y-x-list
times-list (y-x-list (x, y)) $= \text{exp}(x, y)$

DEFINITION:
perm-y-x-ind (y, z)
$=$ **if** $y \simeq 0$ **then t**
elseif $z \simeq 0$ **then t**
else perm-y-x-ind $(y - 1, z - 1)$ **endif**

THEOREM: perm-y-x-list-append
perm (append (y-x-list (x, y), *list1*), append (y-x-list (x, z), *list2*))
$=$ **if** $y < z$ **then** perm (*list1*, append (y-x-list $(x, z - y)$, *list2*))
else perm (append (y-x-list $(x, y - z)$, *list1*), *list2*) **endif**

THEOREM: not-member-y-x-list
$(x \neq x1) \rightarrow (x \not\in \text{y-x-list}(x1,\ y1))$

THEOREM: nequal-perm-y-x-list
$((x \neq x1) \wedge (y \not\simeq 0))$
$\rightarrow \quad (\neg\ \text{perm}(\text{append}(\text{y-x-list}(x,\ y),\ list1),\ \text{y-x-list}(x1,\ y1)))$

THEOREM: nequal-perm-y-x-step
$(\text{listp}(y) \wedge (\text{car}(y) \not\in x)) \rightarrow (\neg\ \text{perm}(x,\ y))$

THEOREM: nequal-perm-y-x-list-another
$((x \neq x1) \wedge (y1 \not\simeq 0))$
$\rightarrow \quad (\neg\ \text{perm}(\text{y-x-list}(x,\ y),\ \text{append}(\text{y-x-list}(x1,\ y1),\ list1)))$

THEOREM: not-perm-numcode
$((x \not\simeq 0) \wedge (x \neq x1))$
$\rightarrow \quad (\neg\ \text{perm}(\text{append}(\text{y-x-list}(2,\ x),\ \text{y-x-list}(3,\ y)),$
$\qquad\qquad \text{append}(\text{y-x-list}(2,\ x1),\ \text{y-x-list}(3,\ y1))))$

THEOREM: prime-list-y-x-list
$\text{prime}(x) \rightarrow \text{prime-list}(\text{y-x-list}(x,\ y))$

THEOREM: rem-rec-reduc
$((y \not\simeq 0) \wedge (y < x)) \rightarrow (\text{rem}(x,\ y) = \text{rem}(x - y,\ y))$

THEOREM: rem32
$\text{rem}(3,\ 2) = 1$

THEOREM: prime1-3-2
$\text{prime1}(3,\ 2) = (\neg\ \text{divides}(2,\ 3))$

THEOREM: prime-3
$\text{prime}(3)$

THEOREM: prime-2
$\text{prime}(2)$

THEOREM: times-list-append23-numcode
$(\exp(2,\ x) * \exp(3,\ y))$
$= \quad \text{times-list}(\text{append}(\text{y-x-list}(2,\ x),\ \text{y-x-list}(3,\ y)))$

`;;;;;;;The NUMCODEs of two lists are distinct, if the NUMCODEs of the their first elements are.`

THEOREM: numcode-uniqueness1
$((x \neq x1) \wedge (x \not\simeq 0))$
$\rightarrow \quad ((\exp(2,\ x) * \exp(3,\ y)) \neq (\exp(2,\ x1) * \exp(3,\ y1)))$

EVENT: Enable times-list-append23-numcode; name this event 'g1335'.

THEOREM: perm-append-x-x
perm (append (x, y), append (x, $y1$)) = perm (y, $y1$)

THEOREM: length-delete
length (delete (x, y))
= **if** $x \in y$ **then** length (y) $-$ 1
else length (y) **endif**

THEOREM: member-length
(length (y) $\simeq$ 0) $\rightarrow$ ($x \notin y$)

THEOREM: perm-length
(length (x) $\neq$ length (y)) $\rightarrow$ ($\neg$ perm (x, y))

THEOREM: length-y-x-list
length (y-x-list (x, y)) = fix (y)

THEOREM: not-perm-nequal
(($y \neq y1$) $\wedge$ ($y \not\simeq 0$)) $\rightarrow$ ($\neg$ perm (y-x-list (x, y), y-x-list (x, $y1$)))

`;;;;;;;The NUMCODEs of two lists are distinct, if the NUMCODEs of their second elements are.`

THEOREM: numcode-uniqueness2
(($x = x1$) $\wedge$ ($y \not\simeq 0$) $\wedge$ ($y \neq y1$))
$\rightarrow$ ((exp (2, x) $*$ exp (3, y)) $\neq$ (exp (2, $x1$) $*$ exp (3, $y1$)))

DEFINITION:
double-ind (x, y)
= **if** $x \simeq 0$ **then t**
elseif $y \simeq 0$ **then t**
else double-ind ($x - 1$, $y - 1$) **endif**

THEOREM: double-nequal-add1-double
double (x) $\neq$ (1 + double (y))

THEOREM: times-2-double
(2 $*$ x) = double (x)

THEOREM: plus-double
(double (x) + double (y)) = double ($x + y$)

THEOREM: times-double
(x $*$ double (y)) = double ($x * y$)

THEOREM: numcode-reduc
$(y \not\simeq 0) \rightarrow ((\exp(2, y) * z) = \text{double}(\exp(2, y - 1) * z))$

THEOREM: numcode-nequal
$(x1 \not\simeq 0) \rightarrow ((1 + \text{double}(x)) \neq (\exp(2, x1) * y))$

EVENT: Enable numcode-reduc; name this event 'g1336'.

EVENT: Enable times-2-double; name this event 'g1337'.

EVENT: Enable times-double; name this event 'g1338'.

```
;;;;;;;DECIPHER1 is the function that deciphers the num-tree back from a NUMCODE.
;;;;;;;It essentially starts from Y and Z, and works its way down lexicographically
;;;;;;;until it has decomposed X into a pair of NUMCODEs, which it recursively deciphers.
;;;;;;;In the base case, when X is not the NUMCODE of any pair, it halves (SUB1 X).
```

DEFINITION:
decipher1 (x, y, z)
= **if** $y \simeq 0$ **then** halve $(x - 1)$
elseif $z \simeq 0$ **then** decipher1 $(x, y - 1, x)$
elseif $(y < x) \wedge (z < x)$
then if $x = (\exp(2, y) * \exp(3, z))$
then cons (decipher1 $(y, y - 1, y - 1)$, decipher1 $(z, z - 1, z - 1)$)
else decipher1 $(x, y, z - 1)$ **endif**
else decipher1 $(x, y, z - 1)$ **endif**

THEOREM: lessp-exp
$((y \not\simeq 0) \wedge (1 < x)) \rightarrow (y < \exp(x, y))$

THEOREM: lessp-times
$((1 < x) \wedge (1 < y)) \rightarrow ((x < (x * y)) \wedge (y < (x * y)))$

THEOREM: lessp-numcode1
$((x \not\simeq 0) \wedge (y \not\simeq 0))$
$\rightarrow \quad ((x < (\exp(2, x) * \exp(3, y)))$
$\quad \wedge \quad (y < (\exp(2, x) * \exp(3, y))))$

EVENT: Enable times; name this event 'g1339'.

EVENT: Enable exp; name this event 'g1340'.

THEOREM: decipher-listp

$$((x \not\simeq 0) \land (y \not\simeq 0)) \rightarrow \quad (\text{decipher1}(\exp(2,\ x) * \exp(3,\ y),\ x,\ y) = \text{cons}(\text{decipher1}(x,\ x-1,\ x-1),\ \text{decipher1}(y,\ y-1,\ y-1)))$$

DEFINITION:

decipher-ind $(x,\ y,\ x1,\ y1)$

$=$ **if** $x1 \simeq 0$ **then t**
elseif $y1 \simeq 0$ **then** decipher-ind $(x,\ y,\ x1-1,\ \exp(2,\ x) * \exp(3,\ y))$
elseif $x < x1$ **then** decipher-ind $(x,\ y,\ x1,\ y1-1)$
elseif $y = y1$ **then t**
else decipher-ind $(x,\ y,\ x1,\ y1-1)$ **endif**

THEOREM: decipher1-listp-reduc

$$((x \not\simeq 0) \land (y \not\simeq 0) \land (x \leq x1) \land \textbf{if } x = x1 \textbf{ then } y \leq y1 \textbf{ else t endif}) \rightarrow (\text{decipher1}(\exp(2,\ x) * \exp(3,\ y),\ x1,\ y1) = \text{decipher1}(\exp(2,\ x) * \exp(3,\ y),\ x,\ y))$$

THEOREM: halve-double

$\text{halve}(\text{double}(x)) = \text{fix}(x)$

THEOREM: lessp-numcode

$0 < \text{numcode}(x)$

```
;;;;;;;NUMTREEP is the same as NUM-TREEP.
```

DEFINITION:

numtreep (x)

$=$ **if** listp (x) **then** numtreep $(\text{car}(x)) \land$ numtreep $(\text{cdr}(x))$
else $x \in \mathbf{N}$ **endif**

THEOREM: lessp-sub1-numcode

$$((x \not\simeq 0) \land (y \not\simeq 0)) \rightarrow \quad ((((\exp(2,\ x) * \exp(3,\ y)) - 1) \not< x) \land (((\exp(2,\ x) * \exp(3,\ y)) - 1) \not< y))$$

THEOREM: lessp-numcode-again

$\text{numcode}(x) \neq 0$

THEOREM: lessp-numcode-case

$((x \not\simeq 0) \land (y \not\simeq 0)) \rightarrow (x \neq ((\exp(2,\ x) * \exp(3,\ y)) - 1))$

DEFINITION:
decipher-ind2 (x, y, z)
= **if** $y \simeq 0$ **then t**
elseif $z \simeq 0$ **then** decipher-ind2 $(x, y - 1, 1 + \text{double}(x))$
else decipher-ind2 $(x, y, z - 1)$ **endif**

THEOREM: decipher1-nlistp
decipher1 $(1 + \text{double}(x), y, z) = \text{fix}(x)$

```
;;;;;;;This is the main lemma relating DECIPHER1 and NUMCODE.  It asserts that
;;;;;;;DECIPHER1 applied to (NUMCODE X) (with appropriate values for the Y, Z, arguments)
;;;;;;;returns X.
```

THEOREM: decipher1-numcode
numtreep $(x) \rightarrow (\text{decipher1}(\text{numcode}(x), \text{numcode}(x) - 1, \text{numcode}(x) - 1) = x)$

```
;;;;;;;Correspondence proofs for PLUS.
```

DEFINITION:
g-plusn (x, y, n)
= **if** btmp $(x) \lor$ btmp $(y) \lor (n \simeq 0)$ **then** BTM
else pr-ifn (pr-zeropn (x),
g-fixn (y),
pr-add1n (g-plusn (pr-sub1n (x), y, $n - 1$))) **endif**

EVENT: Disable g-plus; name this event ‘g1341’.

EVENT: Disable g-plus-defn; name this event ‘g1342’.

DEFINITION:
g-plus-ind (x, y, va, n)
= **if** $n \simeq 0$ **then t**
else g-plus-ind (pr-sub1 (0), 1, ev (1, list (x, y), va, FA, n), $n - 1$) **endif**

THEOREM: g-plus-eval
ev (0, g-plus (x, y), va, FA, n) = g-plusn (ev (0, x, va, FA, n), ev (0, y, va, FA, n), n)

THEOREM: pr-add1n-gcode
pr-add1n (gcode (x)) = gcode $(1 + x)$

EVENT: Enable pr-add1n; name this event ‘g1343’.

DEFINITION:
plusn-ind (x, y, n)
= **if** $n \simeq 0$ **then t**
elseif $x \simeq 0$ **then t**
else plusn-ind $(x - 1, y, n - 1)$ **endif**

THEOREM: g-plusn-gcode
$(x < n) \rightarrow (\text{g-plusn}\,(\text{gcode}\,(x),\ \text{gcode}\,(y),\ n) = \text{gcode}\,(x + y))$

EVENT: Enable g-plus; name this event 'g1344'.

EVENT: Enable g-plusn; name this event 'g1345'.

EVENT: Enable g-plus-ind; name this event 'g1346'.

```
;;;;;;;Correspondence proofs for TIMES.
```

DEFINITION:
g-timesn $(x,\ y,\ n)$
$=$ **if** btmp $(x) \vee$ btmp $(y) \vee (n \simeq \mathtt{0})$ **then** BTM
else pr-ifn (pr-zeropn (x),
gcode $(\mathtt{0})$,
g-plusn $(y$, g-timesn (pr-sub1n (x), y, $n - 1)$, $n - 1)$) **endif**

EVENT: Disable g-times; name this event 'g1347'.

EVENT: Disable g-times-defn; name this event 'g1348'.

THEOREM: g-times-eval
ev $(\mathtt{0}$, g-times $(x,\ y)$, va, FA, $n)$ = g-timesn (ev $(\mathtt{0},\ x,\ va$, FA, $n)$, ev $(\mathtt{0},\ y,\ va$, FA, $n)$, $n)$

THEOREM: g-timesn-gcode
$((1 + (x + y)) < n)$
$\rightarrow$ (g-timesn (gcode (x), gcode (y), n) = gcode $(x * y)$)

EVENT: Enable g-times; name this event 'g1349'.

EVENT: Enable g-timesn; name this event 'g1350'.

EVENT: Enable g-times-defn; name this event 'g1351'.

```
;;;;;;;Correspondence proofs for EXP.
```

DEFINITION:
g-expn $(x,\ y,\ n)$
$=$ **if** btmp $(x) \vee$ btmp $(y) \vee (n \simeq \mathtt{0})$ **then** BTM
else pr-ifn (pr-zeropn (y),
gcode $(\mathtt{1})$,
g-timesn $(x$, g-expn $(x$, pr-sub1n (y), $n - 1)$, $n - 1)$) **endif**

EVENT: Disable g-exp; name this event 'g1373'.

EVENT: Disable g-exp-defn; name this event 'g1374'.

DEFINITION:
g-exp-ind $(x,\ y,\ va,\ n)$
$=$ **if** $n \simeq 0$ **then t**
else g-exp-ind (0, pr-sub1 (1), ev (1, list $(x,\ y)$, va, FA, n), $n - 1$) **endif**

THEOREM: g-exp-eval
ev (0, g-exp $(x,\ y)$, va, FA, n) $=$ g-expn (ev (0, x, va, FA, n), ev (0, y, va, FA, n), n)

EVENT: Disable exp; name this event 'g1375'.

EVENT: Disable times; name this event 'g1376'.

THEOREM: g-expn-gcode
$((x \not\simeq 0) \land ((1 + (x + y + \exp(x,\ y))) < n))$
$\rightarrow$ (g-expn (gcode (x), gcode (y), n) $=$ gcode (exp $(x,\ y)$))

EVENT: Enable g-exp; name this event 'g1377'.

EVENT: Enable g-expn; name this event 'g1378'.

EVENT: Enable g-exp-defn; name this event 'g1379'.

EVENT: Enable times; name this event 'g1380'.

EVENT: Enable exp; name this event 'g1381'.

```
;;;;;;;Correspondence proofs for HALVE.
```

DEFINITION:
g-halven $(x,\ n)$
$=$ **if** btmp $(x) \lor (n \simeq 0)$ **then** BTM
else pr-ifn (pr-zeropn (x),
gcode (0),
pr-ifn (pr-zeropn (pr-sub1n (x)),
gcode (0),
pr-add1n (g-halven (pr-sub1n (pr-sub1n (x)),
$n - 1$)))) **endif**

EVENT: Disable g-halve; name this event 'g1382'.

EVENT: Disable g-halve-defn; name this event 'g1383'.

DEFINITION:
g-halve-ind $(x,\ va,\ n)$
$=$ **if** $n \simeq 0$ **then t**
else g-halve-ind (pr-sub1 (pr-sub1 (0)),
ev (1, list (x), va, FA, n),
$n - 1$) **endif**

THEOREM: g-halve-eval
ev (0, g-halve (x), va, FA, n) $=$ g-halven (ev (0, x, va, FA, n), n)

DEFINITION:
halven-ind $(x,\ n)$
$=$ **if** $n \simeq 0$ **then t**
elseif $x \simeq 0$ **then t**
else halven-ind $((x - 1) - 1,\ n - 1)$ **endif**

THEOREM: g-halven-gcode
$(x < n) \rightarrow$ (g-halven (gcode (x), n) $=$ gcode (halve (x)))

EVENT: Enable g-halve; name this event 'g1384'.

EVENT: Enable g-halven; name this event 'g1385'.

EVENT: Enable g-halve-defn; name this event 'g1386'.

```
;;;;;;;Correspondence proofs for LESSP.
```

DEFINITION:
g-lesspn $(x,\ y,\ n)$
$=$ **if** btmp (x) $\vee$ btmp (y) $\vee$ $(n \simeq 0)$ **then** BTM
else pr-ifn (pr-zeropn (y),
bool-fix (**f**),
pr-ifn (pr-zeropn (x),
bool-fix (**t**),
g-lesspn (pr-sub1n (x), pr-sub1n (y), $n - 1$))) **endif**

EVENT: Disable g-lessp; name this event 'g1387'.

EVENT: Disable g-lessp-defn; name this event 'g1388'.

Definition:
g-lessp-ind $(x,\ y,\ va,\ n)$
$=$ **if** $n \simeq \texttt{0}$ **then t**
else g-lessp-ind (pr-sub1 $(\texttt{0})$,
pr-sub1 $(\texttt{1})$,
ev $(\texttt{1}$, list $(x,\ y),\ va,\ \textsc{fa},\ n)$,
$n - 1)$ **endif**

Theorem: g-lessp-eval
ev $(\texttt{0}$, g-lessp $(x,\ y),\ va,\ \textsc{fa},\ n)$ = g-lesspn (ev $(\texttt{0},\ x,\ va,\ \textsc{fa},\ n)$, ev $(\texttt{0},\ y,\ va,\ \textsc{fa},\ n),\ n)$

Definition:
lesspn-ind $(x,\ y,\ n)$
$=$ **if** $n \simeq \texttt{0}$ **then t**
elseif $y \simeq \texttt{0}$ **then t**
elseif $x \simeq \texttt{0}$ **then t**
else lesspn-ind $(x - 1,\ y - 1,\ n - 1)$ **endif**

Theorem: g-lesspn-gcode
$((x + y) < n) \rightarrow$ (g-lesspn (gcode (x), gcode (y), n) = bool-fix $(x < y)$)

Event: Enable g-lessp; name this event 'g1389'.

Event: Enable g-lesspn; name this event 'g1390'.

Event: Enable g-lessp-defn; name this event 'g1391'.

```
;;;;;;;Correspondence proofs for DECIPHER1.
```

Definition:
g-decipher1n $(x,\ y,\ z,\ n)$
$=$ **if** btmp (x) $\vee$ btmp (y) $\vee$ btmp (z) $\vee$ $(n \simeq \texttt{0})$ **then** btm
else pr-ifn (pr-zeropn (y),
g-halven (pr-sub1n (x), $n - 1$),
pr-ifn (pr-zeropn (z),
g-decipher1n $(x$, pr-sub1n (y), $x,\ n - 1)$,
pr-ifn (pr-andn (g-lesspn $(y,\ x,\ n - 1)$,
g-lesspn $(z,\ x,\ n - 1)$),
pr-ifn (pr-equaln $(x$,
g-timesn (g-expn (gcode $(\texttt{2})$,
y,
$n - 1)$,
g-expn (gcode $(\texttt{3})$,

```
                                          z,
                                          n − 1),
                                    n − 1)),
                  pr-consn (g-decipher1n (y,
                                          pr-sub1n (y),
                                          pr-sub1n (y),
                                          n − 1),
                            g-decipher1n (z,
                                          pr-sub1n (z),
                                          pr-sub1n (z),
                                          n − 1)),
                  g-decipher1n (x,
                                y,
                                pr-sub1n (z),
                                n − 1)),
            g-decipher1n (x, y, pr-sub1n (z), n − 1)))) endif
```

EVENT: Disable g-decipher1; name this event ‘g1392’.

EVENT: Disable g-decipher1-defn; name this event ‘g1393’.

DEFINITION:

```
g-decipher1-ind (x, y, z, va, n)
=    if n ≃ 0 then t
     else g-decipher1-ind (0,
                           pr-sub1 (1),
                           0,
                           ev (1, list (x, y, z), va, FA, n),
                           n − 1)
          ∧  g-decipher1-ind (0,
                              1,
                              pr-sub1 (2),
                              ev (1, list (x, y, z), va, FA, n),
                              n − 1)
          ∧  g-decipher1-ind (1,
                              pr-sub1 (1),
                              pr-sub1 (1),
                              ev (1, list (x, y, z), va, FA, n),
                              n − 1)
          ∧  g-decipher1-ind (2,
                              pr-sub1 (2),
                              pr-sub1 (2),
                              ev (1, list (x, y, z), va, FA, n),
```

$n - 1)$ **endif**

THEOREM: g-decipher1-eval
ev (0, g-decipher1 (x, y, z), va, FA, n)
= g-decipher1n (ev (0, x, va, FA, n), ev (0, y, va, FA, n), ev (0, z, va, FA, n), n)

DEFINITION:
dec-count (x, y, z)
= **if** $y \simeq 0$ **then** $1 + x$
elseif $z \simeq 0$ **then** $1 +$ dec-count $(x, y - 1, x)$
elseif $(y < x) \wedge (z < x)$
then if $x = (\exp(2, y) * \exp(3, z))$
then $1 + (1 + (x$
$+ \ y$
$+ \ z$
$+ \ \exp(2, y)$
$+ \ \exp(3, z)$
$+$ dec-count $(y, y - 1, y - 1)$
$+$ dec-count $(z, z - 1, z - 1)))$
else $1 + (1 + (x$
$+ \ y$
$+ \ z$
$+ \ \exp(2, y)$
$+ \ \exp(3, z)$
$+$ dec-count $(x, y, z - 1)))$ **endif**
else $1 + (1 + (x$
$+ \ y$
$+ \ z$
$+ \ \exp(2, y)$
$+ \ \exp(3, z)$
$+$ dec-count $(x, y, z - 1)))$ **endif**

THEOREM: gcode-halve
gcode (halve (x)) = halve (x)

DEFINITION:
decipher1n-ind (x, y, z, n)
= **if** $n \simeq 0$ **then t**
elseif $y \simeq 0$ **then t**
elseif $z \simeq 0$ **then** decipher1n-ind $(x, y - 1, x, n - 1)$
elseif $(y < x) \wedge (z < x)$
then if $x = (\exp(2, y) * \exp(3, z))$
then decipher1n-ind $(y, y - 1, y - 1, n - 1)$
$\wedge$ decipher1n-ind $(z, z - 1, z - 1, n - 1)$
else decipher1n-ind $(x, y, z - 1, n - 1)$ **endif**
else decipher1n-ind $(x, y, z - 1, n - 1)$ **endif**

THEOREM: g-decipher1n-gcode
$(\text{dec-count}\,(x,\ y,\ z) < n)$
$\rightarrow \quad (\text{g-decipher1n}\,(\text{gcode}\,(x),\ \text{gcode}\,(y),\ \text{gcode}\,(z),\ n) = \text{decipher1}\,(x,\ y,\ z))$

EVENT: Enable g-decipher1; name this event 'g1394'.

EVENT: Enable g-decipher1n; name this event 'g1395'.

EVENT: Enable g-decipher1-defn; name this event 'g1396'.

THEOREM: num-treep-gcode
num-treep (gcode (x))

DEFINITION:
g-deciphern $(x,\ n)$ = g-decipher1n $(x$, pr-sub1n (x), pr-sub1n (x), $n)$

```
;;;;;;;Correspondence proofs for INV-CODE.
```

DEFINITION:
g-inv-coden $(x,\ n)$

$$
\begin{array}{l}
= \quad \textbf{if } \text{btmp}\,(x) \lor (n \simeq \texttt{0}) \textbf{ then } \textsc{btm} \\
\quad\quad \textbf{else } \text{pr-ifn}\,(\text{pr-listpn}\,(x), \\
\quad\quad\quad\quad \text{pr-ifn}\,(\text{pr-equaln}\,(\text{pr-carn}\,(x),\ \texttt{1}), \\
\quad\quad\quad\quad\quad \text{g-consn}\,(\text{g-inv-coden}\,(\text{pr-carn}\,(\text{pr-cdrn}\,(x)), \\
\quad\quad\quad\quad\quad\quad\quad n - 1), \\
\quad\quad\quad\quad\quad\quad \text{g-inv-coden}\,(\text{pr-cdrn}\,(\text{pr-cdrn}\,(x)), \\
\quad\quad\quad\quad\quad\quad\quad n - 1)), \\
\quad\quad\quad\quad\quad \text{pr-ifn}\,(\text{pr-equaln}\,(\text{pr-carn}\,(x),\ \texttt{2}), \\
\quad\quad\quad\quad\quad\quad \text{g-f-notn}\,(\text{g-inv-coden}\,(\text{pr-carn}\,(\text{pr-cdrn}\,(x)), \\
\quad\quad\quad\quad\quad\quad\quad n - 1)), \\
\quad\quad\quad\quad\quad\quad \text{pr-ifn}\,(\text{pr-equaln}\,(\text{pr-carn}\,(x),\ \texttt{3}), \\
\quad\quad\quad\quad\quad\quad\quad \text{g-f-orn}\,(\text{g-inv-coden}\,(\text{pr-carn}\,(\text{pr-cdrn}\,(x)), \\
\quad\quad\quad\quad\quad\quad\quad\quad n - 1), \\
\quad\quad\quad\quad\quad\quad\quad\quad \text{g-inv-coden}\,(\text{pr-cdrn}\,(\text{pr-cdrn}\,(x)), \\
\quad\quad\quad\quad\quad\quad\quad\quad n - 1)), \\
\quad\quad\quad\quad\quad\quad\quad \text{pr-ifn}\,(\text{pr-equaln}\,(\text{pr-carn}\,(x),\ \texttt{4}), \\
\quad\quad\quad\quad\quad\quad\quad\quad \text{g-forsomen}\,(\text{g-inv-coden}\,(\text{pr-carn}\,(\text{pr-cdrn}\,(x)), \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad n - 1), \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{g-inv-coden}\,(\text{pr-cdrn}\,(\text{pr-cdrn}\,(x)), \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad n - 1)), \\
\quad\quad\quad\quad\quad\quad\quad\quad \text{pr-ifn}\,(\text{pr-equaln}\,(\text{pr-carn}\,(x), \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \texttt{5}), \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{gcode}\,(\mathbf{t}),
\end{array}
$$

pr-ifn (pr-equaln (pr-carn (x),
6),
gcode (**f**),
gcode (**nil**))))))),
g-fixn (x)) **endif**

EVENT: Disable g-inv-code; name this event 'g1398'.

EVENT: Disable g-inv-code-defn; name this event 'g1399'.

DEFINITION:
g-inv-code-ind (x, va, n)
= **if** $n \simeq 0$ **then t**
else g-inv-code-ind (pr-car (pr-cdr (0)),
ev (1, list (x), va, FA, n),
$n - 1$)
$\wedge$ g-inv-code-ind (pr-cdr (pr-cdr (0)),
ev (1, list (x), va, FA, n),
$n - 1$) **endif**

THEOREM: g-inv-code-eval
ev (0, g-inv-code (x), va, FA, n) = g-inv-coden (ev (0, x, va, FA, n), n)

THEOREM: pr-carn-reduc1
$(x \in \mathbf{N}) \rightarrow$ (pr-carn (x) = 0)

THEOREM: pr-carn-reduc2
pr-carn (cons (x, y)) = x

THEOREM: pr-cdrn-reduc1
$(x \in \mathbf{N}) \rightarrow$ (pr-cdrn (x) = 0)

THEOREM: pr-cdrn-reduc
pr-cdrn (cons (x, y)) = y

THEOREM: pr-listp-reduc1
$(x \in \mathbf{N}) \rightarrow$ (pr-listpn (x) = bool-fix (**f**))

THEOREM: pr-listp-reduc2
pr-listpn (cons (x, y)) = bool-fix (**t**)

DEFINITION:
inv-code (x)
= **if** listp (x)
then if car (x) = 1 **then** cons (inv-code (cadr (x)), inv-code (cddr (x)))

elseif car (x) = 2 **then** f-not (inv-code (cadr (x)))
elseif car (x) = 3
then f-or (inv-code (cadr (x)), inv-code (cddr (x)))
elseif car (x) = 4
then forsome (inv-code (cadr (x)), inv-code (cddr (x)))
elseif car (x) = 5 **then t**
elseif car (x) = 6 **then f**
else nil endif
else fix (x) **endif**

DEFINITION:
inv-coden-ind (x, n)
= **if** $n \simeq 0$ **then t**
elseif listp (x)
then if car (x) = 1
then inv-coden-ind (cadr (x), $n - 1$)
$\wedge$ inv-coden-ind (cddr (x), $n - 1$)
elseif car (x) = 2 **then** inv-coden-ind (cadr (x), $n - 1$)
elseif car (x) = 3
then inv-coden-ind (cadr (x), $n - 1$)
$\wedge$ inv-coden-ind (cddr (x), $n - 1$)
elseif car (x) = 4
then inv-coden-ind (cadr (x), $n - 1$)
$\wedge$ inv-coden-ind (cddr (x), $n - 1$)
else t endif
else t endif

EVENT: Disable g-fixn; name this event 'g1400'.

THEOREM: g-fixn-reduc
$(x \in \mathbf{N}) \rightarrow$ (g-fixn (x) = gcode (x))

EVENT: Enable g-fixn; name this event 'g1401'.

EVENT: Disable pr-equaln; name this event 'g1402'.

THEOREM: pr-equaln-reduc2
(($\neg$ btmp (x)) $\wedge$ ($\neg$ btmp (y)))
$\rightarrow$ (pr-equaln (x, y) = bool-fix (ncode-equal (x, y)))

EVENT: Enable pr-equaln; name this event 'g1403'.

EVENT: Disable g-consn; name this event 'g1404'.

EVENT: Disable g-f-notn; name this event 'g1405'.

EVENT: Disable g-f-orn; name this event 'g1406'.

EVENT: Disable g-forsomen; name this event 'g1407'.

THEOREM: g-consn-numberp
$((x \in \mathbf{N}) \wedge (y \in \mathbf{N})) \rightarrow (\text{g-consn}(x, y) = \text{gcode}(\text{cons}(x, y)))$

THEOREM: g-f-notn-numberp
$(x \in \mathbf{N}) \rightarrow (\text{g-f-notn}(x) = \text{gcode}(\text{f-not}(x)))$

THEOREM: g-f-orn-numberp
$((x \in \mathbf{N}) \wedge (y \in \mathbf{N})) \rightarrow (\text{g-f-orn}(x, y) = \text{gcode}(\text{f-or}(x, y)))$

THEOREM: g-forsomen-numberp
$((x \in \mathbf{N}) \wedge (y \in \mathbf{N})) \rightarrow (\text{g-forsomen}(x, y) = \text{gcode}(\text{forsome}(x, y)))$

EVENT: Enable g-consn; name this event 'g1408'.

EVENT: Enable g-f-notn; name this event 'g1409'.

EVENT: Enable g-f-orn; name this event 'g1410'.

THEOREM: pr-equaln-reduc3
$(\text{num-treep}(x) \wedge \text{num-treep}(y))$
$\rightarrow \quad (\text{pr-equaln}(x, y) = \text{bool-fix}(\text{ncode-equal}(x, y)))$

THEOREM: g-forsomen-numberp2
$(x \in \mathbf{N}) \rightarrow (\text{g-forsomen}(x, \text{gcode}(y)) = \text{gcode}(\text{forsome}(x, y)))$

EVENT: Enable g-forsomen; name this event 'g1411'.

THEOREM: g-inv-coden-gcode
$(\text{num-treep}(x) \wedge (\text{exp-count}(x) < n))$
$\rightarrow \quad (\text{g-inv-coden}(x, n) = \text{gcode}(\text{inv-code}(x)))$

THEOREM: inv-code-gcode
$\text{gcode}(\text{inv-code}(\text{gcode}(x))) = \text{gcode}(x)$

EVENT: Enable g-inv-code; name this event 'g1412'.

EVENT: Enable g-inv-coden; name this event 'g1413'.

EVENT: Enable g-inv-code-defn; name this event 'g1414'.

```
;;;;;;;Correspondence proofs for the "theorem-checker".  There is no Lisp function
;;;;;;;ISTHM (pronounced "is theorem") because it would have to search upwards
;;;;;;;along the enumeration of proofs given by (NUMCODE (GCODE PF)), until it found a
;;;;;;;a proof or a disproof of EXP.  This search might not terminate.
;;;;;;;The function G-ISTHMN however has a stack-counter N which is decremented with
;;;;;;;each recursive call, and hence does terminate.
```

DEFINITION:
g-isthmn (pfn, exp, $given$, $defns$, $symbols$, n)
= **if** btmp (pfn)
  $\lor$ btmp (exp)
  $\lor$ btmp ($given$)
  $\lor$ btmp ($defns$)
  $\lor$ btmp ($symbols$)
  $\lor$ ($n \simeq 0$) **then** BTM
  else pr-ifn (g-provesn (g-inv-coden (g-deciphern (pfn, $n - 1$), $n - 1$),
    exp,
    $given$,
    $defns$,
    $symbols$,
    $n - 1$),
  bool-fix (**t**),
  pr-ifn (g-provesn (g-inv-coden (g-deciphern (pfn, $n - 1$),
      $n - 1$),
    g-f-notn (exp),
    $given$,
    $defns$,
    $symbols$,
    $n - 1$),
  bool-fix (**f**),
  g-isthmn (pr-add1n (pfn),
    exp,
    $given$,
    $defns$,
    $symbols$,
    $n - 1$))) **endif**

EVENT: Disable g-isthm; name this event 'g1415'.

EVENT: Disable g-isthm-defn; name this event ‘g1416’.

DEFINITION:
g-thm-ind (*pfn*, *exp*, *given*, *defns*, *symbols*, *va*, *n*)
= **if** $n \simeq$ `0` **then** **t**
  else g-thm-ind (pr-add1 (`0`),
    `1`,
    `2`,
    `3`,
    `4`,
    ev (`1`, list (*pfn*, *exp*, *given*, *defns*, *symbols*), *va*, FA, *n*),
    $n - 1$) **endif**

THEOREM: g-isthm-eval
ev (`0`, g-isthm (*pfn*, *exp*, *given*, *defns*, *symbols*), *va*, FA, *n*)
= g-isthmn (ev (`0`, *pfn*, *va*, FA, *n*),
  ev (`0`, *exp*, *va*, FA, *n*),
  ev (`0`, *given*, *va*, FA, *n*),
  ev (`0`, *defns*, *va*, FA, *n*),
  ev (`0`, *symbols*, *va*, FA, *n*),
  n)

```
;;;;;;;Analogous to G-ISTHMN, we can define G-FIND-PROOFN which locates the number of the
;;;;;;;proof/disproof at which G-ISTHMN terminated.
```

DEFINITION:
g-find-proofn (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*)
= **if** btmp (*pfn*)
  ∨ btmp (*exp*)
  ∨ btmp (*given*)
  ∨ btmp (*defns*)
  ∨ btmp (*symbols*)
  ∨ ($n \simeq$ `0`) **then** BTM
  else pr-ifn (g-provesn (g-inv-coden (g-deciphern (*pfn*, $n - 1$), $n - 1$),
    exp,
    given,
    defns,
    symbols,
    $n - 1$),
   pfn,
   pr-ifn (g-provesn (g-inv-coden (g-deciphern (*pfn*, $n - 1$),
     $n - 1$),
    g-f-notn (*exp*),
    given,

```
                         defns,
                         symbols,
                         n - 1),
                pfn,
                g-find-proofn (pr-add1n (pfn),
                               exp,
                               given,
                               defns,
                               symbols,
                               n - 1))) endif
```

`;;;;;;;DECIPHER is defined in terms of DECIPHER1.`

DEFINITION: decipher (x) = decipher1 $(x,\ x - 1,\ x - 1)$

DEFINITION:

```
find-proofn (pfn, exp, given, defns, symbols, n)
=   if btmp (exp)
      ∨ btmp (given)
      ∨ btmp (defns)
      ∨ btmp (symbols)
      ∨ (n ≃ 0) then BTM
    else pr-ifn (g-provesn (g-inv-coden (g-deciphern (gcode (pfn), n - 1),
                                         n - 1),
                            exp,
                            given,
                            defns,
                            symbols,
                            n - 1),
                 pfn,
                 pr-ifn (g-provesn (g-inv-coden (g-deciphern (gcode (pfn),
                                                              n - 1),
                                                 n - 1),
                                    g-f-notn (exp),
                                    given,
                                    defns,
                                    symbols,
                                    n - 1),
                         pfn,
                         find-proofn (1 + pfn,
                                      exp,
                                      given,
                                      defns,
                                      symbols,
                                      n - 1))) endif
```

Definition:
find-proof-ind $(pfn,\ n)$
$=$ **if** $n \simeq 0$ **then t**
else find-proof-ind $(1 + pfn,\ n - 1)$ **endif**

Event: Disable pr-ifn; name this event 'g1417'.

Event: Disable bool-fix; name this event 'g1418'.

Event: Disable pr-andn; name this event 'g1419'.

Theorem: pr-andn-t-or-f1
$((\neg \text{ btmp}(\text{pr-andn}(x,\ y))) \wedge (\text{pr-andn}(x,\ y) \neq \text{bool-fix}(\mathbf{f})))$
$\rightarrow \quad (\text{pr-andn}(x,\ y) = \text{bool-fix}(\mathbf{t}))$

Theorem: pr-andn-t-or-f2
$((\neg \text{ btmp}(\text{pr-andn}(x,\ y))) \wedge (\text{pr-andn}(x,\ y) \neq \text{bool-fix}(\mathbf{t})))$
$\rightarrow \quad (\text{pr-andn}(x,\ y) = \text{bool-fix}(\mathbf{f}))$

Event: Enable pr-andn; name this event 'g1420'.

Event: Disable g-provesn; name this event 'g1421'.

```
;;;;;;;If G-PROVESN terminates, it returns either (BOOL-FIX T) or (BOOL-FIX F).
```

Theorem: g-provesn-t-or-f1
$((\neg \text{ btmp}(\text{g-provesn}(pf,\ exp,\ given,\ defns,\ symbols,\ n)))$
$\wedge \quad (\text{g-provesn}(pf,\ exp,\ given,\ defns,\ symbols,\ n) \neq \text{bool-fix}(\mathbf{f})))$
$\rightarrow \quad (\text{g-provesn}(pf,\ exp,\ given,\ defns,\ symbols,\ n) = \text{bool-fix}(\mathbf{t}))$

Theorem: g-provesn-t-or-f2
$((\neg \text{ btmp}(\text{g-provesn}(pf,\ exp,\ given,\ defns,\ symbols,\ n)))$
$\wedge \quad (\text{g-provesn}(pf,\ exp,\ given,\ defns,\ symbols,\ n) \neq \text{bool-fix}(\mathbf{t})))$
$\rightarrow \quad (\text{g-provesn}(pf,\ exp,\ given,\ defns,\ symbols,\ n) = \text{bool-fix}(\mathbf{f}))$

Event: Enable g-provesn; name this event 'g1422'.

Event: Enable bool-fix; name this event 'g1423'.

Theorem: g-find-proofn-gcode
$(\neg \text{ btmp}(\text{g-isthmn}(\text{gcode}(pfn),\ exp,\ given,\ defns,\ symbols,\ n)))$
$\rightarrow \quad (\text{g-find-proofn}(\text{gcode}(pfn),\ exp,\ given,\ defns,\ symbols,\ n)$
$\quad = \quad \text{gcode}(\text{find-proofn}(pfn,\ exp,\ given,\ defns,\ symbols,\ n)))$

EVENT: Enable pr-ifn; name this event ‘g1424’.

```
;;;;;;;DECIPHER1 always returns a num-tree.
```

THEOREM: num-treep-decipher1
num-treep (decipher1 (x, y, z))

```
;;;;;;;The lemma below essentially says that G-INV-CODEN returns the GCODE of whatever
;;;;;;;INV-CODE returns.
```

THEOREM: inv-code-decipher
((dec-count (x, $x - 1$, $x - 1$) + exp-count (decipher (x))) $<$ n)
$\rightarrow$ (g-inv-coden (g-deciphern (gcode (x), n), n)
$=$ gcode (inv-code (decipher (x))))

EVENT: Enable g-inv-coden; name this event ‘g1425’.

EVENT: Enable g-deciphern; name this event ‘g1426’.

EVENT: Enable inv-code; name this event ‘g1427’.

EVENT: Enable decipher; name this event ‘g1428’.

```
;;;;;;;G-FINDN returns the value of stack counter N at the point at which G-ISTHMN located
;;;;;;;a proof/disproof.
```

DEFINITION:
g-findn (*pfn*, *exp*, *given*, *defns*, *symbols*, n)
= **if** btmp (*pfn*)
∨ btmp (*exp*)
∨ btmp (*given*)
∨ btmp (*defns*)
∨ btmp (*symbols*)
∨ ($n \simeq 0$) **then** BTM
else pr-ifn (g-provesn (g-inv-coden (g-deciphern (*pfn*, $n - 1$), $n - 1$),
exp,
given,
defns,
symbols,
$n - 1$),
$n - 1$,
pr-ifn (g-provesn (g-inv-coden (g-deciphern (*pfn*, $n - 1$),

```
                                          n − 1),
                                 g-f-notn (exp),
                                 given,
                                 defns,
                                 symbols,
                                 n − 1),
                       n − 1,
                       g-findn (pr-add1n (pfn),
                               exp,
                               given,
                               defns,
                               symbols,
                               n − 1))) endif
```

```
;;;;;;;G-FIND-PROOF returns the GCODE of the proof/disproof.
```

DEFINITION:

```
g-find-proof (pfn, exp, given, defns, symbols, n)
=   if btmp (pfn)
       ∨  btmp (exp)
       ∨  btmp (given)
       ∨  btmp (defns)
       ∨  btmp (symbols)
       ∨  (n ≃ 0)  then BTM
    else pr-ifn (g-provesn (g-inv-coden (g-deciphern (pfn, n − 1), n − 1),
                            exp,
                            given,
                            defns,
                            symbols,
                            n − 1),
                 g-inv-coden (g-deciphern (pfn, n − 1), n − 1),
                 pr-ifn (g-provesn (g-inv-coden (g-deciphern (pfn, n − 1),
                                                 n − 1),
                                    g-f-notn (exp),
                                    given,
                                    defns,
                                    symbols,
                                    n − 1),
                         g-inv-coden (g-deciphern (pfn, n − 1), n − 1),
                         g-find-proof (pr-add1n (pfn),
                                       exp,
                                       given,
                                       defns,
                                       symbols,
                                       n − 1))) endif
```

EVENT: Disable pr-ifn; name this event 'g1429'.

EVENT: Disable bool-fix; name this event 'g1430'.

```
;;;;;;;If G-ISTHMN returns (BOOL-FIX T), then G-FIND-PROOF returns the GCODE of a proof.
```

THEOREM: find-proof-proves-t-case
(g-isthmn (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*) = bool-fix (**t**))
$\rightarrow$ (g-provesn (g-find-proof (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*),
exp,
given,
defns,
symbols,
g-findn (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*))
= bool-fix (**t**))

```
;;;;;;;If G-ISTHMN returns (BOOL-FIX F), then G-FIND-PROOF returns the GCODE of a disproof.
```

THEOREM: find-proof-proves-f-case
(g-isthmn (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*) = bool-fix (**f**))
$\rightarrow$ ((g-provesn (g-find-proof (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*),
exp,
given,
defns,
symbols,
g-findn (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*))
= bool-fix (**f**))
$\land$ (g-provesn (g-find-proof (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*),
g-f-notn (*exp*),
given,
defns,
symbols,
g-findn (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*))
= bool-fix (**t**)))

```
;;;;;;;This lemma relates G-FIND-PROOF and G-FIND-PROOFN (which returns the proof number).
```

THEOREM: g-find-proof-gcode
($\neg$ btmp (g-isthmn (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*)))
$\rightarrow$ (g-find-proof (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*)
= g-inv-coden (g-deciphern (g-find-proofn (*pfn*,
exp,
given,
defns,

symbols,
n),
g-findn (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*)),
g-findn (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*)))

EVENT: Enable g-find-proof; name this event 'g1431'.

EVENT: Enable g-find-proofn; name this event 'g1432'.

EVENT: Enable g-findn; name this event 'g1433'.

EVENT: Enable find-proofn; name this event 'g1434'.

EVENT: Enable dec-count; name this event 'g1435'.

DEFINITION: dec-countn (x) = dec-count (x, $x - 1$, $x - 1$)

DEFINITION:
thm-count (*pfn*, *exp*, *var*, *term*, *flg*, *given*, *defns*, *symbols*)
= ((dec-countn (*pfn*) + exp-count (decipher (*pfn*)))
+ exp-count (inv-code (decipher (*pfn*)))
+ formula-cnt (subst (*exp*, *var*, *term*, *flg*), *symbols*)
+ prf-cnt (inv-code (decipher (*pfn*)), *given*, *defns*, 0, *symbols*))

```
;;;;;;;The same lemmas as above with PROVES, INV-CODE, and DECIPHER, instead of
;;;;;;;G-PROVESN, G-INV-CODEN, and G-DECIPHERN.
```

THEOREM: find-proof-proves-t-case-gcode
(((1 + thm-count (find-proofn (*pfn*,
gcode (subst (*exp*, *var*, *term*, *flg*)),
gcode (*given*),
gcode (*defns*),
gcode (*symbols*),
n),
exp,
var,
term,
flg,
given,
defns,
symbols))
< g-findn (gcode (*pfn*),

$$
\begin{array}{l}
\quad\quad\quad \text{gcode}\,(\text{subst}\,(exp,\ var,\ term,\ flg)),\\
\quad\quad\quad \text{gcode}\,(given),\\
\quad\quad\quad \text{gcode}\,(defns),\\
\quad\quad\quad \text{gcode}\,(symbols),\\
\quad\quad\quad n))\\
\land\quad (\text{g-isthmn}\,(\text{gcode}\,(pfn),\\
\quad\quad\quad\quad \text{gcode}\,(\text{subst}\,(exp,\ var,\ term,\ flg)),\\
\quad\quad\quad\quad \text{gcode}\,(given),\\
\quad\quad\quad\quad \text{gcode}\,(defns),\\
\quad\quad\quad\quad \text{gcode}\,(symbols),\\
\quad\quad\quad\quad n)\\
\quad = \quad \text{bool-fix}\,(\mathbf{t})))\\
\to\quad \text{proves}\,(\text{inv-code}\,(\text{decipher}\,(\text{find-proofn}\,(pfn,\\
\quad\quad\quad\quad\quad\quad\quad\quad \text{gcode}\,(\text{subst}\,(exp,\ var,\ term,\ flg)),\\
\quad\quad\quad\quad\quad\quad\quad\quad \text{gcode}\,(given),\\
\quad\quad\quad\quad\quad\quad\quad\quad \text{gcode}\,(defns),\\
\quad\quad\quad\quad\quad\quad\quad\quad \text{gcode}\,(symbols),\\
\quad\quad\quad\quad\quad\quad\quad\quad n))),\\
\quad\quad\quad \text{subst}\,(exp,\ var,\ term,\ flg),\\
\quad\quad\quad given,\\
\quad\quad\quad defns,\\
\quad\quad\quad symbols)
\end{array}
$$

Theorem: find-proof-proves-f-case-case

$$
\begin{array}{l}
(\text{g-isthmn}\,(pfn,\ exp,\ given,\ defns,\ symbols,\ n) = \text{bool-fix}\,(\mathbf{f}))\\
\to\quad (\text{g-provesn}\,(\text{g-find-proof}\,(pfn,\ exp,\ given,\ defns,\ symbols,\ n),\\
\quad\quad\quad\quad \text{g-f-notn}\,(exp),\\
\quad\quad\quad\quad given,\\
\quad\quad\quad\quad defns,\\
\quad\quad\quad\quad symbols,\\
\quad\quad\quad\quad \text{g-findn}\,(pfn,\ exp,\ given,\ defns,\ symbols,\ n))\\
\quad = \quad \text{bool-fix}\,(\mathbf{t}))
\end{array}
$$

Theorem: lessp-thm-count1

$$
\begin{array}{l}
(\text{exp-count}\,(\text{inv-code}\,(\text{decipher}\,(pfn)))\\
\ +\quad \text{formula-cnt}\,(\text{subst}\,(exp,\ var,\ term,\ flg),\ symbols)\\
\ +\quad \text{prf-cnt}\,(\text{inv-code}\,(\text{decipher}\,(pfn)),\ given,\ defns,\ \mathtt{0},\ symbols))\\
< \quad (1 + \text{thm-count}\,(pfn,\ exp,\ var,\ term,\ flg,\ given,\ defns,\ symbols))
\end{array}
$$

Event: Enable find-proof-proves-f-case; name this event ‘g1436’.

Event: Enable subst-ident; name this event ‘g1437’.

Theorem: lessp-thm-count2

(dec-countn (*pfn*) + exp-count (decipher (*pfn*)))
< (1 + thm-count (*pfn*, *exp*, *var*, *term*, *flg*, *given*, *defns*, *symbols*))

THEOREM: find-proof-proves-f-case-gcode
(((1 + thm-count (find-proofn (*pfn*,
gcode (subst (*exp*, *var*, *term*, 0)),
gcode (*given*),
gcode (*defns*),
gcode (*symbols*),
n),
f-not (*exp*),
var,
term,
0,
given,
defns,
symbols))
< g-findn (gcode (*pfn*),
gcode (subst (*exp*, *var*, *term*, 0)),
gcode (*given*),
gcode (*defns*),
gcode (*symbols*),
n))
∧ (g-isthmn (gcode (*pfn*),
gcode (subst (*exp*, *var*, *term*, 0)),
gcode (*given*),
gcode (*defns*),
gcode (*symbols*),
n)
= bool-fix (**f**)))
→ proves (inv-code (decipher (find-proofn (*pfn*,
gcode (subst (*exp*, *var*, *term*, 0)),
gcode (*given*),
gcode (*defns*),
gcode (*symbols*),
n))),
f-not (subst (*exp*, *var*, *term*, 0)),
given,
defns,
symbols)

EVENT: Enable thm-count; name this event 'g1438'.

```
;;;;;;;Now we start constructing the undecidable sentence.
```

DEFINITION:
a-undec1 (flg, exp, fa, val)
= forsome (1,
f-and (isin (z-list (list (ncode (val),
ncode (flg),
ncode (exp),
z-opair (6, ncode (gcode (**nil**))),
ncode (fa))),
1),
a-ev-defn (1)))

EVENT: Disable a-ev1; name this event ‘g1439’.

EVENT: Disable a-ev-defn; name this event ‘g1440’.

EVENT: Disable a-ev; name this event ‘g1441’.

THEOREM: replace-va-a-undec1
subst (a-undec1 (flg, exp, fa, val), 6, ncode ($va1$), 0)
= a-ev1 (ncode (flg),
ncode (exp),
z-opair (ncode ($va1$), ncode (gcode (**nil**))),
ncode (fa),
ncode (val))

EVENT: Disable a-ev-defn-part1; name this event ‘g1442’.

THEOREM: collect-free-a-undec1
($var \notin$ collect-free ($term$, 0))
$\rightarrow$ ($var \notin$ collect-free (subst (a-undec1 (flg, exp, fa, val), 6, $term$, 0), 0))

EVENT: Enable a-ev-defn-part1; name this event ‘g1443’.

EVENT: Enable a-ev-defn; name this event ‘g1444’.

THEOREM: formula-a-undec1
all-symbs ($symbols$) $\rightarrow$ formula (a-undec1 (flg, exp, fa, val), $symbols$)

EVENT: Enable a-ev1; name this event ‘g1445’.

EVENT: Enable a-ev; name this event ‘g1446’.

EVENT: Enable bool-fix; name this event 'g1447'.

EVENT: Enable g-isthm; name this event 'g1448'.

EVENT: Enable g-isthmn; name this event 'g1449'.

THEOREM: numeral-gcode
numeral (gcode (x)) = numeral (x)

```
;;;;;;;Correspondence for GC-NCODE.
```

DEFINITION:
gc-ncoden (x, n)
= **if** btmp (x) $\vee$ ($n \simeq 0$) **then** BTM
else pr-ifn (pr-listpn (x),
g-z-opairn (gc-ncoden (pr-carn (x), $n - 1$),
gc-ncoden (pr-cdrn (x), $n - 1$)),
g-numeraln (x, $n - 1$)) **endif**

EVENT: Disable gc-ncode; name this event 'g1450'.

EVENT: Disable gc-ncode-defn; name this event 'g1451'.

DEFINITION:
gc-ncode-ind (x, va, n)
= **if** $n \simeq 0$ **then** **t**
else gc-ncode-ind (pr-car (0), ev (1, list (x), va, FA, n), $n - 1$)
$\wedge$ gc-ncode-ind (pr-cdr (0), ev (1, list (x), va, FA, n), $n - 1$) **endif**

THEOREM: gc-ncode-eval
ev (0, gc-ncode (x), va, FA, n) = gc-ncoden (ev (0, x, va, FA, n), n)

EVENT: Disable z-opair; name this event 'g1452'.

EVENT: Disable z-sing; name this event 'g1453'.

EVENT: Disable z-pair; name this event 'g1454'.

THEOREM: z-pairn-gcode
g-z-pairn (gcode (x), gcode (y)) = gcode (z-pair (x, y))

EVENT: Enable z-pair; name this event 'g1455'.

EVENT: Enable z-pair; name this event 'g1456'.

EVENT: Enable g-z-pairn; name this event 'g1457'.

THEOREM: z-opairn-gcode
g-z-opairn (gcode (x), gcode (y)) = gcode (z-opair (x, y))

EVENT: Enable z-sing; name this event 'g1458'.

EVENT: Enable g-z-singn; name this event 'g1459'.

EVENT: Enable z-opair; name this event 'g1460'.

EVENT: Enable g-z-opairn; name this event 'g1461'.

EVENT: Disable g-numeraln; name this event 'g1462'.

EVENT: Disable ncode; name this event 'g1463'.

EVENT: Disable z-succ; name this event 'g1464'.

THEOREM: g-z-succ-gcode
g-z-succn (gcode (x)) = gcode (z-succ (x))

EVENT: Enable z-succ; name this event 'g1465'.

EVENT: Enable g-z-succn; name this event 'g1466'.

EVENT: Disable pr-zeropn; name this event 'g1467'.

EVENT: Disable pr-sub1n; name this event 'g1468'.

EVENT: Disable pr-add1n; name this event 'g1469'.

EVENT: Disable phi; name this event 'g1470'.

THEOREM: g-phin-gcode
$\text{G-PHIN} = \text{gcode}(\text{PHI})$

EVENT: Enable phi; name this event 'g1471'.

EVENT: Enable g-phin; name this event 'g1472'.

THEOREM: g-numeraln-ncode
$((x < n) \wedge (x \in \mathbf{N})) \rightarrow (\text{g-numeraln}(x,\ n) = \text{gcode}(\text{ncode}(x)))$

DEFINITION:
$\text{ncode-cnt}(x)$
$= \quad$ **if** $\text{listp}(x)$ **then** $1 + (\text{ncode-cnt}(\text{car}(x)) + \text{ncode-cnt}(\text{cdr}(x)))$
else $1 + \text{fix}(x)$ **endif**

DEFINITION:
$\text{ncode-ind}(x,\ n)$
$= \quad$ **if** $n \simeq 0$ **then t**
elseif $\text{listp}(x)$
then $\text{ncode-ind}(\text{car}(x),\ n - 1) \wedge \text{ncode-ind}(\text{cdr}(x),\ n - 1)$
else t endif

THEOREM: gc-ncoden-gcode
$((\text{ncode-cnt}(x) < n) \wedge \text{num-treep}(x))$
$\rightarrow \quad (\text{gc-ncoden}(x,\ n) = \text{gcode}(\text{ncode}(x)))$

EVENT: Enable gc-ncode; name this event 'g1473'.

EVENT: Enable gc-ncoden; name this event 'g1474'.

EVENT: Enable ncode; name this event 'g1475'.

THEOREM: gc-ncoden-gcode-again
$(\text{ncode-cnt}(\text{gcode}(x)) < n)$
$\rightarrow \quad (\text{gc-ncoden}(\text{gcode}(x),\ n) = \text{gcode}(\text{ncode}(\text{gcode}(x))))$

EVENT: Enable gc-ncode-defn; name this event 'g1476'.

```
;;;;;;;G-UNDEC (when PFN is 0) is the EV analogue of a the metatheoretic statement,
;;;;;;;"There exists a proof of the result of substituting (NCODE (GCODE EXP)) for the
;;;;;;;variable VAR in EXP (from GIVEN, DEFNS, and SYMBOLS)".
```

DEFINITION:
g-undec (*pfn*, *exp*, *var*, *given*, *defns*, *symbols*)
= g-isthm (*pfn*,
g-subst (*exp*, *var*, gc-ncode (*exp*), pr-quote (gcode (0))),
given,
defns,
symbols)

DEFINITION:
g-undecn (*pfn*, *exp*, *var*, *given*, *defns*, *symbols*, *n*)
= g-isthmn (*pfn*,
g-substn (*exp*, *var*, gc-ncoden (*exp*, *n*), gcode (0), *n*),
given,
defns,
symbols,
n)

THEOREM: g-undec-eval
ev (0, g-undec (*pfn*, *exp*, *var*, *given*, *defns*, *symbols*), *va*, FA, *n*)
= g-undecn (ev (0, *pfn*, *va*, FA, *n*),
ev (0, *exp*, *va*, FA, *n*),
ev (0, *var*, *va*, FA, *n*),
ev (0, *given*, *va*, FA, *n*),
ev (0, *defns*, *va*, FA, *n*),
ev (0, *symbols*, *va*, FA, *n*),
n)

EVENT: Enable g-undec; name this event 'g1477'.

```
;;;;;;;A-UNDEC is the formula from which the undecidable sentence is constructed.
;;;;;;;It is got from A-EV and asserts that the value of G-UNDEC on a particular
;;;;;;;EXP (represented by a free variable, say "y", in A-UNDEC) with GIVEN, DEFNS,
;;;;;;;and SYMBOLS, is (BOOL-FIX F).  This means that it is asserting, "there is
;;;;;;;no proof of the result of substituting (NCODE (GCODE y)) for the variable "y"
;;;;;;;in y, from GIVEN, DEFNS, and SYMBOLS."   The undecidable sentence will be got
;;;;;;;by substituting the formula the NCODE of the GCODE of A-UNDEC for the free "y"
;;;;;;;in A-UNDEC itself.
```

DEFINITION:
a-undec (*given*, *defns*, *symbols*)
= a-undec1 (0,
g-undec (pr-quote (gcode (0)),
0,
pr-quote (gcode (6)),

pr-quote (gcode (*given*)),
pr-quote (gcode (*defns*)),
pr-quote (gcode (*symbols*))),
FA,
bool-fix (**f**))

DEFINITION:
undec-count (*exp*, *var*, *term*, *given*, *defns*, *symbols*, *n*)
= ((1 + thm-count (find-proofn (0,
gcode (subst (*exp*, *var*, *term*, 0)),
gcode (*given*),
gcode (*defns*),
gcode (*symbols*),
n),
f-not (*exp*),
var,
term,
0,
given,
defns,
symbols))
+ (1 + thm-count (find-proofn (0,
gcode (subst (*exp*, *var*, *term*, 0)),
gcode (*given*),
gcode (*defns*),
gcode (*symbols*),
n),
exp,
var,
term,
0,
given,
defns,
symbols))
+ ncode-cnt (gcode (*exp*)))

DEFINITION:
isthm-a-proves-a-proof(*pfn*, *exp*, *var*, *term*, *given*, *defns*, *symbols*, *n*)
= inv-code (decipher (find-proofn (*pfn*,
gcode (subst (*exp*, *var*, *term*, 0)),
gcode (*given*),
gcode (*defns*),
gcode (*symbols*),
n)))

DEFINITION:
findn-count (*pfn*, *exp*, *var*, *term*, *given*, *defns*, *symbols*, *n*)
= g-findn (gcode (*pfn*),
gcode (subst (*exp*, *var*, *term*, `0`)),
gcode (*given*),
gcode (*defns*),
gcode (*symbols*),
n)

EVENT: Enable gcode; name this event 'g1478'.

EVENT: Enable exp-count; name this event 'g1479'.

```
;;;;;;;The lemma below begins the first of four important lemmas.
;;;;;;;The undecidable sentence is of the form;
;;;;;;;(SUBST (A-UNDEC GIVEN DEFNS SYMBOLS)
;;;;;;;       6
;;;;;;;       (NCODE (GCODE (A-UNDEC GIVEN DEFNS SYMBOLS))) 0),
;;;;;;;where DEFNS and SYMBOLS will be instantiated by a specific list of
;;;;;;;definitions and symbols, respectively.  The lemma asserts that if
;;;;;;;G-ISTHMN returns (BOOL-FIX T) on the GCODE of the undecidable sentence,
;;;;;;;then the undecidable sentence is provable.  This makes sense because
;;;;;;;G-ISTHMN returns (BOOL-FIX T) when it has found a proof in the enumeration
;;;;;;;of proofs.
```

THEOREM: isthm-a-proves-a
((undec-count (a-undec (*given*, *defns*, *symbols*),
`6`,
ncode (gcode (a-undec (*given*, *defns*, *symbols*))),
given,
defns,
symbols,
n)
$<$ findn-count (`0`,
a-undec (*given*, *defns*, *symbols*),
`6`,
ncode (gcode (a-undec (*given*, *defns*, *symbols*))),
given,
defns,
symbols,
n))
$\land$ (exp-count (a-undec (*given*, *defns*, *symbols*)) $<$ *n*)
$\land$ (g-isthmn (gcode (`0`),

```
                    g-substn (gcode (a-undec (given, defns, symbols)),
                              gcode (6),
                              gcode (ncode (gcode (a-undec (given, defns, symbols)))),
                              gcode (0),
                              n),
                    gcode (given),
                    gcode (defns),
                    gcode (symbols),
                    n)
          =    bool-fix (t)))
 →    proves (isthm-a-proves-a-proof(0,
                                     a-undec (given, defns, symbols),
                                     6,
                                     ncode (gcode (a-undec (given, defns, symbols))),
                                     given,
                                     defns,
                                     symbols,
                                     n),
                 subst (a-undec (given, defns, symbols),
                        6,
                        ncode (gcode (a-undec (given, defns, symbols))),
                        0),
                 given,
                 defns,
                 symbols)
```

```
;;;;;;;Similarly, if G-ISTHMN returns (BOOL-FIX F), it means that it found a disproof of
;;;;;;;the undecidable sentence, and its negation is therefore provable.
```

THEOREM: not-isthm-a-proves-not-a

```
 ((undec-count (a-undec (given, defns, symbols),
                6,
                ncode (gcode (a-undec (given, defns, symbols))),
                given,
                defns,
                symbols,
                n)
   <    findn-count (0,
                     a-undec (given, defns, symbols),
                     6,
                     ncode (gcode (a-undec (given, defns, symbols))),
                     given,
                     defns,
                     symbols,
```

n))
∧ (exp-count (a-undec (*given*, *defns*, *symbols*)) $<$ *n*)
∧ (g-isthmn (gcode (0),
g-substn (gcode (a-undec (*given*, *defns*, *symbols*)),
gcode (6),
gcode (ncode (gcode (a-undec (*given*, *defns*, *symbols*)))),
gcode (0),
n),
gcode (*given*),
gcode (*defns*),
gcode (*symbols*),
n)
= bool-fix (**f**)))
→ proves (isthm-a-proves-a-proof(0,
a-undec (*given*, *defns*, *symbols*),
6,
ncode (gcode (a-undec (*given*, *defns*, *symbols*))),
given,
defns,
symbols,
n),
f-not (subst (a-undec (*given*, *defns*, *symbols*),
6,
ncode (gcode (a-undec (*given*, *defns*, *symbols*))),
0)),
given,
defns,
symbols)

EVENT: Enable isthm-a-proves-a-proof; name this event ‘g1480’.

EVENT: Enable a-ev1-not-good-proof; name this event ‘g1481’.

DEFINITION:
isthm-a-proves-not-a-proof(*given*, *defns*, *symbols*, *n*)
= a-ev1-not-good-proof(0,
g-undec (pr-quote (gcode (0)),
0,
pr-quote (gcode (6)),
pr-quote (gcode (*given*)),
pr-quote (gcode (*defns*)),
pr-quote (gcode (*symbols*))),
cons (gcode (a-undec (*given*, *defns*, *symbols*)),

gcode (**nil**)),
FA,
n,
bool-fix (**f**))

```
;;;;;;;The more surprising pair of facts are that if G-ISTHMN returns (BOOL-FIX T)
;;;;;;;the GCODE of the undecidable sentence, then the negation of the undecidable
;;;;;;;sentence is provable.  This is because the undecidable sentence through the
;;;;;;;fact that it represents some instance of the Lisp interpreter asserts that G-ISTHMN
;;;;;;;returns (BOOL-FIX F) on the undecidable sentence itself, which in this case is false.
```

THEOREM: isthm-a-proves-not-a-proof-proves
(ev-hyps (*given*, *defns*, *symbols*)
∧ (ncode-cnt (gcode (a-undec (*given*, *defns*, *symbols*))) $<$ n)
∧ (exp-count (a-undec (*given*, *defns*, *symbols*)) $<$ n)
∧ (g-isthmn (gcode (0),
gcode (subst (a-undec (*given*, *defns*, *symbols*),
6,
ncode (gcode (a-undec (*given*, *defns*, *symbols*))),
0)),
gcode (*given*),
gcode (*defns*),
gcode (*symbols*),
n)
= bool-fix (**t**))
∧ (*concl* = f-not (subst (a-undec (*given*, *defns*, *symbols*),
6,
ncode (gcode (a-undec (*given*, *defns*, *symbols*))),
0))))
→ proves (isthm-a-proves-not-a-proof (*given*, *defns*, *symbols*, n),
concl,
given,
defns,
symbols)

EVENT: Enable isthm-a-proves-not-a-proof; name this event 'g1482'.

DEFINITION:
not-isthm-a-proves-a-proof (*given*, *defns*, *symbols*, n)
= a-ev1-ok-proof (0,
g-undec (pr-quote (gcode (0)),
0,
pr-quote (gcode (6)),
pr-quote (gcode (*given*)),

```
                                pr-quote (gcode (defns)),
                                pr-quote (gcode (symbols))),
                      cons (gcode (a-undec (given, defns, symbols)), gcode (nil)),
                      FA,
                      n)
```

```
;;;;;;;If G-ISTHMN returns (BOOL-FIX F) on the the GCODE of the undecidable sentence,
;;;;;;;then the undecidable sentence is provable.  This is because the undecidable
;;;;;;;sentence through the fact that it represents some instance of the Lisp interpreter
;;;;;;;asserts that G-ISTHMN does return (BOOL-FIX F) on the undecidable sentence itself.
```

THEOREM: not-isthm-a-proves-a-proof-proves

```
 (ev-hyps (given, defns, symbols)
  ∧   (ncode-cnt (gcode (a-undec (given, defns, symbols))) < n)
  ∧   (exp-count (a-undec (given, defns, symbols)) < n)
  ∧   (g-isthmn (gcode (0),
                 gcode (subst (a-undec (given, defns, symbols),
                               6,
                               ncode (gcode (a-undec (given, defns, symbols))),
                               0)),
                 gcode (given),
                 gcode (defns),
                 gcode (symbols),
                 n)
        =    bool-fix (f))
  ∧   (concl = subst (a-undec (given, defns, symbols),
                      6,
                      ncode (gcode (a-undec (given, defns, symbols))),
                      0)))
 →   proves (not-isthm-a-proves-a-proof(given, defns, symbols, n),
             concl,
             given,
             defns,
             symbols)
```

```
;;;;;;;Now we need to prove that if Z2 is complete, then G-ISTHMN returns either (BOOL-FIX T)
;;;;;;;or (BOOL-FIX F).  Then by the above four lemmas, we have that the undecidable sentence
;;;;;;;is both provable and disprovable.  Therefore, we would have established that Z2 is
;;;;;;;either incomplete or inconsistent.
```

EVENT: Enable isthm-a-proves-not-a-proof; name this event ‘g1483’.

DEFINITION:
thm-count1 (*pfn*, *exp*, *given*, *defns*, *symbols*)

$=\quad$ ((dec-countn (*pfn*) + exp-count (decipher (*pfn*)))
  + exp-count (inv-code (decipher (*pfn*)))
  + formula-cnt (*exp*, *symbols*)
  + prf-cnt (inv-code (decipher (*pfn*)), *given*, *defns*, `0`, *symbols*))

THEOREM: thm-count1-lessp
(dec-count (*pfn*, *pfn* $-$ 1, *pfn* $-$ 1) + exp-count (decipher (*pfn*)))
$<$ (1 + thm-count1 (*pfn*, *exp*, *given*, *defns*, *symbols*))

THEOREM: thm-count1-lessp1
(exp-count (inv-code (decipher (*pfn*)))
+ formula-cnt (*exp*, *symbols*)
+ prf-cnt (inv-code (decipher (*pfn*)), *given*, *defns*, `0`, *symbols*))
$<$ (1 + thm-count1 (*pfn*, *exp*, *given*, *defns*, *symbols*))

EVENT: Enable thm-count1; name this event 'g1484'.

DEFINITION:
comp-count (*pf*, *pfn*, *exp*, *given*, *defns*, *symbols*, *extras*)
$=$ **if** *pf* $\leq$ *pfn*
  then (1 + thm-count1 (*pfn*, *exp*, *given*, *defns*, *symbols*))
    + (1 + thm-count1 (*pfn*, f-not (*exp*), *given*, *defns*, *symbols*))
    + (1 + *extras*)
  elseif proves (inv-code (decipher (*pfn*)), *exp*, *given*, *defns*, *symbols*)
  then (1 + thm-count1 (*pfn*, *exp*, *given*, *defns*, *symbols*))
    + (1 + thm-count1 (*pfn*, f-not (*exp*), *given*, *defns*, *symbols*))
    + (1 + *extras*)
  elseif proves (inv-code (decipher (*pfn*)), f-not (*exp*), *given*, *defns*, *symbols*)
  then (1 + thm-count1 (*pfn*, *exp*, *given*, *defns*, *symbols*))
    + (1 + thm-count1 (*pfn*, f-not (*exp*), *given*, *defns*, *symbols*))
    + (1 + *extras*)
  else (1 + thm-count1 (*pfn*, *exp*, *given*, *defns*, *symbols*))
    + (1 + thm-count1 (*pfn*, f-not (*exp*), *given*, *defns*, *symbols*))
    + (1 + *extras*)
    + (1 + comp-count (*pf*,
        1 + *pfn*,
        exp,
        given,
        defns,
        symbols,
        extras)) **endif**

DEFINITION:
proves-isthmn-ind (*pf*, *pfn*, *exp*, *given*, *defns*, *symbols*, *n*)

$=$ **if** $n \simeq 0$ **then t**
elseif $pf \leq pfn$ **then t**
elseif proves (inv-code (decipher (*pfn*)), *exp*, *given*, *defns*, *symbols*) **then t**
elseif proves (inv-code (decipher (*pfn*)), f-not (*exp*), *given*, *defns*, *symbols*)
then t
else proves-isthmn-ind (*pf*, $1 + pfn$, *exp*, *given*, *defns*, *symbols*, $n - 1$) **endif**

EVENT: Disable g-isthmn; name this event 'g1485'.

THEOREM: lessp-comp-count1
thm-count1 (*pfn*, *exp*, *given*, *defns*, *symbols*)
$<$ comp-count (*pf*, *pfn*, *exp*, *given*, *defns*, *symbols*, *extras*)

THEOREM: lessp-comp-count2
thm-count1 (*pfn*, f-not (*exp*), *given*, *defns*, *symbols*)
$<$ comp-count (*pf*, *pfn*, *exp*, *given*, *defns*, *symbols*, *extras*)

THEOREM: lessp-extras-comp-count
extras $<$ comp-count (*pf*, *pfn*, *exp*, *given*, *defns*, *symbols*, *extras*)

EVENT: Enable find-proof-proves-f-case-gcode; name this event 'g1486'.

EVENT: Enable find-proof-proves-t-case-gcode; name this event 'g1487'.

EVENT: Enable g-provesn-t-or-f1; name this event 'g1488'.

EVENT: Enable g-provesn-t-or-f2; name this event 'g1489'.

EVENT: Enable pr-ifn; name this event 'g1490'.

```
;;;;;;;The lemma below is the crucial one in the final proof of the incompleteness theorem.
;;;;;;;It asserts that if there is a number PF which corresponds to a proof or disproof of EXP,
;;;;;;;then G-ISTHMN (with a large enough stack-counter) when started on a proof-number PFN
;;;;;;;less than or equal to PF, will either find PF or some proof/disproof before it, and
;;;;;;;will therefore either return (BOOL-FIX T) or (BOOL-FIX F).  This with the four lemmas
;;;;;;;above provides enough material to prove the incompleteness theorem, but we still need
;;;;;;;to build up a large enough stack-counter to cover all these five lemmas.
```

THEOREM: proves-t-not-btmp-isthmn
(($pfn \in \mathbf{N}$)
$\wedge$ ($pf \in \mathbf{N}$)
$\wedge$ (proves (inv-code (decipher (*pf*)), *exp*, *given*, *defns*, *symbols*)

$\lor$ proves (inv-code (decipher (pf)), f-not (exp), $given$, $defns$, $symbols$))
$\land$ ($pfn \leq pf$)
$\land$ (comp-count (pf, pfn, exp, $given$, $defns$, $symbols$, $extras$) $<$ n))
$\rightarrow$ ((g-isthmn (gcode (pfn),
gcode (exp),
gcode ($given$),
gcode ($defns$),
gcode ($symbols$),
n)
$=$ bool-fix (**t**))
$\lor$ (g-isthmn (gcode (pfn),
gcode (exp),
gcode ($given$),
gcode ($defns$),
gcode ($symbols$),
n)
$=$ bool-fix (**f**)))

EVENT: Enable proves-t-not-btmp-isthmn; name this event 'g1491'.

EVENT: Disable find-proofn; name this event 'g1492'.

DEFINITION:
findn (pfn, exp, $given$, $defns$, $symbols$, n)
$=$ **if** btmp (exp)
$\lor$ btmp ($given$)
$\lor$ btmp ($defns$)
$\lor$ btmp ($symbols$)
$\lor$ ($n \simeq 0$) **then** BTM
else pr-ifn (g-provesn (g-inv-coden (g-deciphern (gcode (pfn), $n - 1$),
$n - 1$),
exp,
$given$,
$defns$,
$symbols$,
$n - 1$),
$n - 1$,
pr-ifn (g-provesn (g-inv-coden (g-deciphern (gcode (pfn),
$n - 1$),
$n - 1$),
g-f-notn (exp),
$given$,
$defns$,

$symbols$,
$n - 1$),
$n - 1$,
findn (1 + pfn, exp, $given$, $defns$, $symbols$, $n - 1$))) **endif**

EVENT: Enable pr-ifn; name this event 'g1493'.

THEOREM: lessp-comp-count3
(thm-count1 (pfn, f-not (exp), $given$, $defns$, $symbols$)
+ thm-count1 (pfn, exp, $given$, $defns$, $symbols$)
+ $extras$)
< comp-count (pf, pfn, exp, $given$, $defns$, $symbols$, $extras$)

THEOREM: findn-thm-count-lessp
(($pfn \in \mathbf{N}$)
∧ ($pf \in \mathbf{N}$)
∧ (proves (inv-code (decipher (pf)), exp, $given$, $defns$, $symbols$)
∨ proves (inv-code (decipher (pf)), f-not (exp), $given$, $defns$, $symbols$))
∧ ($pfn \leq pf$)
∧ ((1 + (1 + comp-count (pf, pfn, exp, $given$, $defns$, $symbols$, $extras$))) $< n$))
→ (((1 + thm-count1 (find-proofn (pfn,
gcode (exp),
gcode ($given$),
gcode ($defns$),
gcode ($symbols$),
n),
f-not (exp),
$given$,
$defns$,
$symbols$))
+ (1 + thm-count1 (find-proofn (pfn,
gcode (exp),
gcode ($given$),
gcode ($defns$),
gcode ($symbols$),
n),
exp,
$given$,
$defns$,
$symbols$))
+ $extras$)
< findn (pfn,
gcode (exp),
gcode ($given$),

gcode (*defns*),
gcode (*symbols*),
n))

Event: Disable g-findn; name this event 'g1494'.

Theorem: findn-g-findn
g-findn (gcode (*pfn*), *exp*, *given*, *defns*, *symbols*, *n*)
= findn (*pfn*, *exp*, *given*, *defns*, *symbols*, *n*)

Event: Enable g-findn; name this event 'g1495'.

Event: Disable thm-count; name this event 'g1496'.

Event: Disable thm-count1; name this event 'g1497'.

Theorem: thm-count-thm-count1
thm-count (*pfn*, *exp*, *var*, *term*, *flg*, *given*, *defns*, *symbols*)
= thm-count1 (*pfn*, subst (*exp*, *var*, *term*, *flg*), *given*, *defns*, *symbols*)

Event: Enable thm-count; name this event 'g1498'.

Event: Enable thm-count1; name this event 'g1499'.

Event: Disable decipher; name this event 'g1500'.

Theorem: num-treep-num-treep
numtreep (x) = num-treep (x)

Theorem: inv-code-decipher-numcode
gcode (inv-code (decipher (numcode (gcode (x))))) = gcode (x)

Event: Enable decipher; name this event 'g1501'.

Event: Disable bool-fix; name this event 'g1502'.

Theorem: proves-inv-code-decipher
proves (inv-code (decipher (numcode (gcode (*pf*)))), *exp*, *given*, *defns*, *symbols*)
= proves (*pf*, *exp*, *given*, *defns*, *symbols*)

EVENT: Enable findn; name this event 'g1503'.

EVENT: Enable bool-fix; name this event 'g1504'.

```
;;;;;;;The undecidable sentence - DEFNS and SYMBOLS will have to be instantiated, but
;;;;;;;GIVEN will be left as a parameter so that for any specific GIVEN, we get a specific
;;;;;;;undecidable sentence.  This would show that the addition of the undecidable sentence
;;;;;;;or any finite list of new axioms will still leave the theory incomplete or inconsistent.
```

DEFINITION:
undec-sent (*given*, *defns*, *symbols*)
= subst (a-undec (*given*, *defns*, *symbols*),
6,
ncode (gcode (a-undec (*given*, *defns*, *symbols*))),
0)

DEFINITION:
final-count (*pf*, *given*, *defns*, *symbols*)
= (1 + ((1 + (1 + comp-count (numcode (gcode (*pf*)),
0,
undec-sent (*given*, *defns*, *symbols*),
given,
defns,
symbols,
ncode-cnt (gcode (a-undec (*given*,
defns,
symbols))))))
+ (1 + ncode-cnt (gcode (a-undec (*given*, *defns*, *symbols*))))
+ (1 + exp-count (a-undec (*given*, *defns*, *symbols*)))))

EVENT: Enable ncode-cnt; name this event 'g1505'.

EVENT: Enable proves-is-formula-again; name this event 'g1506'.

EVENT: Enable comp-count; name this event 'g1507'.

EVENT: Enable numcode; name this event 'g1508'.

EVENT: Enable exp-count; name this event 'g1509'.

EVENT: Enable findn-thm-count-lessp; name this event 'g1510'.

THEOREM: undec-count-lessp
$$
\begin{array}{l}
((pf \in \mathbf{N}) \\
\ \wedge\ \ (\text{proves}\,(\text{inv-code}\,(\text{decipher}\,(pf)), \\
\qquad\qquad\quad \text{subst}\,(exp,\ var,\ term,\ \mathtt{0}), \\
\qquad\qquad\quad given, \\
\qquad\qquad\quad defns, \\
\qquad\qquad\quad symbols) \\
\qquad \vee\ \ \text{proves}\,(\text{inv-code}\,(\text{decipher}\,(pf)), \\
\qquad\qquad\qquad \text{f-not}\,(\text{subst}\,(exp,\ var,\ term,\ \mathtt{0})), \\
\qquad\qquad\qquad given, \\
\qquad\qquad\qquad defns, \\
\qquad\qquad\qquad symbols)) \\
\ \wedge\ \ ((1 + (1 + \text{comp-count}\,(pf, \\
\qquad\qquad\qquad\qquad\qquad \mathtt{0}, \\
\qquad\qquad\qquad\qquad\qquad \text{subst}\,(exp,\ var,\ term,\ \mathtt{0}), \\
\qquad\qquad\qquad\qquad\qquad given, \\
\qquad\qquad\qquad\qquad\qquad defns, \\
\qquad\qquad\qquad\qquad\qquad symbols, \\
\qquad\qquad\qquad\qquad\qquad \text{ncode-cnt}\,(\text{gcode}\,(exp))))) \\
\qquad <\ \ n)) \\
\to\ \ (\text{undec-count}\,(exp,\ var,\ term,\ given,\ defns,\ symbols,\ n) \\
\qquad <\ \ \text{findn}\,(\mathtt{0}, \\
\qquad\qquad\qquad \text{gcode}\,(\text{subst}\,(exp,\ var,\ term,\ \mathtt{0})), \\
\qquad\qquad\qquad \text{gcode}\,(given), \\
\qquad\qquad\qquad \text{gcode}\,(defns), \\
\qquad\qquad\qquad \text{gcode}\,(symbols), \\
\qquad\qquad\qquad n))
\end{array}
$$

EVENT: Enable undec-count; name this event ‘g1511’.

EVENT: Enable not-isthm-a-proves-a-proof; name this event ‘g1512’.

THEOREM: comp-count-final-count-lessp
$$
\begin{array}{l}
(1 + (1 + \text{comp-count}\,(\text{numcode}\,(\text{gcode}\,(pf)), \\
\qquad\qquad\qquad\qquad \mathtt{0}, \\
\qquad\qquad\qquad\qquad \text{subst}\,(\text{a-undec}\,(given,\ defns,\ symbols), \\
\qquad\qquad\qquad\qquad\qquad \mathtt{6}, \\
\qquad\qquad\qquad\qquad\qquad \text{ncode}\,(\text{gcode}\,(\text{a-undec}\,(given,\ defns,\ symbols))), \\
\qquad\qquad\qquad\qquad\qquad \mathtt{0}), \\
\qquad\qquad\qquad\qquad given, \\
\qquad\qquad\qquad\qquad defns, \\
\qquad\qquad\qquad\qquad symbols, \\
\qquad\qquad\qquad\qquad \text{ncode-cnt}\,(\text{gcode}\,(\text{a-undec}\,(given,\ defns,\ symbols))))))
\end{array}
$$

$<$ final-count (pf, $given$, $defns$, $symbols$)

THEOREM: lessp-ncode-cnt-final-count
ncode-cnt (gcode (a-undec ($given$, $defns$, $symbols$)))
$<$ final-count (pf, $given$, $defns$, $symbols$)

THEOREM: lessp-exp-count-final-count
exp-count (a-undec ($given$, $defns$, $symbols$))
$<$ final-count (pf, $given$, $defns$, $symbols$)

THEOREM: lessp-num-final-count
4 $<$ final-count (pf, $given$, $defns$, $symbols$)

```
;;;;;;;The final list of symbols, THM-SYMBS.
```

DEFINITION:
THM-SYMBS
= list (fn (16, 1),
fn (15, 2),
fn (14, 2),
fn (13, 2),
fn (12, 2),
fn (11, 2),
fn (10, 1),
fn (9, 1),
fn (8, 1),
fn (7, 1),
fn (6, 1),
fn (5, 1),
fn (4, 1),
fn (3, 1),
p (2, 1),
fn (2, 2),
fn (1, 2),
p (1, 2),
fn (0, 0))

```
;;;;;;;The final list of function/predicate definitions.
```

DEFINITION:
THM-DEFNS
= list (cons (p (2, 1), z-int-defn (0, 1, 2)),
cons (fn (3, 1), f-equal (z-succ (0), z-union (z-sing (0), 0))),
cons (fn (4, 1), z-car-defn (0, z-car (0), 1, 2)),
cons (fn (5, 1), z-cdr-defn (0, z-cdr (0), 1, 2)),

cons (fn (6, 1), z-listp-defn (0, z-listp (0), 1, 2)),
cons (fn (7, 1), z-add1-defn (0, z-add1 (0), 1, 2)),
cons (fn (8, 1), z-numberp-defn (0, z-numberp (0))),
cons (fn (9, 1), z-zerop-defn (0, z-zerop (0))),
cons (fn (10, 1), z-sub1-defn (0, z-sub1 (0), 3)),
cons (fn (11, 2), z-equal-defn (0, 1, z-equal (0, 1))),
cons (fn (12, 2), z-cons-defn (0, 1, z-cons (0, 1))),
cons (fn (13, 2), z-apply-subr1-defn (0, 1, z-apply-subr1 (0, 1))),
cons (fn (14, 2), z-apply-subr2-defn (0, 1, z-apply-subr2 (0, 1))),
cons (fn (15, 2), z-apply-subr-defn (0, 1, z-apply-subr (0, 1))),
cons (fn (16, 1), z-subrp-defn (0, z-subrp (0))))

EVENT: Disable z-subrp-hyps; name this event ‘g1549’.

EVENT: Disable z-apply-subr-hyps; name this event ‘g1550’.

EVENT: Disable z-apply-subr1-hyps; name this event ‘g1551’.

EVENT: Disable z-apply-subr2-hyps; name this event ‘g1552’.

EVENT: Disable z-succ-hyps; name this event ‘g1553’.

EVENT: Disable z-int-hyps; name this event ‘g1554’.

EVENT: Disable z-car-hyps; name this event ‘g1555’.

EVENT: Disable z-cdr-hyps; name this event ‘g1556’.

EVENT: Disable z-listp-hyps; name this event ‘g1557’.

EVENT: Disable z-cons-hyps; name this event ‘g1558’.

EVENT: Disable z-add1-hyps; name this event ‘g1559’.

EVENT: Disable z-sub1-hyps; name this event ‘g1560’.

EVENT: Disable z-equal-hyps; name this event ‘g1561’.

EVENT: Disable z-numberp-hyps; name this event ‘g1562’.

EVENT: Disable rest-of; name this event ‘g1563’.

EVENT: Enable eql-gcode-equal; name this event ‘g1564’.

THEOREM: ev-hyps-z-int
z-int-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: ev-hyps-z-succ
z-succ-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: ev-hyps-z-car
z-car-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: ev-hyps-z-cdr
z-cdr-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: z-listp-ev-hyps
z-listp-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: z-cons-ev-hyps
z-cons-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: z-numberp-ev-hyps
z-numberp-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: z-equal-ev-hyps
z-equal-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: z-add1-ev-hyps
z-add1-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: z-sub1-ev-hyps
z-sub1-hyps (THM-DEFNS, THM-SYMBS)

EVENT: Enable subset-cons; name this event ‘g1565’.

EVENT: Enable prime-member; name this event ‘g1566’.

EVENT: Enable member-delete; name this event ‘g1567’.

THEOREM: z-apply-subr1-ev-hyps
z-apply-subr1-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: z-apply-subr2-ev-hyps
z-apply-subr2-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: z-apply-subr-ev-hyps
z-apply-subr-hyps (THM-DEFNS, THM-SYMBS)

THEOREM: z-subrp-ev-hyps
z-subrp-hyps (THM-DEFNS, THM-SYMBS)

EVENT: Enable z-subrp-hyps; name this event 'g1568'.

EVENT: Enable z-apply-subr-hyps; name this event 'g1569'.

EVENT: Enable z-apply-subr1-hyps; name this event 'g1570'.

EVENT: Enable z-apply-subr2-hyps; name this event 'g1571'.

EVENT: Enable z-succ-hyps; name this event 'g1572'.

EVENT: Enable z-int-hyps; name this event 'g1573'.

EVENT: Enable z-car-hyps; name this event 'g1574'.

EVENT: Enable z-cdr-hyps; name this event 'g1575'.

EVENT: Enable z-listp-hyps; name this event 'g1576'.

EVENT: Enable z-cons-hyps; name this event 'g1577'.

EVENT: Enable z-add1-hyps; name this event 'g1578'.

EVENT: Enable z-sub1-hyps; name this event 'g1579'.

EVENT: Enable z-equal-hyps; name this event 'g1580'.

EVENT: Enable z-numberp-hyps; name this event 'g1581'.

THEOREM: subset-thm-symbs

subset (list (fn (16, 1),
fn (15, 2),
fn (14, 2),
fn (13, 2),
fn (12, 2),
fn (0, 0),
fn (3, 1),
fn (4, 1),
fn (5, 1),
fn (6, 1),
fn (7, 1),
fn (8, 1),
fn (9, 1),
fn (10, 1),
fn (11, 2),
fn (1, 2),
fn (2, 2),
p (2, 1),
p (1, 2)),
THM-SYMBS)

EVENT: Disable z-zerop-hyps; name this event 'g1585'.

THEOREM: z-zerop-ev-hyps
z-zerop-hyps (THM-DEFNS, THM-SYMBS)

EVENT: Enable rest-of; name this event 'g1586'.

EVENT: Enable thm-defns; name this event 'g1587'.

THEOREM: all-symbs-thm-symbs
all-symbs (THM-SYMBS)

EVENT: Enable thm-symbs; name this event 'g1589'.

THEOREM: ev-hyps-thm-defns-symbols
ev-hyps (*given*, THM-DEFNS, THM-SYMBS)

THEOREM: all-symbs-termp-ncode
all-symbs (*symbs*) $\rightarrow$ termp (ncode (x), 0, *symbs*)

EVENT: Enable all-symbs; name this event 'g1591'.

THEOREM: formula-undec-sent
all-symbs (*symbs*) → formula (undec-sent (*given*, *defns*, *symbs*), *symbs*)

```
;;;;;;;The lemma below is very nearly the final one.  It asserts that if PF is either
;;;;;;;a proof or disproof of the undecidable sentence, then in one of two possible ways
;;;;;;;(as given by the four lemmas above), the undecidable sentence is both provable and
;;;;;;;disprovable.  The observation is that if Z2 was complete, the undecidable sentence
;;;;;;;would either be a provable or disprovable.  PF below could be instantiated with
;;;;;;;the given proof/disproof, yielding the conclusion of the lemma that Z2 is then
;;;;;;;inconsistent.  We still need to show that the undecidable sentence below is a
;;;;;;;Z2 formula and that it contains no free variables.  This is done next.
```

THEOREM: incompleteness-main-part
(proves (*pf*, undec-sent (*given*, THM-DEFNS, THM-SYMBS), *given*, THM-DEFNS, THM-SYMBS)
 ∨ proves (*pf*,
  f-not (undec-sent (*given*, THM-DEFNS, THM-SYMBS)),
  given,
  THM-DEFNS,
  THM-SYMBS))
→ ((proves (isthm-a-proves-a-proof(0,
  a-undec (*given*, THM-DEFNS, THM-SYMBS),
  6,
  ncode (gcode (a-undec (*given*,
  THM-DEFNS,
  THM-SYMBS))),
  given,
  THM-DEFNS,
  THM-SYMBS,
  final-count (*pf*,
  given,
  THM-DEFNS,
  THM-SYMBS)),
  undec-sent (*given*, THM-DEFNS, THM-SYMBS),
  given,
  THM-DEFNS,
  THM-SYMBS)
 ∧ proves (isthm-a-proves-not-a-proof(*given*,
  THM-DEFNS,
  THM-SYMBS,
  final-count (*pf*,
  given,
  THM-DEFNS,
  THM-SYMBS)),
  f-not (undec-sent (*given*, THM-DEFNS, THM-SYMBS)),

```
                    given,
                    THM-DEFNS,
                    THM-SYMBS))
     ∨   (proves (not-isthm-a-proves-a-proof (given,
                                               THM-DEFNS,
                                               THM-SYMBS,
                                               final-count (pf,
                                                            given,
                                                            THM-DEFNS,
                                                            THM-SYMBS)),
                    undec-sent (given, THM-DEFNS, THM-SYMBS),
                    given,
                    THM-DEFNS,
                    THM-SYMBS)
          ∧   proves (isthm-a-proves-a-proof (0,
                                               a-undec (given,
                                                        THM-DEFNS,
                                                        THM-SYMBS),
                                               6,
                                               ncode (gcode (a-undec (given,
                                                                      THM-DEFNS,
                                                                      THM-SYMBS))),
                                               given,
                                               THM-DEFNS,
                                               THM-SYMBS,
                                               final-count (pf,
                                                            given,
                                                            THM-DEFNS,
                                                            THM-SYMBS)),
                      f-not (undec-sent (given, THM-DEFNS, THM-SYMBS)),
                      given,
                      THM-DEFNS,
                      THM-SYMBS)))
```

THEOREM: collect-free-undec-sent
$var \notin$ collect-free (undec-sent (*given*, *defns*, *symbs*), 0)

EVENT: Enable undec-sent; name this event 'g1593'.

```
;;;;;;;The undecidable sentence: UNDEC-SENT with DEFNS and SYMBOLS instantiated.o
```

DEFINITION:
undecidable-sentence (*given*) = undec-sent (*given*, THM-DEFNS, THM-SYMBS)

Definition:
some-proof1 (*given*, *pf*)
= isthm-a-proves-a-proof(0,
a-undec (*given*, THM-DEFNS, THM-SYMBS),
6,
ncode (gcode (a-undec (*given*, THM-DEFNS, THM-SYMBS))),
given,
THM-DEFNS,
THM-SYMBS,
final-count (*pf*, *given*, THM-DEFNS, THM-SYMBS))

Definition:
some-disproof1 (*given*, *pf*)
= isthm-a-proves-not-a-proof(*given*,
THM-DEFNS,
THM-SYMBS,
final-count (*pf*, *given*, THM-DEFNS, THM-SYMBS))

Definition:
some-proof2 (*given*, *pf*)
= not-isthm-a-proves-a-proof(*given*,
THM-DEFNS,
THM-SYMBS,
final-count (*pf*, *given*, THM-DEFNS, THM-SYMBS))

Definition:
some-disproof2 (*given*, *pf*)
= isthm-a-proves-a-proof(0,
a-undec (*given*, THM-DEFNS, THM-SYMBS),
6,
ncode (gcode (a-undec (*given*, THM-DEFNS, THM-SYMBS))),
given,
THM-DEFNS,
THM-SYMBS,
final-count (*pf*, *given*, THM-DEFNS, THM-SYMBS))

```
;;;;;;;The final statement of the incompleteness theorem: [1] The undecidable sentence
;;;;;;;is a Z2 formula in a theory whose symbols are from (THM-SYMBS),
;;;;;;;[2] the undecidable sentence is a sentence, i.e., contains no variables VAR free,
;;;;;;;[3] if the undecidable sentence is either provable or disprovable, then it is
;;;;;;;both provable and disprovable.
```

Theorem: incompleteness-theorem
formula (undecidable-sentence (*given*), THM-SYMBS)
$\wedge$ (*var* $\notin$ collect-free (undecidable-sentence (*given*), 0))

$$
\begin{array}{l}
\wedge \quad ((\mathrm{proves}\,(pf,\ \mathrm{undecidable\text{-}sentence}\,(given),\ given,\ \textsc{thm-defns},\ \textsc{thm-symbs})\\
\qquad \vee \quad \mathrm{proves}\,(pf,\\
\qquad\qquad\qquad \mathrm{f\text{-}not}\,(\mathrm{undecidable\text{-}sentence}\,(given)),\\
\qquad\qquad\qquad given,\\
\qquad\qquad\qquad \textsc{thm-defns},\\
\qquad\qquad\qquad \textsc{thm-symbs}))\\
\qquad \rightarrow \quad ((\mathrm{proves}\,(\mathrm{some\text{-}proof1}\,(given,\ pf),\\
\qquad\qquad\qquad\quad \mathrm{undecidable\text{-}sentence}\,(given),\\
\qquad\qquad\qquad\quad given,\\
\qquad\qquad\qquad\quad \textsc{thm-defns},\\
\qquad\qquad\qquad\quad \textsc{thm-symbs})\\
\qquad\qquad \wedge \quad \mathrm{proves}\,(\mathrm{some\text{-}disproof1}\,(given,\ pf),\\
\qquad\qquad\qquad\qquad \mathrm{f\text{-}not}\,(\mathrm{undecidable\text{-}sentence}\,(given)),\\
\qquad\qquad\qquad\qquad given,\\
\qquad\qquad\qquad\qquad \textsc{thm-defns},\\
\qquad\qquad\qquad\qquad \textsc{thm-symbs}))\\
\qquad\qquad \vee \quad (\mathrm{proves}\,(\mathrm{some\text{-}proof2}\,(given,\ pf),\\
\qquad\qquad\qquad\qquad \mathrm{undecidable\text{-}sentence}\,(given),\\
\qquad\qquad\qquad\qquad given,\\
\qquad\qquad\qquad\qquad \textsc{thm-defns},\\
\qquad\qquad\qquad\qquad \textsc{thm-symbs})\\
\qquad\qquad\qquad \wedge \quad \mathrm{proves}\,(\mathrm{some\text{-}disproof2}\,(given,\ pf),\\
\qquad\qquad\qquad\qquad\quad \mathrm{f\text{-}not}\,(\mathrm{undecidable\text{-}sentence}\,(given)),\\
\qquad\qquad\qquad\qquad\quad given,\\
\qquad\qquad\qquad\qquad\quad \textsc{thm-defns},\\
\qquad\qquad\qquad\qquad\quad \textsc{thm-symbs}))))
\end{array}
$$

Index