

#|

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.

|#

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

```
;Annotated script for mechanical proof of the Tautology theorem.
;Proof involves -
;Definition of proof-checker for Schoenfield's FOL.
;Proof of several derived inference rules, primarily the
;subset lemma.
;Definition of tautology-checker.
;Every tautology has a proof.
;Correctness of tautology-checker - every tautology is
;always logically-true, and all logical-truths are tautologies.
;First, functions, variables and predicate symbols.
```

DEFINITION:

```
function (fn)
= ((fn = list ('f, cadr (fn), caddr (fn)))
   ^ (cadr (fn) ∈ ℕ)
   ^ (caddr (fn) ∈ ℕ))
```

DEFINITION:

$\text{variable}(x) = ((x = \text{list}('x, \text{cadr}(x))) \wedge (\text{cadr}(x) \in \mathbf{N}))$

DEFINITION:

$\text{predicate}(p)$
= $((p = \text{list}('p, \text{cadr}(p), \text{caddr}(p)))$
 $\wedge (\text{cadr}(p) \in \mathbf{N})$
 $\wedge (\text{caddr}(p) \in \mathbf{N}))$
 $\vee (p = \text{'equal}))$

DEFINITION:

$\text{degree}(fn)$
= **if** $fn = \text{'equal}$ **then** 2
 else $\text{caddr}(fn)$ **endif**

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

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

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

THEOREM: numberp-fix

$\text{fix}(x) \in \mathbf{N}$

THEOREM: variable-v

$\text{variable}(v(x))$

DEFINITION: $\text{fn}(x, y) = \text{list}('f, \text{fix}(x), \text{fix}(y))$

DEFINITION: $p(x, y) = \text{list}('p, \text{fix}(x), \text{fix}(y))$

THEOREM: function-fn

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

THEOREM: predicate-p

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

;quantifer, there exists.

DEFINITION: $\text{quantifier}(x) = (x = \text{'forsome})$

DEFINITION:

$(x \cup y)$
= **if** $\text{listp}(x)$
 then if $\text{car}(x) \in y$ **then** $\text{cdr}(x) \cup y$
 else $\text{cons}(\text{car}(x), \text{cdr}(x) \cup y)$ **endif**
 else y **endif**

EVENT: Enable variable; name this event 'g0223'.

EVENT: Enable quantifier; name this event 'g0224'.

THEOREM: predicate-f-equal
predicate('equal')

EVENT: Enable function; name this event 'g0225'.

EVENT: Enable predicate; name this event 'g0226'.

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

DEFINITION:
delete(x, y)
= **if** listp(y)
 then if $x = \text{car}(y)$ **then** delete(x , cdr(y))
 else cons(car(y), delete(x , cdr(y))) **endif**
 else y **endif**

THEOREM: not-member-delete
 $x \notin \text{delete}(x, y)$

;returns list of free variables in EXP.

DEFINITION:
collect-free(exp, flg)
= **if** listp(exp)
 then if $flg = \mathbf{t}$
 then if variable(exp) **then** cons(exp , nil)
 elseif quantifier(car(exp)) \wedge listp(cdr(exp))
 then delete(cadr(exp), collect-free(cddr(exp), 'list))
 elseif func-pred(car(exp))
 \vee (car(exp) = 'not)
 \vee (car(exp) = 'or)
 then collect-free(cdr(exp), 'list)
 else nil endif
 else append(collect-free(car(exp), \mathbf{t}),
 collect-free(cdr(exp), 'list)) **endif**
 else nil endif

DEFINITION: $\text{sentence}(exp) = (\text{collect-free}(exp, \mathbf{t}) = \mathbf{nil})$

;returns bound variables in EXP that surround free occurrences of VAR.

DEFINITION:

$\text{covering}(exp, var, flg)$

```
=  if listp(exp)
    then if flg = t
        then if variable(exp) then nil
            elseif quantifier(car(exp)) ∧ listp(cdr(exp))
                then if cadr(exp) = var then nil
                    elseif var ∈ collect-free(cddr(exp), 'list)
                        then cons(cadr(exp), covering(cddr(exp), var, 'list))
                    else nil endif
                elseif func-pred(car(exp))
                    ∨ (car(exp) = 'not)
                    ∨ (car(exp) = 'or)
                then covering(cdr(exp), var, 'list)
                else nil endif
        else append(covering(car(exp), var, t),
                    covering(cdr(exp), var, 'list)) endif
    else nil endif
```

;X and Y are disjoint.

DEFINITION:

$\text{nil-intersect}(x, y)$

```
=  if listp(x) then (car(x) ∉ y) ∧ nil-intersect(cdr(x), y)
    else t endif
```

;TERM is free for VAR in EXP.

DEFINITION:

$\text{free-for}(exp, var, term, flg)$

```
=  nil-intersect(covering(exp, var, flg), collect-free(term, t))
```

DEFINITION: $\text{f-equal}(x, y) = \text{list}('equal, x, y)$

DEFINITION: $\text{f-not}(x) = \text{list}('not, x)$

DEFINITION: $\text{f-or}(x, y) = \text{list}('or, x, y)$

DEFINITION: $\text{forsome}(x, y) = \text{list}('forsome, x, y)$

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

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

DEFINITION: $\forall var\ exp = f\text{-not}(\text{forsome}(var, f\text{-not}(exp)))$

DEFINITION: $f\text{-iff}(x, y) = f\text{-and}(f\text{-implies}(x, y), f\text{-implies}(y, x))$

DEFINITION:

$\text{var-list}(list, n)$

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

DEFINITION:

$\text{var-set}(list, n)$

= **if** $n \simeq 0$ **then** $list = \text{nil}$
 else $\text{variable}(\text{car}(list))$
 $\wedge (\text{car}(list) \notin \text{cdr}(list))$
 $\wedge \text{var-set}(\text{cdr}(list), n - 1)$ **endif**

;Recognizer for terms.

DEFINITION:

$\text{term}(exp, flg, count)$

= **if** $flg = \mathbf{t}$
 then if $exp \simeq \text{nil}$ **then** **f**
 else $\text{variable}(exp)$
 $\vee (\text{function}(\text{car}(exp))$
 $\wedge \text{term}(\text{cdr}(exp),$
 ' list ,
 $\text{degree}(\text{car}(exp)))$) **endif**
 elseif $(exp \simeq \text{nil}) \vee (count \simeq 0)$ **then** $(exp = \text{nil}) \wedge (count \simeq 0)$
 else $\text{term}(\text{car}(exp), \mathbf{t}, 0) \wedge \text{term}(\text{cdr}(exp), '\text{list}, count - 1)$ **endif**

DEFINITION: $\text{arg1}(x) = \text{cadr}(x)$

DEFINITION: $\text{arg2}(x) = \text{caddr}(x)$

;EXP is an atom, pred. symbol followed by list of terms.

DEFINITION:

$\text{atomp}(exp)$

= $(\text{predicate}(\text{car}(exp)) \wedge \text{term}(\text{cdr}(exp), '\text{list}, \text{degree}(\text{car}(exp))))$

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

;EXP is a formula

DEFINITION:

$\text{formula}(exp, flg, count)$

```
=  if flg = t
    then if exp  $\simeq$  nil then f
        else atomp(exp)
             $\vee$  ((car(exp) = 'not)
                 $\wedge$  formula(cdr(exp), 'list, 1))
             $\vee$  ((car(exp) = 'or)
                 $\wedge$  formula(cdr(exp), 'list, 2))
             $\vee$  ((car(exp) = 'forsome)
                 $\wedge$  variable(cadr(exp))
                 $\wedge$  formula(cddr(exp), 'list, 1)) endif
    elseif (exp  $\simeq$  nil)  $\vee$  (count  $\simeq$  0) then (exp = nil)  $\wedge$  (count  $\simeq$  0)
    else formula(car(exp), t, 0)
         $\wedge$  formula(cdr(exp), 'list, count - 1) endif
```

;Result of substituting TERM for VAR in EXP.

DEFINITION:

$\text{subst}(exp, var, term, flg)$

```
=  if listp(exp)
    then if flg = t
        then if variable(exp)
            then if exp = var then term
                else exp endif
            elseif quantifier(car(exp))  $\wedge$  listp(cdr(exp))
                then if cadr(exp) = var then exp
                    else cons(car(exp),
                        cons(cadr(exp),
                            subst(cddr(exp), var, term, 'list))) endif
            elseif func-pred(car(exp))
                 $\vee$  (car(exp) = 'not)
                 $\vee$  (car(exp) = 'or)
                then cons(car(exp), subst(cdr(exp), var, term, 'list))
            else exp endif
        else cons(subst(car(exp), var, term, t),
            subst(cdr(exp), var, term, 'list)) endif
    else exp endif
```

DEFINITION:

$\text{neg}(exp1, exp2) = ((exp1 = \text{f-not}(exp2)) \vee (exp2 = \text{f-not}(exp1)))$

DEFINITION:

$\text{conc}(pf, flg)$

```
=  if pf  $\simeq$  nil then nil
```

```

elseif flag = t then caddr(pf)
else cons(conc(car(pf), t), conc(cdr(pf), 'list)) endif

```

DEFINITION:

```

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

```

DEFINITION: set-equal(*x*, *y*) = (subset(*x*, *y*) ∧ subset(*y*, *x*))

;The axioms: propositional, substitution, identity, equality for functions and predicates.

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

DEFINITION:

```

subst-axiom(exp, var, term)
= f-implies(subst(exp, var, term, t), forsome(var, exp))

```

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

DEFINITION:

```

pairequals(vars1, vars2, exp)
= if listp(vars1)
  then f-implies(f-equal(car(vars1), car(vars2)),
    pairequals(cdr(vars1), cdr(vars2), exp))
  else exp endif

```

DEFINITION:

```

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

```

DEFINITION:

```

assume(exp, list, flag)
= if listp(list)
  then if (caaar(list) = flag) ∧ (exp = cadar(list))
    then cdr(list)
    else assume(exp, cdr(list), flag) endif
  else f endif

```

;Proof-constructors

DEFINITION:

```

prop-axiom-proof(exp)
= list('axiom, list('prop-axiom, exp), prop-axiom(exp))

```

DEFINITION:

```
subst-axiom-proof(exp, var, term)  
= list('axiom,  
      list('subst-axiom, exp, var, term),  
      subst-axiom(exp, var, term))
```

DEFINITION:

```
ident-axiom-proof(var)  
= list('axiom, list('ident-axiom, 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('axiom,  
      list('equal-axiom1, fn, vars1, vars2),  
      equal-axiom1(vars1, vars2, fn))
```

DEFINITION:

```
equal-axiom2-proof(pr, vars1, vars2)  
= list('axiom,  
      list('equal-axiom2, pr, vars1, vars2),  
      equal-axiom2(vars1, vars2, pr))
```

DEFINITION:

```
expan-proof(a, b, pf) = list('rule, list('expan, a, b), f-or(a, b), pf)
```

DEFINITION:

```
contrac-proof(a, pf) = list('rule, list('contrac, a), a, pf)
```

DEFINITION:

```
assoc-proof(a, b, c, pf)  
= list('rule, list('assoc, a, b, c), f-or(f-or(a, b), c), pf)
```

DEFINITION:

```
cut-proof(a, b, c, pf1, pf2)  
= list('rule, list('cut, a, b, c), f-or(b, c), list(pf1, pf2))
```

DEFINITION:

```
forsome-intro-proof(var, a, b, pf)  
= list('rule, list('e-intro, var, a, b), f-implies(forsome(var, a), b), pf)
```

EVENT: Disable atomp; name this event 'g2737'.

DEFINITION: $\text{hint1}(pf) = \text{caadr}(pf)$

DEFINITION: $\text{hint2}(pf) = \text{cadadr}(pf)$

DEFINITION: $\text{hint3}(pf) = \text{caddadr}(pf)$

DEFINITION: $\text{hint4}(pf) = \text{caddadr}(pf)$

DEFINITION: $\text{sub-proof}(pf) = \text{caddr}(pf)$

;The proof-checker, PF is a proof.

DEFINITION:

$\text{prf}(pf)$

```
= if pf  $\simeq$  nil then f
  elseif car(pf) = 'axiom
  then if hint1(pf) = 'prop-axiom
    then formula(hint2(pf), t, 0)
       $\wedge$  (pf = prop-axiom-proof(hint2(pf)))
    elseif hint1(pf) = 'subst-axiom
    then formula(hint2(pf), t, 0)
       $\wedge$  variable(hint3(pf))
       $\wedge$  term(hint4(pf), t, 0)
       $\wedge$  free-for(hint2(pf), hint3(pf), hint4(pf), t)
       $\wedge$  (pf = subst-axiom-proof(hint2(pf),
                                hint3(pf),
                                hint4(pf)))
    elseif hint1(pf) = 'ident-axiom
    then variable(hint2(pf))
       $\wedge$  (pf = ident-axiom-proof(hint2(pf)))
    elseif hint1(pf) = 'equal-axiom1
    then function(hint2(pf))
       $\wedge$  var-list(hint3(pf), degree(hint2(pf)))
       $\wedge$  var-list(hint4(pf), degree(hint2(pf)))
       $\wedge$  (pf = equal-axiom1-proof(hint2(pf),
                                hint3(pf),
                                hint4(pf)))
    elseif hint1(pf) = 'equal-axiom2
    then predicate(hint2(pf))
       $\wedge$  var-list(hint3(pf), degree(hint2(pf)))
       $\wedge$  var-list(hint4(pf), degree(hint2(pf)))
       $\wedge$  (pf = equal-axiom2-proof(hint2(pf),
                                hint3(pf),
                                hint4(pf)))
  else f endif
```

```

elseif car (pf) = 'rule
then if hint1 (pf) = 'expan
  then formula (hint2 (pf), t, 0)
    ∧ (pf = expan-proof (hint2 (pf),
                        hint3 (pf),
                        sub-proof (pf)))
    ∧ (conc (sub-proof (pf), t) = hint3 (pf))
    ∧ prf (sub-proof (pf))
  elseif hint1 (pf) = 'contrac
  then (pf = contrac-proof (hint2 (pf), sub-proof (pf)))
    ∧ (conc (sub-proof (pf), t) = f-or (hint2 (pf), hint2 (pf)))
    ∧ prf (sub-proof (pf))
  elseif hint1 (pf) = 'assoc
  then (pf = assoc-proof (hint2 (pf),
                        hint3 (pf),
                        hint4 (pf),
                        sub-proof (pf)))
    ∧ (conc (sub-proof (pf), t)
      = f-or (hint2 (pf), f-or (hint3 (pf), hint4 (pf))))
    ∧ prf (sub-proof (pf))
  elseif hint1 (pf) = 'cut
  then (pf = cut-proof (hint2 (pf),
                        hint3 (pf),
                        hint4 (pf),
                        car (sub-proof (pf)),
                        cadr (sub-proof (pf))))
    ∧ (conc (sub-proof (pf), 'list)
      = list (f-or (hint2 (pf), hint3 (pf)),
              f-or (f-not (hint2 (pf)), hint4 (pf))))
    ∧ prf (car (sub-proof (pf)))
    ∧ prf (cadr (sub-proof (pf)))
  elseif hint1 (pf) = 'e-intro
  then variable (hint2 (pf))
    ∧ (pf = forsome-intro-proof (hint2 (pf),
                                hint3 (pf),
                                hint4 (pf),
                                sub-proof (pf)))
    ∧ (hint2 (pf) ∉ collect-free (hint4 (pf), t))
    ∧ (conc (sub-proof (pf), t)
      = f-implies (hint3 (pf), hint4 (pf)))
    ∧ prf (sub-proof (pf))
  else f endif
else f endif

```

THEOREM: formula-or-reduc
 $\text{formula}(\text{list}(' \text{or}, a, b), \mathbf{t}, 0) = (\text{formula}(a, \mathbf{t}, 0) \wedge \text{formula}(b, \mathbf{t}, 0))$

THEOREM: formula-not-reduc
 $\text{formula}(\text{list}(' \text{not}, a), \mathbf{t}, 0) = \text{formula}(a, \mathbf{t}, 0)$

THEOREM: formula-forsome-reduc
 $\text{formula}(\text{list}(' \text{forsome}, x, a), \mathbf{t}, 0) = (\text{variable}(x) \wedge \text{formula}(a, \mathbf{t}, 0))$

;PF is a valid proof of EXP.

DEFINITION:
 $\text{proves}(pf, exp) = ((\text{conc}(pf, \mathbf{t}) = exp) \wedge \text{formula}(exp, \mathbf{t}, 0) \wedge \text{prf}(pf))$

THEOREM: proves-is-formula
 $\text{proves}(pf, exp) \rightarrow \text{formula}(exp, \mathbf{t}, 0)$

THEOREM: proves-is-formula-again
 $(\neg \text{formula}(exp, \mathbf{t}, 0)) \rightarrow (\neg \text{proves}(pf, exp))$

;Getting rid of PRF by lemmas.

THEOREM: prop-axiom-proves
 $(\text{formula}(exp, \mathbf{t}, 0) \wedge (\text{concl} = \text{f-or}(\text{f-not}(exp), exp)))$
 $\rightarrow \text{proves}(\text{prop-axiom-proof}(exp), \text{concl})$

THEOREM: subst-axiom-proves
 $(\text{formula}(\text{concl}, \mathbf{t}, 0)$
 $\wedge \text{variable}(var)$
 $\wedge \text{term}(term, \mathbf{t}, 0)$
 $\wedge \text{free-for}(exp, var, term, \mathbf{t})$
 $\wedge (\text{concl} = \text{subst-axiom}(exp, var, term)))$
 $\rightarrow \text{proves}(\text{subst-axiom-proof}(exp, var, term), \text{concl})$

THEOREM: equal-axiom1-proves
 $(\text{function}(fn)$
 $\wedge \text{var-list}(vars1, \text{degree}(fn))$
 $\wedge \text{var-list}(vars2, \text{degree}(fn))$
 $\wedge \text{formula}(\text{concl}, \mathbf{t}, 0)$
 $\wedge (\text{concl} = \text{equal-axiom1}(vars1, vars2, fn)))$
 $\rightarrow \text{proves}(\text{equal-axiom1-proof}(fn, vars1, vars2), \text{concl})$

THEOREM: equal-axiom2-proves
 $(\text{predicate}(pr)$
 $\wedge \text{var-list}(vars1, \text{degree}(pr))$
 $\wedge \text{var-list}(vars2, \text{degree}(pr))$
 $\wedge \text{formula}(\text{concl}, \mathbf{t}, 0)$
 $\wedge (\text{concl} = \text{equal-axiom2}(vars1, vars2, pr)))$
 $\rightarrow \text{proves}(\text{equal-axiom2-proof}(pr, vars1, vars2), \text{concl})$

THEOREM: ident-axiom-proves
 $(\text{variable}(var) \wedge (\text{concl} = \text{ident-axiom}(var)) \wedge \text{formula}(\text{concl}, \mathbf{t}, 0))$
 $\rightarrow \text{proves}(\text{ident-axiom-proof}(var), \text{concl})$

THEOREM: expan-proof-proves
 $(\text{formula}(a, \mathbf{t}, 0) \wedge \text{proves}(pf, b) \wedge (\text{concl} = \text{f-or}(a, b)))$
 $\rightarrow \text{proves}(\text{expan-proof}(a, b, pf), \text{concl})$

THEOREM: contrac-proof-proves
 $\text{proves}(pf, \text{f-or}(a, a)) \rightarrow \text{proves}(\text{contrac-proof}(a, pf), a)$

THEOREM: assoc-proof-proves
 $(\text{proves}(pf, \text{f-or}(a, \text{f-or}(b, c))) \wedge (\text{concl} = \text{f-or}(\text{f-or}(a, b), c)))$
 $\rightarrow \text{proves}(\text{assoc-proof}(a, b, c, pf), \text{concl})$

THEOREM: cut-proof-proves
 $(\text{proves}(pf1, \text{f-or}(a, b))$
 $\wedge \text{proves}(pf2, \text{f-or}(\text{f-not}(a), c))$
 $\wedge (\text{concl} = \text{f-or}(b, c)))$
 $\rightarrow \text{proves}(\text{cut-proof}(a, b, c, pf1, pf2), \text{concl})$

;disabling the proof-constructors since the lemmas above show they work.

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 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'.

THEOREM: *forsome-intro-proves*
 (proves (*pf*, f-implies (*a*, *b*))
 \wedge (*var* \notin collect-free (*b*, *t*))
 \wedge variable (*var*)
 \wedge (*a-prime* = f-implies (forsome (*var*, *a*), *b*)))
 \rightarrow proves (forsome-intro-proof (*var*, *a*, *b*, *pf*), *a-prime*)

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

EVENT: Enable *prf*; name this event 'g2764'.

EVENT: Enable *proves*; name this event 'g2765'.

DEFINITION:
 commut-proof (*a*, *b*, *pf*) = cut-proof (*a*, *b*, *a*, *pf*, prop-axiom-proof (*a*))
;The first derived inference rule - commutativity of disjunction.

THEOREM: *commut-proof-proves*
 (proves (*pf*, f-or (*a*, *b*)) \wedge formula (f-or (*a*, *b*), *t*, 0) \wedge (*concl* = f-or (*b*, *a*)))
 \rightarrow proves (commut-proof (*a*, *b*, *pf*), *concl*)

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

;Modus Ponens .

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))

THEOREM: *detach-proof-proves1*
 (proves (*pf1*, *a*) \wedge proves (*pf2*, f-implies (*a*, *b*)) \wedge formula (*b*, *t*, 0))
 \rightarrow proves (detach-proof (*a*, *b*, *pf1*, *pf2*), *b*)

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

DEFINITION:
 proves-list (*pflist*, *explist*)

```
=  if explist  $\simeq$  nil then pflist = nil
    else proves(car(pflist), car(explist))
         $\wedge$  proves-list(cdr(pflist), cdr(explist)) endif
```

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
```

;Chained Modus Ponens.

THEOREM: detach-list-implies

```
(list(c)
  $\wedge$  proves(pf, a)
  $\wedge$  proves(pf2, list-implies(cons(a, c), b))
  $\wedge$  formula(a, t, 0)
  $\wedge$  formula(list-implies(c, b), t, 0))
 $\rightarrow$  proves(detach-proof(a, list-implies(c, b), pf, pf2), list-implies(c, b))
```

THEOREM: formula-list-implies

```
(formula(list-implies(alist, b), t, 0)  $\wedge$  listp(alist))
 $\rightarrow$  formula(list-implies(cdr(alist), b), t, 0)
```

THEOREM: detach-rule-corr

```
(proves-list(pflist, alist)
  $\wedge$  proves(pf2, list-implies(alist, b))
  $\wedge$  formula(b, t, 0))
 $\rightarrow$  proves(list-detach-proof(alist, b, pflist, pf2), b)
```

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

EVENT: Enable detach-list-implies; name this event 'g0221'.

DEFINITION:

$\text{rt-expan-proof}(a, b, pf) = \text{commut-proof}(b, a, \text{expan-proof}(b, a, pf))$

THEOREM: $\text{rt-expan-proof-proves}$

$(\text{proves}(pf, a) \wedge \text{formula}(b, \mathbf{t}, 0) \wedge (\text{concl} = \text{f-or}(a, b)))$
 $\rightarrow \text{proves}(\text{rt-expan-proof}(a, b, pf), \text{concl})$

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

;Takes list of formulas and returns disjunction.

DEFINITION:

$\text{make-disjunct}(flist)$

$=$ **if** $flist \simeq \mathbf{nil}$ **then** \mathbf{nil}
 elseif $\text{cdr}(flist) \simeq \mathbf{nil}$ **then** $\text{car}(flist)$
 else $\text{f-or}(\text{car}(flist), \text{make-disjunct}(\text{cdr}(flist)))$ **endif**

DEFINITION:

$\text{m1-proof}(exp, flist, pf)$

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

THEOREM: m1-proof-proves1

$(\text{formula}(\text{make-disjunct}(flist), \mathbf{t}, 0) \wedge (exp \in flist) \wedge \text{proves}(pf, exp))$
 $\rightarrow \text{proves}(\text{m1-proof}(exp, flist, pf), \text{make-disjunct}(flist))$

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

DEFINITION:

$\text{rt-assoc-proof}(a, b, c, pf)$

$=$ $\text{commut-proof}(\text{f-or}(b, c),$
 $a,$
 $\text{assoc-proof}(b,$
 $c,$
 $a,$
 $\text{commut-proof}(\text{f-or}(c, a),$
 $b,$
 $\text{assoc-proof}(c,$
 $a,$

$b,$
 $\text{commut-proof}(\text{f-or}(a,$
 $\qquad\qquad\qquad b),$
 $\qquad\qquad\qquad c,$
 $\qquad\qquad\qquad pf))))))$

THEOREM: rt-assoc-proof-proves
 (proves (pf , f-or (f-or (a , b), c))
 \wedge formula (a , \mathbf{t} , 0)
 \wedge formula (b , \mathbf{t} , 0)
 \wedge formula (c , \mathbf{t} , 0)
 \wedge ($concl = \text{f-or}(a, \text{f-or}(b, c))$))
 \rightarrow proves (rt-assoc-proof(a , b , c , pf), $concl$)

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

DEFINITION:
 insert-proof(a , b , c , pf)
 $=$ commut-proof(f-or(a , c),
 $\qquad\qquad\qquad b,$
 $\qquad\qquad\qquad \text{assoc-proof}(a,$
 $\qquad\qquad\qquad\qquad\qquad c,$
 $\qquad\qquad\qquad\qquad\qquad b,$
 $\qquad\qquad\qquad\qquad\qquad \text{expan-proof}(a, \text{f-or}(c, b), \text{commut-proof}(b, c, pf))))$

THEOREM: insert-proof-proves
 (proves (pf , f-or (b , c))
 \wedge formula (a , \mathbf{t} , 0)
 \wedge formula (b , \mathbf{t} , 0)
 \wedge formula (c , \mathbf{t} , 0)
 \wedge ($concl = \text{f-or}(b, \text{f-or}(a, c))$))
 \rightarrow proves (insert-proof(a , b , c , pf), $concl$)

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

DEFINITION:
 m2-proof-step($exp1$, $exp2$, $flist$, pf)
 $=$ **if** $flist \simeq \mathbf{nil}$ **then** \mathbf{nil}
 \quad **elseif** $\text{cdr}(flist) \simeq \mathbf{nil}$
 \quad **then if** $exp2 = \text{car}(flist)$ **then** pf
 $\quad\quad$ **else** \mathbf{nil} **endif**
 \quad **elseif** $exp2 = \text{car}(flist)$
 \quad **then** rt-assoc-proof($exp1$,
 $\qquad\qquad\qquad exp2,$


```

                                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), t, 0)
 \wedge (exp2 \in flist)
 \wedge formula (exp1, t, 0)
 \wedge formula (exp2, t, 0)
 \wedge proves (pf, f-or (exp1, exp2)))
 \rightarrow proves (m2-proof-step (exp1, exp2, flist, pf),
 f-or (exp1, make-disjunct (flist)))

THEOREM: m2-proof-step-proves1
 (formula (make-disjunct (flist), t, 0)
 \wedge (exp2 \in flist)
 \wedge formula (exp1, t, 0)
 \wedge formula (exp2, t, 0)
 \wedge proves (pf, f-or (exp1, exp2))
 \wedge (concl = f-or (exp1, make-disjunct (flist))))
 \rightarrow proves (m2-proof-step (exp1, exp2, flist, pf), concl)

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  $\simeq$  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

```

THEOREM: m1-proof-proves

(formula (make-disjunct (*flist*), **t**, 0)
 ∧ (*exp* ∈ *flist*)
 ∧ proves (*pf*, *exp*)
 ∧ (*concl* = make-disjunct (*flist*)))
 → proves (m1-proof (*exp*, *flist*, *pf*), *concl*)

THEOREM: m2-proof-proves

(formula (make-disjunct (*flist*), **t**, 0)
 ∧ formula (*exp1*, **t**, 0)
 ∧ formula (*exp2*, **t**, 0)
 ∧ (*exp1* ∈ *flist*)
 ∧ (*exp2* ∈ *flist*)
 ∧ proves (*pf*, f-or (*exp1*, *exp2*)))
 → proves (m2-proof (*exp1*, *exp2*, *flist*, *pf*), make-disjunct (*flist*))

THEOREM: m2-proof-proves1

(formula (make-disjunct (*flist*), **t**, 0)
 ∧ formula (*exp1*, **t**, 0)
 ∧ formula (*exp2*, **t**, 0)
 ∧ (*exp1* ∈ *flist*)
 ∧ (*exp2* ∈ *flist*)
 ∧ proves (*pf*, f-or (*exp1*, *exp2*)))
 ∧ (*concl* = make-disjunct (*flist*)))
 → proves (m2-proof (*exp1*, *exp2*, *flist*, *pf*), *concl*)

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 (fli
                                exp1),
                                flist2),
                                assoc-proof (make-disjunct (exp1,
                                exp2,
                                commut-proof (

```

DEFINITION:

```

m-proof (flist1, flist2, pf)
=  if flist1  $\simeq$  nil then nil
    elseif cdr (flist1)  $\simeq$  nil then m1-proof (car (flist1), flist2, pf)
    elseif cddr (flist1)  $\simeq$  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

```

THEOREM: subset-cons

$\text{subset}(x, y) \rightarrow \text{subset}(x, \text{cons}(z, y))$

DEFINITION:

```

form-list (flist)
=  if listp (flist)
    then formula (car (flist), t, 0)  $\wedge$  form-list (cdr (flist))
    else t endif

```

THEOREM: formlist-formula-make-disj

$(\text{form-list}(\textit{flist}) \wedge \text{listp}(\textit{flist})) \rightarrow \text{formula}(\text{make-disjunct}(\textit{flist}), \text{t}, 0)$

THEOREM: m3-proof-proves
 (formula (*exp1*, **t**, 0)
 \wedge formula (*exp2*, **t**, 0)
 \wedge form-list (*flist2*)
 \wedge proves (*pf*, make-disjunct (cons (f-or (*exp1*, *exp2*), *flist2*)))
 \wedge (*exp1* \in *flist2*)
 \wedge (*exp2* \in *flist2*)
 \rightarrow proves (m3-proof (*exp1*, *exp2*, *flist2*, *pf*), make-disjunct (*flist2*)))

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

THEOREM: m3-proof-proves1
 (formula (*exp1*, **t**, 0)
 \wedge formula (*exp2*, **t**, 0)
 \wedge form-list (*flist2*)
 \wedge proves (*pf*, make-disjunct (cons (f-or (*exp1*, *exp2*), *flist2*)))
 \wedge (*exp1* \in *flist2*)
 \wedge (*exp2* \in *flist2*)
 \wedge (*concl* = make-disjunct (*flist2*)))
 \rightarrow proves (m3-proof (*exp1*, *exp2*, *flist2*, *pf*), *concl*)

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

;The subset lemma

THEOREM: m-proof-proves
 (form-list (*flist1*)
 \wedge listp (*flist1*)
 \wedge form-list (*flist2*)
 \wedge listp (*flist2*)
 \wedge subset (*flist1*, *flist2*)
 \wedge proves (*pf*, make-disjunct (*flist1*)))
 \rightarrow proves (m-proof (*flist1*, *flist2*, *pf*), make-disjunct (*flist2*)))

EVENT: Enable m-proof; name this event 'g0247'.

THEOREM: m-proof-proves1
 (form-list (*flist1*)
 \wedge form-list (*flist2*)
 \wedge subset (*flist1*, *flist2*)
 \wedge proves (*pf*, make-disjunct (*flist1*)))
 \wedge (*concl* = make-disjunct (*flist2*)))
 \rightarrow proves (m-proof (*flist1*, *flist2*, *pf*), *concl*)

;Disjunctions.

DEFINITION: $\text{or-type}(exp) = (exp = \text{f-or}(\text{cadr}(exp), \text{caddr}(exp)))$

;Negation of disjunctions.

DEFINITION:

$\text{nor-type}(exp) = (exp = \text{f-not}(\text{f-or}(\text{cadadr}(exp), \text{caddadr}(exp))))$

EVENT: Enable atomp; name this event 'g0250'.

;Elementary and negation of elementary formulas

DEFINITION:

$\text{elem-form}(exp) = (\text{atomp}(exp) \vee (exp = \text{forsome}(\text{cadr}(exp), \text{caddr}(exp))))$

DEFINITION:

$\text{neg-elem-form}(exp)$
 $= ((exp = \text{f-not}(\text{cadr}(exp))) \wedge \text{elem-form}(\text{arg1}(exp)))$

DEFINITION:

$\text{prop-atomp}(exp) = (\text{elem-form}(exp) \vee \text{neg-elem-form}(exp))$

;Double-negations

DEFINITION:

$\text{dble-neg-type}(exp) = (exp = \text{f-not}(\text{f-not}(\text{cadadr}(exp))))$

EVENT: Disable atomp; name this event 'g0251'.

THEOREM: $\text{dble-neg-not-prop-atomp}$

$\text{dble-neg-type}(exp) \rightarrow (\neg \text{prop-atomp}(exp))$

THEOREM: $\text{or-type-not-prop-atomp}$

$\text{or-type}(exp) \rightarrow (\neg \text{prop-atomp}(exp))$

THEOREM: $\text{nor-type-not-prop-atomp}$

$\text{nor-type}(exp) \rightarrow (\neg \text{prop-atomp}(exp))$

DEFINITION:

$\text{list-count}(list)$

$= \text{if } list \simeq \text{nil} \text{ then } 0$
 $\text{else } (1 + \text{count}(\text{car}(list))) + \text{list-count}(\text{cdr}(list)) \text{ endif}$

DEFINITION:

neg-list (*exp*, *list*)

= **if** *list* \simeq **nil** **then** **f**
 else neg (*exp*, car (*list*)) \vee neg-list (*exp*, cdr (*list*)) **endif**

THEOREM: lessp-list-count

listp (*flist*) \rightarrow (list-count (cdr (*flist*)) < list-count (*flist*))

THEOREM: or-type-list-count

(or-type (car (*flist*)) \wedge listp (*flist*))
 \rightarrow (list-count (cons (cadar (*flist*), cons (caddar (*flist*), cdr (*flist*))))
 < list-count (*flist*))

THEOREM: nor-type-list-count1

(listp (*flist*) \wedge nor-type (car (*flist*)))
 \rightarrow (list-count (cons (list ('not, cadadar (*flist*)), cdr (*flist*)))
 < list-count (*flist*))

THEOREM: nor-type-list-count2

(listp (*flist*) \wedge nor-type (car (*flist*)))
 \rightarrow (list-count (cons (list ('not, caddadar (*flist*)), cdr (*flist*)))
 < list-count (*flist*))

THEOREM: dble-neg-list-count

(listp (*flist*) \wedge dble-neg-type (car (*flist*)))
 \rightarrow (list-count (cons (cadadar (*flist*), cdr (*flist*))) < list-count (*flist*))

EVENT: Enable prop-atomp; name this event 'g0230'.

EVENT: Enable or-type; name this event 'g0237'.

EVENT: Enable nor-type; name this event 'g0238'.

EVENT: Enable dble-neg-type; name this event 'g0239'.

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

;The tautology-checker.

DEFINITION:

tautologyp1 (*flist*, *auxlist*)

= **if** *flist* \simeq **nil** **then** **f**
 elseif prop-atomp (car (*flist*))

```

then neg-list (car (flist), auxlist)
       $\vee$  tautologyp1 (cdr (flist), cons (car (flist), auxlist))
elseif or-type (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 (arg1 (car (flist)))), cdr (flist)),
                  auxlist)
       $\wedge$  tautologyp1 (cons (f-not (arg2 (arg1 (car (flist)))), cdr (flist)),
                  auxlist)
elseif dble-neg-type (car (flist))
then tautologyp1 (cons (arg1 (arg1 (car (flist))), cdr (flist)), auxlist)
else f endif

```

THEOREM: form-list-append
 $(\text{form-list}(x) \wedge \text{form-list}(y)) \rightarrow \text{form-list}(\text{append}(x, y))$

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
 $(\text{formula}(\text{exp1}, \mathbf{t}, 0) \wedge \text{formula}(\text{exp2}, \mathbf{t}, 0) \wedge \text{neg}(\text{exp1}, \text{exp2}) \wedge (\text{concl} = \text{f-or}(\text{exp1}, \text{exp2}))) \rightarrow \text{proves}(\text{neg-proof}(\text{exp1}, \text{exp2}), \text{concl})$

EVENT: Enable neg-proof; name this event 'g0245'.

THEOREM: neg-list-reduc
neg-list(*exp*, *flist*)
= $((\text{f-not}(\text{exp}) \in \text{flist}) \vee ((\text{exp} = \text{f-not}(\text{cadr}(\text{exp}))) \wedge (\text{cadr}(\text{exp}) \in \text{flist})))$

DEFINITION:
neg-list-proof(*exp*, *flist*)
= **if** f-not(*exp*) \in *flist*
then m2-proof(*exp*,
f-not(*exp*),
cons(*exp*, *flist*),
neg-proof(*exp*, f-not(*exp*)))

```

else m2-proof(exp,
               cadr(exp),
               cons(exp, flist),
               neg-proof(exp, cadr(exp))) endif

```

THEOREM: neg-list-proof-proves
 (formula(*exp*, **t**, 0)
 \wedge form-list(*flist*)
 \wedge neg-list(*exp*, *flist*)
 \wedge (*concl* = make-disjunct(cons(*exp*, *flist*))))
 \rightarrow proves(neg-list-proof(*exp*, *flist*), *concl*)

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

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: nlistp-neg-list
 (*list* \simeq **nil**) \rightarrow (\neg neg-list(*exp*, *list*))

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*)
 \wedge listp(*flist1*)
 \wedge form-list(*flist2*)
 \wedge neg-list(car(*flist1*), *flist2*)
 \wedge (*concl* = make-disjunct(append(*flist1*, *flist2*))))
 \rightarrow proves(prop-atom-proof1(*flist1*, *flist2*), *concl*)

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*)) \wedge form-list (*list2*))
 \rightarrow form-list (append (*list1*, cons (*exp*, *list2*)))

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*)
 \wedge listp (*flist1*)
 \wedge form-list (*flist2*)
 \wedge proves (*pf*,
 make-disjunct (append (cdr (*flist1*), cons (car (*flist1*), *flist2*))))
 \wedge (*concl* = make-disjunct (append (*flist1*, *flist2*))))
 \rightarrow proves (prop-atom-proof2 (*flist1*, *flist2*, *pf*), *concl*)

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*))
 \wedge proves (*pf2*, f-or (*a*, *b*))
 \wedge formula (*a*, *t*, 0)
 \wedge formula (*b*, *t*, 0))
 \rightarrow proves (cancel-proof (*a*, *b*, *pf1*, *pf2*), *b*)

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*))))

THEOREM: nlistp-nor-type-proof-proves
 (formula (*a*, **t**, 0)
 ∧ formula (*b*, **t**, 0)
 ∧ proves (*pf1*, f-not (*a*))
 ∧ proves (*pf2*, f-not (*b*))
 ∧ (*concl* = f-not (f-or (*a*, *b*))))
 → proves (nlistp-nor-type-proof (*a*, *b*, *pf1*, *pf2*), *concl*)

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))))

THEOREM: listp-nor-type-proof-proves
 (formula (*a*, **t**, 0)
 ∧ formula (*b*, **t**, 0)
 ∧ formula (*c*, **t**, 0)
 ∧ proves (*pf1*, f-or (f-not (*a*), *c*))

\wedge proves($pf2$, f-or(f-not(b), c))
 \wedge ($concl =$ f-or(f-not(f-or(a , b)), c)))
 \rightarrow proves(listp-nor-type-proof(a , b , c , $pf1$, $pf2$), $concl$)

EVENT: Enable m-proof-proves; name this event 'g0242'.

EVENT: Enable nlistp-nor-type-proof; name this event 'g0243'.

EVENT: Enable listp-nor-type-proof; name this event 'g0244'.

DEFINITION:

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

EVENT: Disable nor-type; name this event 'g0292'.

THEOREM: nor-type-proof-proves

(nor-type(exp)
 \wedge formula(exp , \mathbf{t} , 0)
 \wedge form-list($clist$)
 \wedge proves($pf1$, make-disjunct(cons(f-not(cadadr(exp)), $clist$)))
 \wedge proves($pf2$, make-disjunct(cons(f-not(caddadr(exp)), $clist$)))
 \wedge ($concl =$ make-disjunct(cons(exp , $clist$))))
 \rightarrow proves(nor-type-proof(cadadr(exp), caddadr(exp), $clist$, $pf1$, $pf2$), $concl$)

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))))))

EVENT: Enable nor-type-proof; name this event 'g0248'.

THEOREM: nlistp-dble-neg-proof-proves

(formula(a , \mathbf{t} , 0) \wedge proves(pf , a) \wedge ($concl =$ f-not(f-not(a))))
 \rightarrow proves(nlistp-dble-neg-proof(a , pf), $concl$)

EVENT: Enable nlistp-dble-neg-proof; name this event ‘g0249’.

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))))))
```

THEOREM: listp-dble-neg-proof-proves

```
(formula(a, t, 0)
 ∧ formula(c, t, 0)
 ∧ proves(pf, f-or(a, c))
 ∧ (concl = f-or(f-not(f-not(a)), c)))
→  proves(listp-dble-neg-proof(a, c, pf), concl)
```

EVENT: Enable listp-dble-neg-proof; name this event ‘g0203’.

DEFINITION:

```
dble-neg-type-proof(a, clist, pf)
=  if clist  $\simeq$  nil then nlistp-dble-neg-proof(a, pf)
    else listp-dble-neg-proof(a, make-disjunct(clist), pf) endif
```

EVENT: Disable dble-neg-type; name this event ‘g0252’.

THEOREM: dble-neg-type-proof-proves

```
(dble-neg-type(exp)
 ∧ formula(exp, t, 0)
 ∧ form-list(clist)
 ∧ proves(pf, make-disjunct(cons(cadadr(exp), clist)))
 ∧ (concl = make-disjunct(cons(exp, clist))))
→  proves(dble-neg-type-proof(cadadr(exp), clist, pf), concl)
```

DEFINITION:

```
or-type-proof(a, b, clist, pf)
=  if clist  $\simeq$  nil then pf
    else assoc-proof(a, b, make-disjunct(clist), pf) endif
```

EVENT: Disable or-type; name this event ‘g0260’.

THEOREM: or-type-proof-proves

```
(or-type (car (flist1))
  ∧ form-list (flist1)
  ∧ listp (flist1)
  ∧ form-list (flist2)
  ∧ proves (pf,
    make-disjunct (append (cons (cadar (flist1),
      cons (caddar (flist1), cdr (flist1))),
      flist2)))
  ∧ (concl = make-disjunct (append (flist1, flist2))))
→ proves (or-type-proof (cadar (flist1),
  caddar (flist1),
  append (cdr (flist1), flist2),
  pf),
  concl)
```

EVENT: Enable or-type-proof; name this event 'g0271'.

THEOREM: or-type-form-list

```
(or-type (car (flist)) ∧ form-list (flist) ∧ listp (flist))
→ form-list (cons (cadar (flist), cons (caddar (flist), cdr (flist))))
```

THEOREM: nor-type-form-list

```
(nor-type (car (flist)) ∧ form-list (flist) ∧ listp (flist))
→ form-list (cons (list ('not, cadadar (flist)), cdr (flist)))
```

THEOREM: nor-type-form-list2

```
(nor-type (car (flist)) ∧ form-list (flist) ∧ listp (flist))
→ form-list (cons (list ('not, caddadar (flist)), cdr (flist)))
```

THEOREM: dble-neg-type-form-list

```
(dbler-neg-type (car (flist)) ∧ form-list (flist) ∧ listp (flist))
→ form-list (cons (cadadar (flist), cdr (flist)))
```

EVENT: Enable or-type; name this event 'g0272'.

EVENT: Enable nor-type; name this event 'g0273'.

EVENT: Enable dble-neg-type; name this event 'g0274'.

EVENT: Enable dble-neg-type-proof; name this event 'g0254'.

;The proof-constructor for tautologies.

DEFINITION:

```

taut-proof1 (flist, auxlist)
=  if flist  $\simeq$  nil then nil
    elseif prop-atomp (car (flist))
    then if 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
    elseif or-type (car (flist))
    then or-type-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-type-proof (arg1 (arg1 (car (flist))),
                        arg2 (arg1 (car (flist))),
                        append (cdr (flist), auxlist),
                        taut-proof1 (cons (f-not (arg1 (arg1 (car (flist)))),
                                          cdr (flist)),
                                      auxlist),
                        taut-proof1 (cons (f-not (arg2 (arg1 (car (flist)))),
                                          cdr (flist)),
                                      auxlist))
    elseif dble-neg-type (car (flist))
    then dble-neg-type-proof (arg1 (arg1 (car (flist))),
                             append (cdr (flist), auxlist),
                             taut-proof1 (cons (arg1 (arg1 (car (flist))),
                                               cdr (flist)),
                                           auxlist))
    else nil endif

```

;Every tautology has a proof (when AUXLIST is NIL)

THEOREM: taut-thm1

```

(form-list (flist)  $\wedge$  form-list (auxlist)  $\wedge$  tautologyp1 (flist, auxlist))
 $\rightarrow$   proves (taut-proof1 (flist, auxlist),
                    make-disjunct (append (flist, auxlist)))

```

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

THEOREM: taut-thm2

```

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

```

THEOREM: listp-elem-form
 $(exp \simeq \mathbf{nil}) \rightarrow (\neg \text{elem-form}(exp))$

;Truth value evaluator.

DEFINITION:
eval(*exp*, *alist*)
= **if** *exp* \simeq **nil** **then** **f**
 elseif elem-form(*exp*) **then** *exp* \in *alist*
 elseif car(*exp*) = '**not** **then** \neg eval(cadr(*exp*), *alist*)
 elseif car(*exp*) = '**or**
 then eval(cadr(*exp*), *alist*) \vee eval(caddr(*exp*), *alist*)
 else f endif

THEOREM: elem-form-eval
 $\text{elem-form}(exp) \rightarrow (\text{eval}(exp, alist) = (exp \in alist))$

THEOREM: nlistp-eval
 $(exp \simeq \mathbf{nil}) \rightarrow (\neg \text{eval}(exp, alist))$

THEOREM: not-eval
 $(\text{listp}(exp) \wedge (\text{car}(exp) = \mathbf{'not}))$
 $\rightarrow (\text{eval}(exp, alist) = (\neg \text{eval}(\text{cadr}(exp), alist)))$

THEOREM: or-eval
 $(\text{listp}(exp) \wedge (\text{car}(exp) = \mathbf{'or}))$
 $\rightarrow (\text{eval}(exp, alist)$
 $\quad = (\text{eval}(\text{cadr}(exp), alist) \vee \text{eval}(\text{caddr}(exp), alist)))$

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

THEOREM: eval-elem-form
 $(\text{elem-form}(exp)$
 $\quad \wedge (exp \in list)$
 $\quad \wedge (exp \in alist)$
 $\quad \wedge (concl = \text{make-disjunct}(list)))$
 $\rightarrow \text{eval}(concl, alist)$

EVENT: Disable or-type; name this event 'g0278'.

EVENT: Disable nor-type; name this event 'g0279'.

EVENT: Disable dble-neg-type; name this event 'g0280'.

EVENT: Disable prop-atomp; name this event 'g0281'.

THEOREM: member-append

$$(exp \in \text{append}(flist1, flist2)) = ((exp \in flist1) \vee (exp \in flist2))$$

THEOREM: eval-neg-elem-form

$$\begin{aligned} & ((exp \in list) \\ & \wedge \quad (\text{f-not}(exp) \in list) \\ & \wedge \quad (concl = \text{make-disjunct}(list))) \\ & \rightarrow \quad \text{eval}(concl, alist) \end{aligned}$$

THEOREM: eval-make-disjunct

$$\begin{aligned} & \text{eval}(\text{make-disjunct}(\text{append}(list1, list2)), alist) \\ & = \quad (\text{eval}(\text{make-disjunct}(list1), alist) \\ & \quad \vee \quad \text{eval}(\text{make-disjunct}(list2), alist)) \end{aligned}$$

THEOREM: neg-list-eval

$$\begin{aligned} & (\text{listp}(flist1) \\ & \wedge \quad \text{neg-list}(\text{car}(flist1), flist2) \\ & \wedge \quad (concl = \text{make-disjunct}(\text{append}(flist1, flist2)))) \\ & \rightarrow \quad \text{eval}(concl, alist) \end{aligned}$$

THEOREM: eval-prop-atomp

$$\begin{aligned} & (\text{listp}(flist1) \\ & \wedge \quad \text{eval}(\text{make-disjunct}(\text{append}(\text{cdr}(flist1), \text{cons}(\text{car}(flist1), flist2))), \\ & \quad \quad \quad alist)) \\ & \rightarrow \quad \text{eval}(\text{make-disjunct}(\text{append}(flist1, flist2)), alist) \end{aligned}$$

EVENT: Enable eval; name this event 'g1253'.

THEOREM: eval-or-type

$$\begin{aligned} & (\text{listp}(flist1) \wedge \text{or-type}(\text{car}(flist1))) \\ & \rightarrow \quad (\text{eval}(\text{make-disjunct}(\text{append}(flist1, flist2)), alist) \\ & \quad = \quad \text{eval}(\text{make-disjunct}(\text{append}(\text{cons}(\text{cadar}(flist1), \\ & \quad \quad \quad \text{cons}(\text{caddar}(flist1), \\ & \quad \quad \quad \text{cdr}(flist1))), \\ & \quad \quad \quad flist2)), \\ & \quad \quad \quad alist)) \end{aligned}$$

THEOREM: eval-nor-type

$$\begin{aligned} & (\text{listp } (flist1) \wedge \text{nor-type } (\text{car } (flist1))) \\ \rightarrow & \quad (\text{eval } (\text{make-disjunct } (\text{append } (flist1, flist2)), alist) \\ & \quad = \quad (\text{eval } (\text{make-disjunct } (\text{append } (\text{cons } (\text{f-not } (\text{cadadar } (flist1)), \\ & \quad \quad \quad \text{cdr } (flist1)), \\ & \quad \quad \quad flist2)), \\ & \quad \quad \quad alist) \\ & \quad \wedge \quad \text{eval } (\text{make-disjunct } (\text{append } (\text{cons } (\text{f-not } (\text{caddadar } (flist1)), \\ & \quad \quad \quad \text{cdr } (flist1)), \\ & \quad \quad \quad flist2)), \\ & \quad \quad \quad alist))) \end{aligned}$$

THEOREM: eval-dble-neg-type

$$\begin{aligned} & (\text{listp } (flist1) \wedge \text{dble-neg-type } (\text{car } (flist1))) \\ \rightarrow & \quad (\text{eval } (\text{make-disjunct } (\text{append } (flist1, flist2)), alist) \\ & \quad = \quad \text{eval } (\text{make-disjunct } (\text{append } (\text{cons } (\text{cadadar } (flist1), \text{cdr } (flist1)), \\ & \quad \quad \quad flist2)), \\ & \quad \quad \quad alist)) \end{aligned}$$

;All tautologies are logically-true.

THEOREM: taut-eval

$$\begin{aligned} & \text{tautologyp1 } (flist, auxlist) \\ \rightarrow & \quad \text{eval } (\text{make-disjunct } (\text{append } (flist, auxlist)), alist) \end{aligned}$$

THEOREM: not-eval-prop-atomp

$$\begin{aligned} & (\text{listp } (flist1) \\ & \quad \wedge \quad (\neg \text{eval } (\text{make-disjunct } (\text{append } (\text{cdr } (flist1), \text{cons } (\text{car } (flist1), flist2))), \\ & \quad \quad \quad alist))) \\ \rightarrow & \quad (\neg \text{eval } (\text{make-disjunct } (\text{append } (flist1, flist2)), alist)) \end{aligned}$$

THEOREM: prop-atomp-reduc

$$\begin{aligned} & \text{prop-atomp } (exp) \\ = & \quad (\text{elem-form } (exp) \\ & \quad \vee \quad ((exp = \text{f-not } (\text{cadr } (exp))) \wedge \text{elem-form } (\text{cadr } (exp)))) \end{aligned}$$

EVENT: Enable elem-form; name this event 'g0263'.

EVENT: Enable prop-atomp; name this event 'g0264'.

DEFINITION:

$$\begin{aligned} & \text{prop-atomp-list } (list) \\ = & \quad \text{if } list \simeq \text{nil} \text{ then t} \\ & \quad \text{else prop-atomp } (\text{car } (list)) \wedge \text{prop-atomp-list } (\text{cdr } (list)) \text{ endif} \end{aligned}$$

DEFINITION:

```
falsify (list)
=  if list  $\simeq$  nil then nil
    elseif car (list) = f-not (cadar (list))
    then cons (cadar (list), falsify (cdr (list)))
    else falsify (cdr (list)) endif
```

THEOREM: falsify-step

$$(f\text{-not}(exp) \notin auxlist) \rightarrow (exp \notin falsify(auxlist))$$

THEOREM: prop-atomp-auxlist

```
(prop-atomp (exp)
 $\wedge$  ( $\neg$  neg-list (exp, auxlist))
 $\wedge$  prop-atomp-list (auxlist))
 $\rightarrow$  ( $\neg$  eval (exp, falsify (cons (exp, auxlist)))))
```

THEOREM: prop-atomp-auxlist2

```
(( $\neg$  neg-list (exp, auxlist))
 $\wedge$  prop-atomp-list (auxlist)
 $\wedge$  prop-atomp (exp)
 $\wedge$  ( $\neg$  eval (make-disjunct (auxlist), falsify (alist))))
 $\rightarrow$  ( $\neg$  eval (make-disjunct (auxlist), falsify (cons (exp, alist)))))
```

THEOREM: prop-atomp-falsify

```
(prop-atomp (exp)
 $\wedge$  ( $\neg$  neg-list (exp, auxlist))
 $\wedge$  prop-atomp-list (auxlist)
 $\wedge$  ( $\neg$  eval (make-disjunct (auxlist), falsify (auxlist))))
 $\rightarrow$  ( $\neg$  eval (make-disjunct (cons (exp, auxlist), falsify (cons (exp, auxlist)))))
```

EVENT: Disable prop-atomp; name this event 'g0268'.

EVENT: Disable elem-form; name this event 'g0269'.

EVENT: Enable atomp; name this event 'g0204'.

THEOREM: formula-cases1

```
formula (exp, t, 0)
 $\rightarrow$  (prop-atomp (exp)
 $\vee$  or-type (exp)
 $\vee$  nor-type (exp)
 $\vee$  dble-neg-type (exp))
```

EVENT: Disable atomp; name this event 'g0205'.

EVENT: Enable prop-atomp; name this event 'g0296'.

EVENT: Enable or-type; name this event 'g0297'.

EVENT: Enable nor-type; name this event 'g0298'.

EVENT: Enable dble-neg-type; name this event 'g0299'.

THEOREM: formula-cases

$$\begin{aligned} & ((\neg \text{dble-neg-type}(exp)) \\ & \wedge (\neg \text{nor-type}(exp)) \\ & \wedge (\neg \text{or-type}(exp)) \\ & \wedge (\neg \text{prop-atomp}(exp))) \\ & \rightarrow (\neg \text{formula}(exp, t, 0)) \end{aligned}$$

DEFINITION:

falsify-taut(*flist*, *auxlist*)
= **if** *flist* \simeq **nil** **then** falsify(*auxlist*)
 elseif prop-atomp(car(*flist*))
 then if neg-list(car(*flist*), *auxlist*) **then f**
 else falsify-taut(cdr(*flist*), cons(car(*flist*), *auxlist*)) **endif**
 elseif or-type(car(*flist*))
 then falsify-taut(cons(cadar(*flist*), cons(caddar(*flist*), cdr(*flist*))),
 auxlist)
 elseif nor-type(car(*flist*))
 then if falsify-taut(cons(f-not(caddadar(*flist*)), cdr(*flist*)), *auxlist*)
 then falsify-taut(cons(f-not(caddadar(*flist*)), cdr(*flist*)),
 auxlist)
 else falsify-taut(cons(f-not(cadadar(*flist*)), cdr(*flist*)),
 auxlist) **endif**
 elseif dble-neg-type(car(*flist*))
 then falsify-taut(cons(cadadar(*flist*), cdr(*flist*)), *auxlist*)
 else nil endif

THEOREM: append-nlistp

$$(x \simeq \text{nil}) \rightarrow (\text{append}(x, y) = y)$$

THEOREM: not-falsify-taut

$$\text{tautology1}(flist, auxlist) = (\neg \text{falsify-taut}(flist, auxlist))$$

;Non-tautologies are falsifiable.

THEOREM: not-taut-false

(form-list (*flist*)
 ∧ prop-atomp-list (*auxlist*)
 ∧ (¬ eval (make-disjunct (*auxlist*), falsify (*auxlist*)))
 ∧ (¬ tautologyp1 (*flist*, *auxlist*)))
 → (¬ eval (make-disjunct (append (*flist*, *auxlist*),
 falsify-taut (*flist*, *auxlist*))))

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

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

EVENT: Disable append; name this event ‘g0300’.

THEOREM: form-list-append-nil

make-disjunct (append (*flist*, **nil**)) = make-disjunct (*flist*)

THEOREM: tautology-theorem

(form-list (*flist*) ∧ tautologyp (*flist*) ∧ (*concl* = make-disjunct (*flist*)))
 → proves (taut-proof (*flist*), *concl*)

THEOREM: taut-eval2

(tautologyp1 (*flist*, *auxlist*)
 ∧ (*concl* = make-disjunct (append (*flist*, *auxlist*))))
 → eval (*concl*, *alist*)

THEOREM: tautologies-are-true

(form-list (*flist*) ∧ tautologyp (*flist*))
 → eval (make-disjunct (*flist*), *alist*)

THEOREM: not-taut-falsify2

(form-list (*flist*)
 ∧ prop-atomp-list (*auxlist*)
 ∧ (¬ eval (make-disjunct (*auxlist*), falsify (*auxlist*)))
 ∧ (¬ tautologyp1 (*flist*, *auxlist*))
 ∧ (*concl* = make-disjunct (append (*flist*, *auxlist*))))
 → (¬ eval (*concl*, falsify-taut (*flist*, *auxlist*)))

THEOREM: truths-are-tautologies

(form-list (*flist*) ∧ (¬ tautologyp (*flist*)))
 → (¬ eval (make-disjunct (*flist*), falsify-taut (*flist*, **nil**)))

EVENT: Enable truths-are-tautologies; name this event ‘g2439’.

EVENT: Enable not-taut-falsify2; name this event ‘g2440’.

EVENT: Enable tautologies-are-true; name this event 'g2441'.

EVENT: Enable taut-eval2; name this event 'g2442'.

EVENT: Enable form-list-append-nil; name this event 'g2443'.

EVENT: Enable taut-proof; name this event 'g2444'.

EVENT: Enable not-taut-false; name this event 'g2445'.

EVENT: Enable not-falsify-taut; name this event 'g2446'.

EVENT: Enable append-nlistp; name this event 'g2447'.

EVENT: Enable falsify-taut; name this event 'g2448'.

EVENT: Enable formula-cases; name this event 'g2449'.

EVENT: Enable formula-cases1; name this event 'g2450'.

EVENT: Enable prop-atomp-falsify; name this event 'g2451'.

EVENT: Enable prop-atomp-auxlist; name this event 'g2453'.

EVENT: Enable prop-atomp-list; name this event 'g2454'.

EVENT: Enable falsify-step; name this event 'g2455'.

EVENT: Enable falsify; name this event 'g2456'.

EVENT: Enable not-eval-prop-atomp; name this event 'g2457'.

EVENT: Enable taut-eval; name this event 'g2458'.

EVENT: Enable eval-dble-neg-type; name this event 'g2459'.

EVENT: Enable eval-nor-type; name this event 'g2460'.

EVENT: Enable eval-or-type; name this event 'g2461'.

EVENT: Enable eval-prop-atomp; name this event 'g2463'.

EVENT: Enable neg-list-eval; name this event 'g2464'.

EVENT: Enable eval-neg-elem-form; name this event 'g2465'.

EVENT: Enable elem-form-eval; name this event 'g2471'.

EVENT: Enable eval; name this event 'g2472'.

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

DEFINITION:
lis-not (*flist*)
= **if** *flist* \simeq **nil** **then** **nil**
 else cons (f-not (car (*flist*)), lis-not (cdr (*flist*))) **endif**

DEFINITION:
taut-conseq (*flist*, *exp*) = tautologyp (append (lis-not (*flist*), cons (*exp*, **nil**)))

DEFINITION:
tautconseq-proof (*flist*, *exp*, *pflist*)
= list-detach-proof (*flist*,
 exp,
 pflist,
 taut-proof (append (lis-not (*flist*), cons (*exp*, **nil**))))

THEOREM: list-implies-reduc
list-implies (*flist*, *exp*)
= make-disjunct (append (lis-not (*flist*), cons (*exp*, **nil**)))

THEOREM: append-exp-form-list
(form-list (*flist*) \wedge formula (*exp*, **t**, 0))
 \rightarrow form-list (append (lis-not (*flist*), cons (*exp*, **nil**)))

THEOREM: taut-conseq-proves

```
(form-list (flist)  
  ∧ formula (exp, t, 0)  
  ∧ taut-conseq (flist, exp)  
  ∧ proves-list (pflist, flist)  
  → proves (tautconseq-proof (flist, exp, pflist), exp)
```

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

THEOREM: eval-tautconseq

```
(form-list (flist)  
  ∧ formula (exp, t, 0)  
  ∧ eval (make-disjunct (append (lis-not (flist), cons (exp, nil))),  
          falsify-taut (append (lis-not (flist), cons (exp, nil)), nil)))  
  → taut-conseq (flist, exp)
```

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

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

EVENT: Enable formula; name this event 'g0295'.

THEOREM: eval-tautconseq-proof-proves

```
(eval (make-disjunct (append (lis-not (flist), cons (exp, nil))),  
      falsify-taut (append (lis-not (flist), cons (exp, nil)), nil)))  
  ∧ proves-list (pflist, flist)  
  ∧ form-list (flist)  
  ∧ formula (exp, t, 0)  
  → proves (tautconseq-proof (flist, exp, pflist), exp)
```

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

DEFINITION:

```
f-iff-reduc-proof (a, b, pf1, pf2)  
= tautconseq-proof (list (f-iff (a, b), a), b, list (pf1, pf2))
```

Index

- append, 3
- append-exp-form-list, 38
- append-nlistp, 35
- arg1, 5, 21, 23, 30
- arg2, 5, 23, 30
- assoc-proof, 8, 10, 12, 16, 19, 28
- assoc-proof-proves, 12
- assume, 7
- atomp, 5, 6, 21

- cancel-proof, 25
- cancel-proof-proves, 25
- collect-free, 3, 4, 10, 13
- commut-proof, 13, 15–17, 19, 23, 27, 28
- commut-proof-proves, 13
- conc, 6, 7, 10, 11
- contrac-proof, 8, 10, 12, 13, 17, 19, 25, 27
- contrac-proof-proves, 12
- covering, 4
- cut-proof, 8, 10, 12, 13, 25–28
- cut-proof-proves, 12

- dbble-neg-list-count, 22
- dbble-neg-not-prop-atomp, 21
- dbble-neg-type, 21–23, 28–30, 33–35
- dbble-neg-type-form-list, 29
- dbble-neg-type-proof, 28, 30
- dbble-neg-type-proof-proves, 28
- degree, 2, 5, 9, 11
- delete, 3
- detach-list-implies, 14
- detach-proof, 13, 14
- detach-proof-proves1, 13
- detach-rule-corr, 14

- elem-form, 21, 31, 33
- elem-form-eval, 31
- equal-axiom1, 8, 11
- equal-axiom1-proof, 8, 9, 11
- equal-axiom1-proves, 11
- equal-axiom2, 7, 8, 11
- equal-axiom2-proof, 8, 9, 11
- equal-axiom2-proves, 11
- eval, 31–34, 36, 38, 39
- eval-dble-neg-type, 33
- eval-elem-form, 31
- eval-make-disjunct, 32
- eval-neg-elem-form, 32
- eval-nor-type, 33
- eval-or-type, 32
- eval-prop-atomp, 32
- eval-tautconseq, 39
- eval-tautconseq-proof-proves, 39
- eval-tautologyp, 38
- expan-proof, 8, 10, 12, 13, 15–17
- expan-proof-proves, 12

- f-and, 4, 5
- f-equal, 4, 7, 8
- f-iff, 5, 39
- f-iff-reduc-proof, 39
- f-implies, 5, 7, 8, 10, 13, 14
- f-not, 4–7, 10–12, 21, 23, 25–28, 30, 32–35, 38
- f-or, 4, 5, 7, 8, 10–13, 15–21, 23, 25–28
- falsify, 34–36
- falsify-step, 34
- falsify-taut, 35, 36, 38, 39
- fn, 2
- forall, 5
- form-list, 19, 20, 23–25, 27–31, 36, 38, 39
- form-list-append, 23
- form-list-append-car, 25
- form-list-append-nil, 36
- formlist-formula-make-disj, 19
- formula, 6, 9–20, 23–28, 34, 35, 38, 39
- formula-cases, 35

formula-cases1, 34	g0250, 21
formula-forsome-reduc, 11	g0251, 21
formula-list-implies, 14	g0252, 28
formula-not-reduc, 11	g0253, 5
formula-or-reduc, 11	g0254, 29
forsome, 4, 5, 7, 8, 13, 21	g0255, 25
forsome-intro-proof, 8, 10, 13	g0256, 24
forsome-intro-proves, 13	g0258, 25
free-for, 4, 9, 11	g0259, 24
func-pred, 2–4, 6	g0260, 28
function, 1, 2, 5, 9, 11	g0263, 33
function-fn, 2	g0264, 33
	g0268, 34
g0203, 28	g0269, 34
g0204, 34	g0271, 29
g0205, 35	g0272, 29
g0220, 14	g0273, 29
g0221, 14	g0274, 29
g0222, 20	g0275, 30
g0223, 3	g0276, 39
g0224, 3	g0277, 39
g0225, 3	g0278, 32
g0226, 3	g0279, 32
g0227, 15	g0280, 32
g0228, 15	g0281, 32
g0229, 20	g0282, 39
g0230, 22	g0283, 39
g0231, 16	g0292, 27
g0232, 16	g0295, 39
g0233, 17	g0296, 35
g0234, 17	g0297, 35
g0235, 18	g0298, 35
g0236, 18	g0299, 35
g0237, 22	g0300, 36
g0238, 22	g1253, 32
g0239, 22	g2439, 36
g0240, 22	g2440, 36
g0242, 27	g2441, 37
g0243, 27	g2442, 37
g0244, 27	g2443, 37
g0245, 23	g2444, 37
g0247, 20	g2445, 37
g0248, 27	g2446, 37
g0249, 28	g2447, 37

- g2448, 37
- g2449, 37
- g2450, 37
- g2451, 37
- g2453, 37
- g2454, 37
- g2455, 37
- g2456, 37
- g2457, 37
- g2458, 37
- g2459, 37
- g2460, 38
- g2461, 38
- g2463, 38
- g2464, 38
- g2465, 38
- g2471, 38
- g2472, 38
- g2737, 8
- g2752, 12
- g2753, 12
- g2754, 12
- g2755, 12
- g2756, 12
- g2759, 12
- g2760, 12
- g2761, 12
- g2762, 12
- g2763, 13
- g2764, 13
- g2765, 13
- g2766, 13
- g2767, 13

- hint1, 9, 10
- hint2, 9, 10
- hint3, 9, 10
- hint4, 9, 10

- ident-axiom, 7, 12
- ident-axiom-proof, 8, 9, 12
- ident-axiom-proves, 12
- index, 2
- insert-proof, 16, 17

- insert-proof-proves, 16

- lessp-list-count, 22
- lis-not, 38, 39
- list-count, 21, 22
- list-detach-proof, 14, 38
- list-implies, 14, 38
- list-implies-reduc, 38
- listp-dble-neg-proof, 28
- listp-dble-neg-proof-proves, 28
- listp-elem-form, 31
- listp-nor-type-proof, 26, 27
- listp-nor-type-proof-proves, 26

- m-proof, 19, 20, 24–26
- m-proof-proves, 20
- m-proof-proves1, 20
- m1-proof, 15, 17–19
- m1-proof-proves, 18
- m1-proof-proves1, 15
- m2-proof, 17–19, 23, 24
- m2-proof-proves, 18
- m2-proof-proves1, 18
- m2-proof-step, 16, 17
- m2-proof-step-proves, 17
- m2-proof-step-proves1, 17
- m3-proof, 18–20
- m3-proof-proves, 20
- m3-proof-proves1, 20
- make-disjunct, 15, 17–20, 24, 25, 27–34, 36, 38, 39
- member-append, 32
- member-eval, 31

- neg, 6, 22, 23
- neg-elem-form, 21
- neg-list, 22–24, 30, 32, 34, 35
- neg-list-eval, 32
- neg-list-proof, 23, 24
- neg-list-proof-proves, 24
- neg-list-reduc, 23
- neg-proof, 23, 24
- neg-proof-proves, 23
- nil-intersect, 4

- nlistp-dble-neg-proof, 27, 28
- nlistp-dble-neg-proof-proves, 27
- nlistp-eval, 31
- nlistp-neg-list, 24
- nlistp-nor-type-proof, 25–27
- nlistp-nor-type-proof-proves, 26
- nor-type, 21–23, 27, 29, 30, 33–35
- nor-type-form-list, 29
- nor-type-form-list2, 29
- nor-type-list-count1, 22
- nor-type-list-count2, 22
- nor-type-not-prop-atomp, 21
- nor-type-proof, 27, 30
- nor-type-proof-proves, 27
- not-eval, 31
- not-eval-prop-atomp, 33
- not-falsify-taut, 35
- not-member-delete, 3
- not-taut-false, 36
- not-taut-falsify2, 36
- numberp-fix, 2

- or-eval, 31
- or-type, 21–23, 29, 30, 32, 34, 35
- or-type-form-list, 29
- or-type-list-count, 22
- or-type-not-prop-atomp, 21
- or-type-proof, 28–30
- or-type-proof-proves, 29

- p, 2
- pairequals, 7, 8
- predicate, 2, 3, 5, 9, 11
- predicate-f-equal, 3
- predicate-p, 2
- prf, 9–11
- prop-atom-proof1, 24, 30
- prop-atom-proof1-proves, 24
- prop-atom-proof2, 25, 30
- prop-atom-proof2-proves, 25
- prop-atomp, 21, 22, 30, 33–35
- prop-atomp-auxlist, 34
- prop-atomp-auxlist2, 34
- prop-atomp-falsify, 34
- prop-atomp-list, 33, 34, 36
- prop-atomp-reduc, 33
- prop-axiom, 7
- prop-axiom-proof, 7, 9, 11, 13, 23, 25–28
- prop-axiom-proves, 11
- proves, 11–18, 20, 23–31, 36, 39
- proves-is-formula, 11
- proves-is-formula-again, 11
- proves-list, 13, 14, 39

- quantifier, 2–4, 6

- rt-assoc-proof, 15–17, 26
- rt-assoc-proof-proves, 16
- rt-expan-proof, 15, 17, 25, 27
- rt-expan-proof-proves, 15

- sentence, 4
- set-equal, 7
- sub-proof, 9, 10
- subset, 7, 19, 20, 24
- subset-append, 24
- subset-append-car, 24
- subset-car, 24
- subset-cons, 19
- subset-ident, 24
- subst, 6, 7
- subst-axiom, 7, 8, 11
- subst-axiom-proof, 8, 9, 11
- subst-axiom-proves, 11

- taut-conseq, 38, 39
- taut-conseq-proves, 39
- taut-eval, 33
- taut-eval2, 36
- taut-proof, 36, 38
- taut-proof1, 30, 31, 36
- taut-thm1, 30
- taut-thm2, 31
- tautconseq-proof, 38, 39
- tautologies-are-true, 36
- tautology-theorem, 36
- tautologyp, 36, 38

tautologyp1, 22, 23, 30, 31, 33, 35,
36
temp, 5, 9, 11
truths-are-tautologies, 36

union, 2

v, 2
var-list, 5, 9, 11
var-set, 5
variable, 2–6, 9–13
variable-v, 2