EVENT: Start with the library "c2".

;; These functions simply allow the accessing of various components
;; of Micro-Gypsy structures.

**Definition**: loop-body \((stmt) = \text{cadr}\ (stmt)\)

**Definition**: prog2-left-branch \((stmt) = \text{cadr}\ (stmt)\)

**Definition**: prog2-right-branch \((stmt) = \text{caddr}\ (stmt)\)

**Definition**: signalled-condition \((stmt) = \text{cadr}\ (stmt)\)

**Theorem**: Signalled-condition-expansion2
signalled-condition (cons (signal, lst)) = car (lst)

**Definition**: if-condition \((stmt) = \text{cadr}\ (stmt)\)

**Definition**: if-true-branch \((stmt) = \text{caddr}\ (stmt)\)

**Definition**: if-false-branch \((stmt) = \text{cadddr}\ (stmt)\)

**Definition**: begin-body \((stmt) = \text{cadr}\ (stmt)\)

**Definition**: when-labels \((stmt) = \text{caddr}\ (stmt)\)

**Definition**: when-handler \((stmt) = \text{cadddr}\ (stmt)\)

**Definition**: call-name \((stmt) = \text{cadr}\ (stmt)\)

**Theorem**: Call-name-expansion
\[
\text{call-name} (\text{cons} (\text{proc-call}, \text{cons} (\text{name}, y))) = \text{name}
\]

**Definition**: call-actuals \((stmt) = \text{caddr}\ (stmt)\)

**Definition**: call-conds \((stmt) = \text{cadddr}\ (stmt)\)

**Definition**: formal-type \((exp) = \text{cadr}\ (exp)\)

**Definition**: formal-initial-value \((local) = \text{caddr}\ (local)\)
Definition:  
PREDEFINED-PROCEDURE-LIST = '  (mg-simple-variable-assignment  
  mg-simple-constant-assignment mg-simple-variable-eq  
  mg-simple-constant-eq mg-integer-le  
  mg-integer-unary-minus mg-integer-add  
  mg-integer-subtract mg-boolean-or mg-boolean-and  
  mg-boolean-not mg-index-array mg-array-element-assignment)

;; The procedure list is an alist of the form  
;; ( ... (name proc-definition) ... ). Notice that this will change when  
;; I return to prefix form since the name is not in the car position in that  
;; representation.

Definition:  
predefined-procp (name) = (name ∈ PREDEFINED-PROCEDURE-LIST)

Definition:  
user-defined-procp (name, proc-list)  
= ((name ∉ PREDEFINED-PROCEDURE-LIST) ∧ definedp (name, proc-list))

Definition:  
defined-procp (name, proc-list)  
= (predefined-procp (name) ∨ user-defined-procp (name, proc-list))

Definition:  
fetch-def (name, proc-list) = assoc (name, proc-list)

Definition:  
fetch-called-def (stmt, proc-list) = fetch-def (call-name (stmt), proc-list)

;; (name data-formals cond-formals data-locals cond-locals body)

Definition:  
def-name (def) = car (def)

Event: Disable def-name.

Definition:  
def-formals (def) = cadr (def)

Event: Disable def-formals.

Definition:  
def-conds (def) = caddr (def)
EVENT: Disable def-conds.

DEFINITION: def-locals (def) = cadddr (def)
EVENT: Disable def-locals.

DEFINITION: def-cond-locals (def) = caddddr (def)
EVENT: Disable def-cond-locals.

DEFINITION: def-body (def) = cadddddr (def)
EVENT: Disable def-body.

DEFINITION: array-elemtype (type) = cadr (type)
EVENT: Disable array-elemtype.

DEFINITION: array-length (type) = caddr (type)
EVENT: Disable array-length.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; THIS RECOGNIZER
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DEFINITION:
reserved-names-list = ' (leave normal routine error)

DEFINITION: reserved-word (wd) = (wd ∈ RESERVED-NAMES-LIST)

DEFINITION:
ok-mg-namep (ident)
= (litatom (ident)
   ∧ (45 ⋄ unpack (ident))
   ∧ (¬ reserved-word (ident)))

EVENT: Disable ok-mg-namep.
Notice that the word size is selected to be exactly that of Piton. This eliminates some of the difficulties of the mappings.

**Definition**: \( mg\text{-word-size} = 32 \)

**Event**: Introduce the function symbol \( mg\text{-max-ctrl-stk-size} \) of 0 arguments.

**Axiom**: \( mg\text{-max-ctrl-stk-size-small-naturalp} \)
\[
\begin{align*}
\text{(MG-MAX-CTRL-STK-SIZE} &\in \mathbb{N}) \\
\land &\ (\text{MG-MAX-CTRL-STK-SIZE} < \exp(2, \text{MG-WORD-SIZE}))
\end{align*}
\]

**Event**: Introduce the function symbol \( mg\text{-max-temp-stk-size} \) of 0 arguments.

**Axiom**: \( mg\text{-max-temp-stk-size-numberp} \)
\[
\begin{align*}
\text{(MG-MAX-TEMP-STK-SIZE} &\in \mathbb{N}) \\
\land &\ (\text{MG-MAX-TEMP-STK-SIZE} < \exp(2, \text{MG-WORD-SIZE}))
\end{align*}
\]

**;>; Notice also the I can simply use the Piton function small-integerp rather than define a specific MG notion of what is an acceptable integer.**

**Definition**: \( \text{MAXINT} = (\exp(2, \text{MG-WORD-SIZE} - 1) - 1) \)

**Definition**: \( \text{MININT} = (-\exp(2, \text{MG-WORD-SIZE} - 1)) \)

**;>; An integer literal is of the form (INT-MG n) where n is in the range (MININT..MAXINT).**

**Definition**: \( \text{int-literalp} (exp) \)
\[
\begin{align*}
= \ (\text{('int-mg} = \text{car} (exp)) \\
\land &\ \text{length-plistp} (exp, 2) \\
\land &\ \text{small-integerp} (\text{cadr} (exp), \text{MG-WORD-SIZE}))
\end{align*}
\]

**Definition**: \( \text{boolean-literalp} (exp) \)
\[
\begin{align*}
= \ (\text{('boolean-mg} = \text{car} (exp)) \\
\land &\ \text{length-plistp} (exp, 2) \\
\land &\ (\text{cadr} (exp) \in \text{'(true-mg false-mg)})
\end{align*}
\]
**Definition:**

character-literalp \((exp)\)

\[
= (\textbf{`character-mg} = \text{car} (exp)) \\
\land \text{length-plistp} (exp, 2) \\
\land (\text{cadr} (exp) \in \mathbb{N}) \\
\land (\text{cadr} (exp) \leq 127))
\]

**Event:** Disable int-literalp.

**Event:** Disable boolean-literalp.

**Event:** Disable character-literalp.

**Definition:**

simple-mg-type-refp \((typref)\)

\[
= (\text{simple-mg-type-refp} (\text{array-elemtype} (typref)) \\
\land (\text{array-length} (typref) \neq 0))
\]

**Definition:**

array-mg-type-refp \((typref)\)

\[
= (\text{array-mg-type-refp} (typref) \\
\lor (\text{array-elemtype} (typref) \neq 0))
\]

**Event:** Disable array-mg-type-refp.

**Definition:**

mg-type-refp \((typref)\)

\[
= (\text{simple-mg-type-refp} (typref) \lor \text{array-mg-type-refp} (typref))
\]

**Definition:**

simple-typed-literalp \((lit, type)\)

\[
= \textbf{if} \hspace{0.5em} \text{type} = \textbf{`int-mg} \hspace{0.5em} \textbf{then} \hspace{0.5em} \text{int-literalp} (lit) \\
\text{elseif} \hspace{0.5em} \text{type} = \textbf{`boolean-mg} \hspace{0.5em} \textbf{then} \hspace{0.5em} \text{boolean-literalp} (lit) \\
\text{elseif} \hspace{0.5em} \text{type} = \textbf{`character-mg} \hspace{0.5em} \textbf{then} \hspace{0.5em} \text{character-literalp} (lit) \\
\textbf{else} \hspace{0.5em} \textbf{f endif}
\]

**Definition:**

simple-typed-literal-plistp \((lst, type)\)

\[
= \textbf{if} \hspace{0.5em} \text{lst} \simeq \textbf{nil} \hspace{0.5em} \textbf{then} \hspace{0.5em} \text{lst} = \textbf{nil} \\
\textbf{else} \hspace{0.5em} \text{simple-typed-literalp} (\text{car} (lst), type) \\
\text{simple-typed-literal-plistp} (\text{cdr} (lst), type) \textbf{endif}
\]
DEFINITION:
array-literalp (exp, length, elemtype)
=  (simple-typed-literal-plistp (exp, elemtype)
    \&  (length (exp) = length))

EVENT: Disable array-literalp.

DEFINITION:
ok-mg-array-value (exp, type)
=  array-literalp (exp, array-length (type), array-elemtype (type))

EVENT: Disable ok-mg-array-value.

DEFINITION:
ok-mg-valuep (exp, type)
=  if simple-mg-type-refp (type) then simple-typed-literalp (exp, type)
    elseif array-mg-type-refp (type) then ok-mg-array-value (exp, type)
    else f endif

EVENT: Disable ok-mg-valuep.

;; The recognizer has a structure called the name-alist of the following
;; form:
;; (... (name type) ...)
;; This allows the identification of the types of variables. It should be
;; the case that the values of the variables in the meaning alist correspond
;; to their types on the name-alist.

DEFINITION:  m-type (x) = cadr (x)
DEFINITION:  get-m-type (name, alist) = m-type (assoc (name, alist))

DEFINITION:
has-array-type (name, alist)
=  (car (get-m-type (name, alist)) = ’array-mg)

DEFINITION:
mg-name-alist-elementp (x)
=  (ok-mg-namep (car (x)) \& mg-type-refp (m-type (x)))

DEFINITION:
mg-name-alistp (alist)
=  if alist \simeq nil then alist = nil
    else mg-name-alist-elementp (car (alist))
        \& mg-name-alistp (cdr (alist)) endif
Definition: identifier\( (name) = \text{ok-mg-namep}(name) \)

Definition:
defined-identifierp (name, alist)
= (identifierp (name) ∧ definedp (name, alist))

;;; I’m restricting the if-condition to be a boolean identifier rather
;;; than a boolean expression just so that the compilation will take a
;;; fixed number of steps. That is, I want every expression within the
;;; code to be a reference to a variable. This can obviously be relaxed
;;; but is general since I allow the initialization of locals. Thus,
;;; any literal values can be stored in local variables.

;;; >>> These should probably be removed in favor of one function with a
;;; type argument.

Definition:
boolean-identifierp (name, alist)
= (identifierp (name) ∧ (get-m-type (name, alist) = ’boolean-mg))

Definition:
int-identifierp (name, alist)
= (identifierp (name) ∧ (get-m-type (name, alist) = ’int-mg))

Definition:
character-identifierp (name, alist)
= (identifierp (name) ∧ (get-m-type (name, alist) = ’character-mg))

Definition:
array-identifierp (name, alist)
= (defined-identifierp (name, alist) ∧ has-array-type (name, alist))

Event: Disable boolean-identifierp.

Event: Disable int-identifierp.

Event: Disable character-identifierp.

Event: Disable array-identifierp.

Definition:
simple-identifierp (name, alist)
= (boolean-identifierp (name, alist) \
  ∨ int-identifierp (name, alist) \
  ∨ character-identifierp (name, alist))

**DEFINITION:**

simple-typed-identifierp (ident, type, alist) =

  if type = 'int-mg then int-identifierp (ident, alist) \
  elseif type = 'boolean-mg then boolean-identifierp (ident, alist) \
  elseif type = 'character-mg then character-identifierp (ident, alist) \
  else f endif

**THEOREM:** int-identifierp-simple

int-identifierp (x, alist) → simple-identifierp (x, alist)

**THEOREM:** boolean-identifierp-simple

boolean-identifierp (x, alist) → simple-identifierp (x, alist)

**THEOREM:** character-identifierp-simple

character-identifierp (x, alist) → simple-identifierp (x, alist)

**EVENT:** Disable simple-identifierp.

**EVENT:** Disable simple-typed-identifierp.

**THEOREM:** int-identifierp-implies-definedp

int-identifierp (x, alist) → definedp (x, alist)

**EVENT:** Disable int-identifierp-implies-definedp.

**THEOREM:** boolean-identifierp-implies-definedp

boolean-identifierp (x, alist) → definedp (x, alist)

**EVENT:** Disable boolean-identifierp-implies-definedp.

**THEOREM:** character-identifierp-implies-definedp

character-identifierp (x, alist) → definedp (x, alist)

**EVENT:** Disable character-identifierp-implies-definedp.

**THEOREM:** simple-identifierp-implies-definedp

simple-identifierp (x, alist) → definedp (x, alist)

8
Event: Disable simple-identifierp-implies-definedp.

Theorem: simple-typed-identifierp-implies-definedp
simple-typed-identifierp \( (x, \text{type}, \text{alist}) \) → definedp \( (x, \text{alist}) \)

Event: Disable simple-typed-identifierp-implies-definedp.

Theorem: array-identifierp-implies-definedp
array-identifierp \( (x, \text{alist}) \) → definedp \( (x, \text{alist}) \)

Event: Disable array-identifierp-implies-definedp.

;; I have decided to make references to individual array elements impossible.
;; That is, a user program can only pass a whole array to a subroutine. Arrays
;; are viewed as abstract data types and the only accesses to them are via
;; the predefined functions which select elements and write elements. This will
;; guarantee that the param list of a routine is unchanged from the non-structured
;; situation.

Definition:
identifier-plistp \( (\text{lst}) \)
= \( \text{if } \text{lst} \simeq \text{nil} \text{ then } \text{lst} = \text{nil} \)
\quad \text{else} \hspace{1em} \text{identifierp} (\text{car} (\text{lst})) \land \text{identifier-plistp} (\text{cdr} (\text{lst})) \text{ endif}

Theorem: identifier-plistp-plistp
identifier-plistp \( (x) \) → plistp \( (x) \)

Event: Disable identifier-plistp-plistp.

Theorem: identifier-plistp-distributes
(identifier-plistp \( (x) \land \text{identifier-plistp} (y) \))
→ identifier-plistp (append \( (x, y) \))

Theorem: leave-not-in-identifier-plistp
identifier-plistp \( (\text{lst}) \) → (‘leave \( \notin \text{lst} \))


Definition:
cond-identifierp \( (x, \text{cond-list}) \)
= \( \{(x = \text{'routineerror}) \lor (\text{identifierp} (x) \land (x \in \text{cond-list}))\} \)
EVENT: Disable cond-identifier-p.

**Definition:**

\[
\text{cond-identifier-plistp} (\text{lst}, \text{cond-list}) = \begin{cases} 
\text{nil} & \text{if } \text{lst} \simeq \text{nil} \text{ then } \text{lst} = \text{nil} \\
\text{false} & \text{else} \text{ cond-identifierp} (\text{car} (\text{lst}), \text{cond-list}) \\
\wedge \text{ cond-identifier-plistp} (\text{cdr} (\text{lst}), \text{cond-list}) \end{cases}
\]

EVENT: Disable cond-identifier-plistp.

**Theorem:** \(\text{cond-identifier-plistp-cond-subsetp}\)

\[
\text{cond-identifier-plistp} (\text{lst}, \text{cond-list}) \rightarrow \text{cond-subsetp} (\text{lst}, \text{cond-list})
\]

EVENT: Disable cond-identifier-plistp-cond-subsetp.

**Theorem:** \(\text{normal-not-in-cond-identifier-plistp}\)

\[
\text{cond-identifier-plistp} (\text{lst}, \text{cond-list}) \rightarrow (\text{normal} \not\in \text{lst})
\]


**Theorem:** \(\text{adding-element-preserves-cond-identifier-plistp}\)

\[
\text{cond-identifier-plistp} (\text{y}, \text{cond-list}) \rightarrow \text{cond-identifier-plistp} (\text{y}, \text{cons} (\text{x}, \text{cond-list}))
\]

EVENT: Disable adding-element-preserves-cond-identifier-plistp.

**Theorem:** \(\text{superset-preserves-cond-identifier-plistp}\)

\[
(\text{subset} (\text{x}, \text{y}) \wedge \text{cond-identifier-plistp} (\text{lst}, \text{x})) \rightarrow \text{cond-identifier-plistp} (\text{lst}, \text{y})
\]

EVENT: Disable superset-preserves-cond-identifier-plistp.

**Definition:**

\[
\text{nonempty-cond-identifier-plistp} (\text{lst}, \text{cond-list}) = (\text{cond-identifier-plistp} (\text{lst}, \text{cond-list}) \wedge (\text{lst} \not\simeq \text{nil}))
\]

**Definition:**

\[
\text{ok-condition} (\text{exp}, \text{cond-list}) = ((\text{exp} = \text{routineerror}) \vee ((\text{ok-mg-namep} (\text{exp}) \vee (\text{exp} = \text{leave})) \wedge (\text{exp} \in \text{cond-list})))
\]
Theorem: ok-condition-litatom
ok-condition (exp, cond-list) → litatom (exp)

Event: Disable ok-condition-litatom.

;; Notice that I’m not allowing references to individual array elements, so
;; this doesn’t change from the previous version.

Definition:
ok-actual-params-list (lst, alist) =
  if lst ≃ nil then lst = nil
  else defined-identifierp (car (lst), alist)
    ∧ ok-actual-params-list (cdr (lst), alist) endif

;; The formal is of the form (name kind typeref); kind is in {var-mg const-mg}.

;; There is a match between actual and formal if the actual is a literal of the type of the
;; formal or is an identifier of exactly the same type as the formal.
;; This will have to be relaxed for subtypes when I add them.

Definition:
ok-identifier-actual (actual, formal, alist) =
  (identifierp (actual)
   ∧ (get-m-type (actual, alist) = formal-type (formal)))

Definition:
data-params-match (actual, formal, alist) =
  ok-identifier-actual (actual, formal, alist)

Definition:
data-param-lists-match (actuals, formals, alist) =
  if (actuals ≃ nil) ∨ (formals ≃ nil)
  then (formals = nil) ∧ (actuals = nil)
  else data-params-match (car (actuals), car (formals), alist)
    ∧ data-param-lists-match (cdr (actuals),
                              cdr (formals),
                              alist) endif

Theorem: data-param-lists-match-in-length
  data-param-lists-match (actuals, formals, alist)
  → (length (formals) = length (actuals))

DEFINITION:
cond-params-match (cond-actuals, conds)
= (length (cond-actuals) = length (conds))

THEOREM: list-count-decreases
((x = car (stmt)) \& (x \neq 0))
→ (((count (cadr (stmt)) < count (stmt)) = t)
∧ ((count (caddr (stmt)) < count (stmt)) = t)
∧ ((count (cadddr (stmt)) < count (stmt)) = t)
∧ ((count (caddddr (stmt)) < count (stmt)) = t))

;; Notice that I define the semantics, etc. of each of the predefined routines
;; individually. This allows me to loosen the restrictions on user-defined
;; procedures. It also, allows me to dispense with much of the overhead of
;; user defined routines. The only condition that any of the predefined's can
;; return is 'routineerror. Consequently, I can do away with the cond translation
;; mechanism required of user-defined routines. These simply set 'routineerror
;; themselves. Also, I can eliminate the aliasing requirement for these since
;; I guarantee by coding the aliasing does not cause any problem.

;; This approach does seem to add an additional burden with respect to the
;; amount of things which must be proved. However, since the predefined's
;; are not defined recursively, this is not particularly bad.

;; THE 'GENERIC' OPERATIONS
;; The operations of assignment and EQ work on any of the simple types. For
;; each of them, we allow the variant where one of the args is a literal
;; of the appropriate type. This is not strictly necessary except insofar
;; as it makes the subset more realistic and usable.

;; (mg-simple-variable-assignment x y)
;; x := y where source must be a variable of the same type as the destination.

DEFINITION:
ok-mg-simple-variable-assignment-args (args, alist)
= (length-plistp (args, 2)
∧ simple-identifierp (car (args), alist)
∧ simple-identifierp (cadr (args), alist)
\[ (\text{cadr (assoc (car (args), alist))}) = \text{cadr (assoc (cadr (args), alist)))} \]

;; (mg-simple-constant-assignment x c)
;; x := c where c must be a literal of the same type as the destination

**Definition:**
ok-mg-simple-constant-assignment-args (args, alist) = (length-plistp (args, 2) \(\land\) simple-identifierp (car (args), alist) \(\land\) simple-typed-literalp (cadr (args), cadr (assoc (car (args), alist))))

;; (mg-simple-variable-eq b x y)
;; b := (x = y) where both x and y are variables of the same type.

**Definition:**
ok-mg-simple-variable-eq-args (args, alist) = (length-plistp (args, 3) \(\land\) boolean-identifierp (car (args), alist) \(\land\) simple-identifierp (cadr (args), alist) \(\land\) simple-identifierp (caddr (args), alist) \(\land\) (cadr (assoc (cadr (args), alist)) = cadr (assoc (caddr (args), alist))))

;; (mg-simple-constant-eq b x c)
;; b := (x = c) where x is a variable and c a literal of compatible type.

**Definition:**
ok-mg-simple-constant-eq-args (args, alist) = (length-plistp (args, 3) \(\land\) boolean-identifierp (car (args), alist) \(\land\) simple-identifierp (cadr (args), alist) \(\land\) simple-identifierp (caddr (args), alist) \(\land\) simple-typed-literalp (caddr (args), cadr (assoc (cadr (args), alist))))

;; THE INTEGER OPERATIONS

;; (mg-integer-le b x y)
;; b := (x le y) where both x and y are integer variables

**Definition:**
ok-mg-integer-le-args (args, alist) = (length-plistp (args, 3)  
  ∧ boolean-identifierp (car (args), alist)  
  ∧ int-identifierp (cadr (args), alist)  
  ∧ int-identifierp (caddr (args), alist))

;; (mg-integer-unary-minus (args alist))
;; z := -x

**DEFINITION:**

ok-mg-integer-unary-minus-args (args, alist) = (length-plistp (args, 2)  
  ∧ int-identifierp (car (args), alist)  
  ∧ int-identifierp (cadr (args), alist))

;; (mg-integer-add (x y z))
;; x := y + z >> Notice that this is a change from the previous version.

**DEFINITION:**

ok-mg-integer-add-args (args, alist) = (length-plistp (args, 3)  
  ∧ int-identifierp (car (args), alist)  
  ∧ int-identifierp (cadr (args), alist)  
  ∧ int-identifierp (caddr (args), alist))

;; (mg-integer-subtract (x y z))
;; x := y - z

**DEFINITION:**

ok-mg-integer-subtract-args (args, alist) = (length-plistp (args, 3)  
  ∧ int-identifierp (car (args), alist)  
  ∧ int-identifierp (cadr (args), alist)  
  ∧ int-identifierp (caddr (args), alist))

;; BOOLEAN OPERATIONS

;; (mg-boolean-or (b c d))
;; b := c or d -- both disjuncts must be boolean identifiers

**DEFINITION:**
ok-mg-boolean-or-args (args, alist) = (length-plistp (args, 3) ∧ boolean-identifierp (car (args), alist) ∧ boolean-identifierp (cadr (args), alist) ∧ boolean-identifierp (caddr (args), alist))

;; (mg-boolean-and (b c d))
;; b := c and d

Definition:
ok-mg-boolean-and-args (args, alist) = (length-plistp (args, 3) ∧ boolean-identifierp (car (args), alist) ∧ boolean-identifierp (cadr (args), alist) ∧ boolean-identifierp (caddr (args), alist))

;; (mg-boolean-not (b c))
;; b := not c

Definition:
ok-mg-boolean-not-args (args, alist) = (length-plistp (args, 2) ∧ boolean-identifierp (car (args), alist) ∧ boolean-identifierp (cadr (args), alist))

;; ARRAY OPERATIONS

;; (mg-index-array (z A i size)) >> Notice the change in the order of the args
;; z := A[i]
;; It is necessary for that last to be passed to do bounds-checking. That information
;; is not available to the translator otherwise. It is a special argument which is a
;; numberp rather than an MG literal. It is expected that the preprocessor
;; will actually supply this argument, not
;; the mg programmer. Thus the prefix form might have args (z A i 24) where 24 is the
;; size of A.

Definition:
ok-mg-index-array-args (args, alist) = (length-plistp (args, 4) ∧ array-identifierp (cadr (args), alist) ∧ int-identifierp (caddr (args), alist) ∧ simple-typed-identifierp (car (args),
(mg-array-element-assignment A i value size)
;; A[i] := value
;; Here i must be an integer variable and value a variable of
;; the appropriate element-type.

**Definition:**

ok-mg-array-element-assignment-args (args, alist) =
(length-plistp (args, 4)
∧ array-identifierp (car (args), alist)
∧ int-identifierp (cadr (args), alist)
∧ (caddr (args) = array-length (cadr (assoc (car (args), alist))))
∧ (caddr (args) < maxint)
∧ simple-typed-identifierp (caddr (args),
array-elmentype (cadr (assoc (cadr (args),
alist)))))

**Definition:**

ok-predefined-proc-args (name, args, alist) =
  case on name:
  case = mg-simple-variable-assignment
  then ok-mg-simple-variable-assignment-args (args, alist)
  case = mg-simple-constant-assignment
  then ok-mg-simple-constant-assignment-args (args, alist)
  case = mg-simple-variable-eq
  then ok-mg-simple-variable-eq-args (args, alist)
  case = mg-simple-constant-eq
  then ok-mg-simple-constant-eq-args (args, alist)
  case = mg-integer-le
  then ok-mg-integer-le-args (args, alist)
  case = mg-integer-unary-minus
  then ok-mg-integer-unary-minus-args (args, alist)
  case = mg-integer-add
  then ok-mg-integer-add-args (args, alist)
  case = mg-integer-subtract
  then ok-mg-integer-subtract-args (args, alist)
  case = mg-boolean-or
  then ok-mg-boolean-or-args (args, alist)
case = mg-boolean-and
then ok-mg-boolean-and-args (args, alist)
case = mg-boolean-not
then ok-mg-boolean-not-args (args, alist)
case = mg-index-array
then ok-mg-index-array-args (args, alist)
case = mg-array-element-assignment
then ok-mg-array-element-assignment-args (args, alist)
otherwise f endcase

EVENT: Disable ok-predefined-proc-args.

;; A predefined proc-call is of the form (predefined-proc-call-mg name actuals)
;; where name is one of the legal predefineds and the actuals are legitimate arguments
;; for that predefined procedure according to the definitions above.

DEFINITION:
ok-predefined-proc-call (stmt, alist)
= (length-plistp (stmt, 3)
   ∧ predefined-procp (call-name (stmt))
   ∧ ok-predefined-proc-args (call-name (stmt),
                             call-actuals (stmt),
                             alist))

EVENT: Disable ok-predefined-proc-call.

DEFINITION:
ok-proc-call (stmt, r-cond-list, alist, proc-list)
= (length-plistp (stmt, 4)
   ∧ identifierp (call-name (stmt))
   ∧ user-defined-procp (call-name (stmt), proc-list)
   ∧ ok-actual-params-list (call-actuals (stmt), alist)
   ∧ no-duplicates (call-actuals (stmt))
   ∧ data-param-lists-match (call-actuals (stmt),
                               def-formals (fetch-called-def (stmt,
                                                             proc-list)),
                               alist)
   ∧ cond-identifier-plistp (call-conds (stmt), r-cond-list)
   ∧ cond-params-match (call-conds (stmt),
                        def-conds (fetch-called-def (stmt, proc-list))))

EVENT: Disable ok-proc-call.
DEFINITION:

ok-mg-statement (stmt, r-cond-list, alist, proc-list) =
  case on car(stmt):
    case = no-op-mg
    then cdr(stmt) = nil
    case = signal-mg
    then length-plistp(stmt, 2)
    ∧ ok-condition(signalled-condition(stmt), r-cond-list)
    case = prog2-mg
    then length-plistp(stmt, 3)
    ∧ ok-mg-statement(prog2-left-branch(stmt),
                       r-cond-list,
                       alist,
                       proc-list)
    ∧ ok-mg-statement(prog2-right-branch(stmt),
                       r-cond-list,
                       alist,
                       proc-list)
    case = loop-mg
    then length-plistp(stmt, 2)
    ∧ ok-mg-statement(loop-body(stmt),
                       cons(’leave, r-cond-list),
                       alist,
                       proc-list)
    case = if-mg
    then length-plistp(stmt, 4)
    ∧ boolean-identifierp(if-condition(stmt), alist)
    ∧ ok-mg-statement(if-true-branch(stmt),
                       r-cond-list,
                       alist,
                       proc-list)
    ∧ ok-mg-statement(if-false-branch(stmt),
                       r-cond-list,
                       alist,
                       proc-list)
    case = begin-mg
    then length-plistp(stmt, 4)
    ∧ ok-mg-statement(begin-body(stmt),
                       append(when-labels(stmt), r-cond-list),
                       alist,
                       proc-list)
    ∧ nonempty-cond-identifier-plistp(when-labels(stmt),
                                      r-cond-list)
    ∧ ok-mg-statement(when-handler(stmt),
                       append(when-labels(stmt), r-cond-list),
                       alist,
                       proc-list)
\[
\text{\texttt{case}} = \text{\texttt{proc-call-mg}} \\
\text{\texttt{then}} \ \text{\texttt{ok-proc-call}} (\texttt{stmt}, \texttt{r-cond-list}, \texttt{alist}, \texttt{proc-list}) \\
\text{\texttt{case}} = \text{\texttt{predefined-proc-call-mg}} \\
\text{\texttt{then}} \ \text{\texttt{ok-predefined-proc-call}} (\texttt{stmt}, \texttt{alist}) \\
\text{\texttt{otherwise}} \ \text{\texttt{f endcase}}
\]

EVENT: Disable signalled-condition.

EVENT: Disable prog2-left-branch.

EVENT: Disable prog2-right-branch.

EVENT: Disable loop-body.

EVENT: Disable if-condition.

EVENT: Disable if-true-branch.

EVENT: Disable if-false-branch.

EVENT: Disable begin-body.

EVENT: Disable when-handler.

EVENT: Disable when-labels.

EVENT: Disable call-name.

EVENT: Disable call-actuals.

EVENT: Disable call-conds.

THEOREM: ok-signal-expansion
\[
\text{\texttt{ok-mg-statement}} (\texttt{cons (\texttt{'signal-mg, args}), r-cond-list, alist, proc-list})
\]
Theorem: ok-prog2-statement
\[
((\text{car}(\text{stmt}) = \text{'prog2-mg})
\land
\text{ok-mg-statement}(\text{stmt}, \text{r-cond-list}, \text{alist}, \text{proc-list}))
\rightarrow
(\text{ok-mg-statement}(\text{prog2-left-branch(\text{stmt)}), \text{r-cond-list}, \text{alist}, \text{proc-list}))
\land
(\text{ok-mg-statement}(\text{prog2-right-branch(\text{stmt)}), \text{r-cond-list}, \text{alist}, \text{proc-list}))
\]

Theorem: ok-loop-statement
\[
((\text{car}(\text{stmt}) = \text{'loop-mg})
\land
\text{ok-mg-statement}(\text{stmt}, \text{r-cond-list}, \text{alist}, \text{proc-list}))
\rightarrow
\text{ok-mg-statement}(\text{loop-body(\text{stmt)}), \text{cons('leave, \text{r-cond-list)}, \text{alist, \text{proc-list)}})
\]

Theorem: ok-if-statement
\[
((\text{car}(\text{stmt}) = \text{'if-mg})
\land
\text{ok-mg-statement}(\text{stmt}, \text{r-cond-list}, \text{alist}, \text{proc-list}))
\rightarrow
(\text{ok-mg-statement}(\text{if-true-branch(\text{stmt)}), \text{r-cond-list}, \text{alist}, \text{proc-list}))
\land
(\text{ok-mg-statement}(\text{if-false-branch(\text{stmt)}), \text{r-cond-list}, \text{alist}, \text{proc-list}))
\]

Theorem: ok-begin-expansion
\[
\text{ok-mg-statement}(\text{cons('begin-mg, args)}, \text{r-cond-list}, \text{alist}, \text{proc-list})
= (\text{length-plistp}(\text{cons('begin-mg, args)}, 4)
\land
\text{ok-mg-statement}(\text{begin-body(\text{cons('begin-mg, args))}, \text{append\text{(when-labels(\text{cons('begin-mg, args)}), \text{r-cond-list)}}, \text{alist, \text{proc-list}}))
\land
\text{nonempty-cond-identifier-plistp}(\text{when-labels(\text{cons('begin-mg, args)}), \text{r-cond-list}})
\land
\text{ok-mg-statement}(\text{when-handler(\text{cons('begin-mg, args)}), \text{r-cond-list}, \text{alist, \text{proc-list}}))
\]
THEOREM: ok-begin-statement
\[ ((\text{car} (\text{stmt}) = \texttt{begin-mg}) \land \text{ok-mg-statement} (\text{stmt}, \text{r-cond-list}, \text{alist}, \text{proc-list})) \rightarrow (\text{ok-mg-statement} (\text{begin-body} (\text{stmt}), \text{append} (\text{when-labels} (\text{stmt}), \text{r-cond-list}), \text{alist}, \text{proc-list}) \land \text{ok-mg-statement} (\text{when-handler} (\text{stmt}), \text{r-cond-list}, \text{alist}, \text{proc-list}))) \]

THEOREM: ok-proc-call-expansion
\[ \text{ok-mg-statement} (\text{cons} (\texttt{proc-call-mg}, \text{args}), \text{r-cond-list}, \text{alist}, \text{proc-list}) = \text{ok-proc-call} (\text{cons} (\texttt{proc-call-mg}, \text{args}), \text{r-cond-list}, \text{alist}, \text{proc-list}) \]

EVENT: Disable ok-mg-statement.

THEOREM: signalled-condition-not-normal
\[ ((\texttt{signal-mg} = \text{car} (\text{stmt})) \land \text{ok-mg-statement} (\text{stmt}, \text{r-cond-list}, \text{alist}, \text{proc-list})) \rightarrow (\text{signalled-condition} (\text{stmt}) \neq \texttt{normal}) \]

EVENT: Disable signalled-condition-not-normal.

;; member of the data formal list is of the form
;; (name typeref)

DEFINITION:
ok-mg-formal-data-param (exp)
\[ = (\text{length-plistp} (exp, 2) \land \text{ok-mg-namep} (\text{car} (exp)) \land \text{mg-type-refp} (\text{formal-type} (exp))) \]

EVENT: Disable ok-mg-formal-data-param.

DEFINITION:
ok-mg-formal-data-params-plistp (lst)
\[ = \text{if } lst \simeq \texttt{nil} \text{ then } lst = \texttt{nil} \text{ else } \text{ok-mg-formal-data-param} (\text{car} (lst)) \land \text{ok-mg-formal-data-params-plistp} (\text{cdr} (lst)) \text{ endif} \]
A local decl is of the form (name typedef initial-value).
Notice that this obviates the need to compute initial values.

**Definition:**
ok-mg-local-data-decl (exp)
= (length-plistp (exp, 3)
  ∧ ok-mg-namep (car (exp))
  ∧ mg-type-refp (formal-type (exp))
  ∧ ok-mg-valuep (formal-initial-value (exp), formal-type (exp)))

**Definition:**
ok-mg-local-data-plistp (lst)
= if lst ≃ nil then lst = nil
else ok-mg-local-data-decl (car (lst))
  ∧ ok-mg-local-data-plistp (cdr (lst)) endif

The legal conditions which can signalled are those which appear in the
the formal or in the local conds.

**Definition:**
make-cond-list (def) = append (def-conds (def), def-cond-locals (def))

This takes a list of formal data params or local var decls and
makes a name-alist.

>>> This is a useless definition. It does nothing

**Definition:**
make-alist-from-formals (lst)
= if lst ≃ nil then nil
else cons (list (name (car (lst)), formal-type (car (lst))),
  make-alist-from-formals (cdr (lst))) endif

This takes the formals and locals and makes a name-alist.

>>> Why not just concatenate them.

**Definition:**
make-name-alist (def)
= append (make-alist-from-formals (def-formals (def)),
  make-alist-from-formals (def-locals (def)))
Given a list of formal-data-params or local-data-decls this lists off the names. This is necessary to check that all local names are unique.

**Definition:**

\[
\text{collect-local-names (def)} = \text{append (listcars (def-formals (def))), listcars (def-locals (def)))}
\]

**Definition:**

\[
\text{ok-mg-def (def, proc-list)} = \left(\text{length-plistp (def, 6)} \land \text{ok-mg-namep (def-name (def))} \land \text{ok-mg-formal-data-params-plistp (def-formals (def))} \land \text{identifier-plistp (def-conds (def))} \land \text{ok-mg-local-data-plistp (def-locals (def))} \land \text{identifier-plistp (def-cond-locals (def))} \land \text{no-duplicates (collect-local-names (def))} \land \left(\left(\text{length (def-conds (def))} + \text{length (def-cond-locals (def))}\right) < \left(\left(\text{exp (2, MG-WORD-SIZE)} - 1\right) - 1\right)\right) \land \text{ok-mg-statement (def-body (def)), make-cond-list (def), make-name-alist (def), proc-list}\right)
\]

**Theorem:** make-cond-list-legal-length

\[
\text{ok-mg-def (def, proc-list)} \rightarrow \left(\left(\text{length (make-cond-list (def))} < \left(\left(\text{exp (2, MG-WORD-SIZE)} - 1\right) - 1\right)\right) = t\right)
\]

**Definition:**

\[
\text{ok-mg-def-plistp1 (lst1, lst2)} = \begin{cases} 
\text{nil} & \text{if } \text{lst1} = \text{nil} \\
\text{ok-mg-def (car (lst1), lst2)} \land \text{ok-mg-def-plistp1 (cdr (lst1), lst2)} & \text{else}
\end{cases}
\]

**Definition:**

\[
\text{ok-mg-def-plistp (proc-list)} = \text{ok-mg-def-plistp1 (proc-list, proc-list)}
\]

**Event:** Disable ok-mg-def.
**Theorem: assoc-def-ok1**

\[(\text{ok-mg-def-plistp} (\text{proc-list1}, \text{proc-list2}) \land \text{definedp} (\text{name}, \text{proc-list1})) \rightarrow \text{ok-mg-def} (\text{assoc} (\text{name}, \text{proc-list1}), \text{proc-list2})\]

**Event:** Disable assoc-def-ok1.

**Theorem: called-def-ok**

\[((\text{‘proc-call-mg = car (stmt)})
\land \text{ok-mg-statement} (\text{stmt}, \text{r-cond-list}, \text{name-alist}, \text{proc-list})
\land \text{ok-mg-def-plistp} (\text{proc-list}))
\rightarrow \text{ok-mg-def} (\text{fetch-called-def} (\text{stmt}, \text{proc-list}), \text{proc-list})\]

**Event:** Disable fetch-called-def.

**Theorem: called-def-formals-ok**

\[((\text{‘proc-call-mg = car (stmt)})
\land \text{ok-mg-statement} (\text{stmt}, \text{r-cond-list}, \text{name-alist}, \text{proc-list})
\land \text{ok-mg-def-plistp} (\text{proc-list}))
\rightarrow (\text{ok-mg-formal-data-params-plistp} (\text{def-formals} (\text{fetch-called-def} (\text{stmt}, \text{proc-list}))))
\land \text{ok-mg-local-data-plistp} (\text{def-locals} (\text{fetch-called-def} (\text{stmt}, \text{proc-list})))
\land \text{data-param-lists-match} (\text{call-actuals} (\text{stmt}),
\quad \text{def-formals} (\text{fetch-called-def} (\text{stmt}, \text{proc-list})),
\quad \text{name-alist})
\land \text{no-duplicates} (\text{listcars} (\text{def-formals} (\text{fetch-called-def} (\text{stmt}, \text{proc-list}))))
\land \text{plistp} (\text{make-cond-list} (\text{fetch-called-def} (\text{stmt}, \text{proc-list}))))
\land \text{no-duplicates} (\text{append} (\text{listcars} (\text{def-formals} (\text{fetch-called-def} (\text{stmt}, \text{proc-list}))),
\quad \text{listcars} (\text{def-locals} (\text{fetch-called-def} (\text{stmt}, \text{proc-list}))))))
\land \text{all-cars-unique} (\text{def-formals} (\text{fetch-called-def} (\text{stmt}, \text{proc-list})))
\land \text{all-cars-unique} (\text{def-locals} (\text{fetch-called-def} (\text{stmt}, \text{proc-list}))))\]

**Theorem: ok-locals-plistp**

\[\text{ok-mg-local-data-plistp} (x) \rightarrow \text{plistp} (x)\]

**Event:** Disable ok-locals-plistp.

**Theorem: locals-plistp**

\[((\text{‘proc-call-mg = car (stmt)})\]

24
∧ ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)
∧ ok-mg-def-plistp (proc-list))
→ plistp (def-locals (fetch-called-def (stmt, proc-list)))

EVENT: Disable mg-name-alist-elementp.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; MG INTERPRETER FUNCTIONS
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The following set of functions defines the interpreter for Micro-Gypsy.
;; A state is defined to be an ordered pair consisting of a variable a-list and a
;; global condition indicator.
;; The meaning of a statement in an environment (state) is the environment
;; which results from executing that statement. Thus, the semantics is very
;; much an operational semantics.

;; The condition component of the state is simply a litatom which is 'normal
;; in the initial state. The variable a-list is of the form
;; ((v1 (value1 type1)) ... (vn (valuen typen))) for each of the entities known in
;; the current scope. Notice that the recognizer alist is not required though I
;; will need to record on the var-alist whether a variable is a const or var
;; param when I add procedure calls.

EVENT: Add the shell mg-state, with recognizer function symbol mg-statep and
3 accessors: cc, with type restriction (none-of) and default value false; mg-
alist, with type restriction (none-of) and default value false; mg-psw, with type
restriction (none-of) and default value false.

DEFINITION:
resource-errorp (mg-state) = (mg-psw (mg-state) ≠ 'run)

DEFINITION:
signal-system-error (mg-state, error)
   = mg-state (cc (mg-state), mg-alist (mg-state), error)

DEFINITION: normal (mg-state) = (cc (mg-state) = 'normal)

DEFINITION: m-value (x) = caddr (x)

DEFINITION:
get-m-value (name, alist) = m-value (assoc (name, alist))
**Definition:**

\[
\text{mg-alist-elementp} (x) = (\text{length-plistp} (x, 3) \land \text{ok-mg-namep} (\text{car} (x)) \land \text{mg-type-refp} (\text{m-type} (x)) \land \text{ok-mg-valuep} (\text{m-value} (x), \text{m-type} (x)))
\]

**Theorem:** new-value-mg-alist-elementp

\[
(\text{mg-alist-elementp} (x) \land \text{ok-mg-valuep} (\text{value}, \text{cadr} (x))) \rightarrow \text{mg-alist-elementp} (\text{cons} (\text{car} (x), \text{cons} (\text{cadr} (x), \text{cons} (\text{value}, \text{cddr} (x)))))
\]

**Event:** Disable mg-alist-elementp.

**Definition:**

\[
\text{mg-alistp} (\text{lst}) = \begin{cases} \text{nil} & \text{if } \text{lst} = \text{nil} \\ \text{mg-alist-elementp} (\text{car} (\text{lst})) \land \text{mg-alistp} (\text{cdr} (\text{lst})) & \text{else} \end{cases}
\]

**Theorem:** mg-alistp-cdr

\[
(\text{listp} (x) \land \text{mg-alistp} (x)) \rightarrow \text{mg-alistp} (\text{cdr} (x))
\]

**Theorem:** mg-alistp-cons

\[
\text{mg-alistp} (\text{cons} (x, \text{cons} (y, z))) \rightarrow \text{mg-alistp} (\text{cons} (x, z))
\]

**Theorem:** mg-alist-member-mg-alist-elementp

\[
(\text{mg-alistp} (\text{mg-alist}) \land (x \in \text{mg-alist})) \rightarrow \text{mg-alist-elementp} (x)
\]

**Theorem:** mg-alistp-distributes

\[
\text{mg-alistp} (\text{append} (x, y)) \rightarrow \text{mg-alistp} (y)
\]

**Theorem:** mg-alistp-distributes2

\[
(\text{mg-alistp} (\text{append} (x, y)) \land \text{plistp} (x)) \rightarrow \text{mg-alistp} (x)
\]

**Event:** Disable mg-alistp-distributes2.

**Theorem:** mg-alist-mg-name-alistp

\[
\text{mg-alistp} (\text{lst}) \rightarrow \text{mg-name-alistp} (\text{lst})
\]

**Theorem:** mg-alistp-plistp

\[
\text{mg-alistp} (\text{alist}) \rightarrow \text{plistp} (\text{alist})
\]

**Event:** Disable mg-alistp-plistp.

**Theorem:** mg-alist-elements-have-ok-values

\[
(\text{mg-alistp} (\text{alist}) \land \text{definedp} (x, \text{alist})) \rightarrow \text{ok-mg-valuep} (\text{caddr} (\text{assoc} (x, \text{alist})), \text{cadr} (\text{assoc} (x, \text{alist})))
\]

26
**Theorem:** restrict-preserves-mg-alistp
\[
\text{mg-alistp}\ (\text{alist}) \rightarrow \text{mg-alistp}\ (\text{restrict}\ (\text{alist, names}))
\]

**Definition:**
\[
\text{ok-cc}\ (c, \text{cond-list}) = (\text{litatom}\ (c) \\wedge \ ((c \in \text{'(normal routine error)}) \vee (c \in \text{cond-list})))
\]

**Theorem:** mg-alistp-implies-mg-statep
\[
\text{mg-alistp}\ (\text{mg-alist}\ (\text{mg-state})) \rightarrow \text{mg-statep}\ (\text{mg-state})
\]

**Definition:**
\[
\text{ok-mg-statep}\ (\text{mg-state}, \text{cond-list}) = (\text{ok-cc}\ (\text{cc}\ (\text{mg-state}), \text{cond-list}) \wedge \text{mg-alistp}\ (\text{mg-alist}\ (\text{mg-state})))
\]

**Theorem:** ok-mg-statep-alist-plistp
\[
\text{ok-mg-statep}\ (\text{mg-state}, \text{cond-list}) \rightarrow \text{plistp}\ (\text{mg-alist}\ (\text{mg-state}))
\]

**Theorem:** cons-preserves-ok-mg-statep
\[
\text{ok-mg-statep}\ (\text{mg-state}, \text{cond-list}) \rightarrow \text{ok-mg-statep}\ (\text{mg-state}, \text{cons}\ (x, \text{cond-list}))
\]

**Event:** Disable cons-preserves-ok-mg-statep.

**Definition:**
\[
\text{set-condition}\ (\text{mg-state}, \text{condition-name}) = \text{mg-state}\ (\text{condition-name}, \text{mg-alist}\ (\text{mg-state}), \text{mg-psw}\ (\text{mg-state}))
\]

**Theorem:** cc-set-condition
\[
\text{cc}\ (\text{set-condition}\ (\text{mg-state}, \text{cond})) = \text{cond}
\]

**Theorem:** mg-alist-set-condition
\[
\text{mg-alist}\ (\text{set-condition}\ (\text{mg-state}, \text{cond})) = \text{mg-alist}\ (\text{mg-state})
\]

**Theorem:** ok-mg-statep-mg-alist-mg-alistp
\[
\text{ok-mg-statep}\ (\text{mg-state}, \text{r-cond-list}) \rightarrow \text{mg-alistp}\ (\text{mg-alist}\ (\text{mg-state}))
\]

**Definition:**
\[
\text{remove-leave}\ (\text{mg-state}) = \text{if}\ \text{cc}\ (\text{mg-state}) = \text{leave}\ \text{then}\ \text{set-condition}\ (\text{mg-state}, \text{normal})\ \text{endif}
\]

**Definition:**
\[
\text{mg-expression-falsep}\ (\text{exp, mg-state}) = (\text{get-m-value}\ (\text{exp}, \text{mg-alist}\ (\text{mg-state})) = \text{(boolean-mg false-mg)})
\]
\[\text{Definition: } \\text{convert-condition1} (\text{cond}, \text{formals}, \text{actuals}) =
\begin{align*}
\text{if formals } &\approx \text{nil} \text{ then } \text{routineerror} \\
\text{elseif } \text{cond} &\text{ = car(formals) then car(actuals)} \\
\text{else } &\text{convert-condition1} (\text{cond}, \text{cdr(formals)}, \text{cdr(actuals)}) \text{ endif}
\end{align*}
\]

\[\text{Definition: } \text{convert-condition} (\text{cond}, \text{formals}, \text{actuals}) =
\begin{align*}
\text{if } \text{cond} &\in \langle \text{normal routineerror} \rangle \text{ then } \text{cond} \\
\text{else } &\text{convert-condition1} (\text{cond}, \text{formals}, \text{actuals}) \text{ endif}
\end{align*}
\]

\[\text{Theorem: convert-condition-non-member} \\
\text{(cond } \notin \text{ formals) } \rightarrow \text{ (convert-condition1} (\text{cond}, \text{formals}, \text{actuals}) = \text{ routineerror})
\]

\[\text{Definition: } \text{set-alist-value} (\text{name, val, alist}) =
\begin{align*}
\text{if } \text{alist } &\approx \text{nil} \text{ then } \text{nil} \\
\text{elseif } \text{car(car(alist)) = name} &\text{ then cons} (\text{cons(name,}} \\
&\text{ cons(m-type(car(alist)), cons(val, cdddr(car(alist))))}}, \\
&\text{cdr(alist))} \\
\text{else } &\text{cons(car(alist), set-alist-value(name, val, cdr(alist)))} \text{ endif}
\end{align*}
\]

\[\text{Theorem: set-alist-value-preserves-definedp} \\
\text{definedp(v, alist) } \rightarrow \text{ definedp(v, set-alist-value(x, y, alist))}
\]

\[\text{Event: Disable set-alist-value-preserves-definedp.}
\]

\[\text{Theorem: set-alist-value-preserves-ok-actual-params-list} \\
\text{ok-actual-params-list} (\text{actuals, alist}) \rightarrow \text{ok-actual-params-list} (\text{actuals, set-alist-value(x, y, alist)})
\]

\[\text{Event: Disable set-alist-value-preserves-ok-actual-params-list.}
\]

\[\text{Theorem: set-alist-value-preserves-cadr-assoc} \\
\text{mg-alistp(alist)} \rightarrow \text{ (cadr(assoc(v, set-alist-value(x, y, alist))))} = \text{ cadr(assoc(v, alist))}
\]

THEOREM: set-alist-value-preserves-data-param-lists-match
\[(\text{mg-alistp (alist)} \land \text{data-param-lists-match (actuals, formals, alist)})\] 
\[\rightarrow \text{data-param-lists-match (actuals, formals, set-alist-value (x, y, alist))}\]


THEOREM: set-alist-value-preserves-listcars
\[\text{listcars (set-alist-value (x, y, alist)) = listcars (alist)}\]

THEOREM: set-alist-value-preserves-all-cars-unique
\[\text{all-cars-unique (alist)} \rightarrow \text{all-cars-unique (set-alist-value (x, y, alist))}\]

EVENT: Disable set-alist-value-preserves-all-cars-unique.

THEOREM: set-alist-value-preserves-signatures-match
\[\text{plistp (alist)} \rightarrow \text{signatures-match (alist, set-alist-value (x, y, alist))}\]


DEFINITION:
\[
\text{copy-out-params (formals, actuals, new-var-alist, old-var-alist)} = \begin{cases} 
\text{if formals} \simeq \text{nil then old-var-alist} \\
\text{else copy-out-params (cdr (formals),} \\
\text{cdr (actuals),} \\
\text{new-var-alist,} \\
\text{set-alist-value (car (actuals),} \\
\text{caddr (assoc (caar (formals),} \\
\text{new-var-alist)},} \\
\text{old-var-alist)) \text{endif} 
\end{cases}
\]

DEFINITION:
\[
\text{map-call-effects (new-state, def, stmt, old-state)} = \text{mg-state (convert-condition (cc (new-state),} \\
\text{def-conds (def),} \\
\text{call-conds (stmt),} \\
\text{copy-out-params (def-formals (def),} \\
\text{call-actuals (stmt),} \\
\text{mg-alist (new-state),} \\
\text{mg-alist (old-state)},} \\
\text{mg-psw (new-state))}
\]

29
Event: Disable map-call-effects.

**Theorem:** map-call-effects-preserves-normal

\[
\text{normal}(\texttt{new-state}) \rightarrow (\text{cc}(\text{map-call-effects}(\texttt{new-state}, \texttt{def}, \texttt{stmt}, \texttt{old-state})) = \textquote{normal})
\]

**Theorem:** map-call-effects-preserves-routineerror

\[
(\text{cc}(\texttt{new-state}) = \textquote{routineerror}) \rightarrow (\text{cc}(\text{map-call-effects}(\texttt{new-state}, \texttt{def}, \texttt{stmt}, \texttt{old-state})) = \textquote{routineerror})
\]

;;; This creates the part of the var alist for the call corresponding to the
;;; formals. The formal has form (name kind type) and the actual is either
;;; a literal or an identifierp.

**Definition:**

\[
\text{make-call-param-alist} (\texttt{formals}, \texttt{actuals}, \texttt{mg-alist}) = \begin{cases} 
\text{nil} & \text{if } \texttt{formals} \simeq \texttt{nil} \\
\text{cons} (\text{list} (\text{caar} (\texttt{formals}), \\
\text{cadar} (\texttt{formals}), \\
\text{caddr} (\text{assoc} (\text{car} (\texttt{actuals}), \texttt{mg-alist}))), \\
\text{make-call-param-alist} (\text{cdr} (\texttt{formals}), \\
\text{cdr} (\texttt{actuals}), \\
\texttt{mg-alist})) & \text{else}
\end{cases}
\]

**Theorem:** make-call-param-alist-plistp

\[
\text{plistp} (\text{make-call-param-alist} (\texttt{formals}, \texttt{actuals}, \texttt{alist}))
\]

**Theorem:** make-call-param-alist-preserves-listcars

\[
\text{listcars} (\text{make-call-param-alist} (\texttt{formals}, \texttt{actuals}, \texttt{mg-alist})) = \text{listcars} (\texttt{formals})
\]

**Definition:**

\[
\text{make-call-var-alist} (\texttt{mg-alist}, \texttt{stmt}, \texttt{def}) = \text{append} (\text{make-call-param-alist} (\text{def-formals} (\texttt{def}), \\
\text{call-actuals} (\texttt{stmt}), \\
\texttt{mg-alist}), \\
\text{def-locals} (\texttt{def}))
\]

;;; This doesn’t really need the hypothesis.

**Theorem:** plistp-make-call-var-alist

\[
\text{plistp} (\text{def-locals} (\texttt{def})) \rightarrow \text{plistp} (\text{make-call-var-alist} (\texttt{state}, \texttt{stmt}, \texttt{def}))
\]
**Definition:**
make-call-environment \((mg-state, stmt, def)\)  
\[
= mg-state('normal, 
    make-call-var-alist (mg-alist (mg-state), stmt, def), 
    mg-psw (mg-state))
\]

**Event:** Disable make-call-environment.

**Theorem:** make-call-environment-decomposition  
\[
(cc (make-call-environment (mg-state, stmt, def)) = 'normal) 
\land (mg-alist (make-call-environment (mg-state, stmt, def))) 
= make-call-var-alist (mg-alist (mg-state), stmt, def)) 
\land (mg-psw (make-call-environment (mg-state, stmt, def))) 
= mg-psw (mg-state))
\]

;;;;;; SEMANTICS FOR THE PREDEFINED PROCEDURES ;;;;;;;

;; These are structurally identical to the Piton version merely
;; altered to change 't and 'f to 'true-mg and 'false-mg.

**Definition:**
mg-bool \((x)\)  
\[
= \text{tag('boolean-mg, if } x \text{ then 'true-mg else 'false-mg endif)}
\]

**Definition:**
mg-or-bool \((x, y)\)  
\[
= \text{if } x = 'false-mg \text{ then } y \text{ else 'true-mg endif}
\]

**Definition:**
mg-and-bool \((x, y)\)  
\[
= \text{if } x = 'false-mg \text{ then 'false-mg else } y \text{ endif}
\]

**Definition:**
mg-not-bool \((x)\)  
\[
= \text{if } x = 'false-mg \text{ then 'true-mg else 'false-mg endif}
\]
DEFINITION:
fetch-array-element (a, i, alist) = get (i, caddr (assoc (a, alist)))

;; This returns the array with the substitution, not the resulting ;; alist.

DEFINITION:
put-array-element (a, i, val, alist) = put (val, i, caddr (assoc (a, alist)))

;; x := y -- y is a variable

DEFINITION:
mg-meaning-mg-simple-variable-assignment (stmt, mg-state) = mg-state ('normal,
set-alist-value (car (call-actuals (stmt))),
get-m-value (cadr (call-actuals (stmt))),
mg-alist (mg-state)),
mg-alist (mg-state)),
mg-psw (mg-state))

;; x := c -- c is a constant

DEFINITION:
mg-meaning-mg-simple-constant-assignment (stmt, mg-state) = mg-state ('normal,
set-alist-value (car (call-actuals (stmt))),
cadr (call-actuals (stmt))),
mg-alist (mg-state)),
mg-psw (mg-state))

;; b := (x = y)

DEFINITION:
mg-meaning-mg-simple-variable-eq (stmt, mg-state) = mg-state ('normal,
set-alist-value (car (call-actuals (stmt))),
mg-bool (untag (get-m-value (cadr (call-actuals (stmt))),
mg-alist (mg-state))))
               = untag (get-m-value (caddr (call-actuals (stmt))),
mg-alist (mg-state)),
mg-alist (mg-state)),
mg-psw (mg-state))
;; b := (x = c)

**Definition:**

\[
\text{mg-meaning-mg-simple-constant-eq}\ (\text{stmt, mg-state}) = \text{mg-state}\ ('\text{normal}, \ \\
\text{set-alist-value (car (call-actuals (stmt))),} \ \\
\text{mg-bool (untag (get-m-value (cadr (call-actuals (stmt)))),} \ \\
\text{mg-alist (mg-state)) = untag (caddr (call-actuals (stmt)))),} \ \\
\text{mg-alist (mg-state)),} \ \\
\text{mg-psw (mg-state))}
\]

;; b := (x le y) -- Here x and y are integer variables

**Definition:**

\[
\text{mg-meaning-mg-integer-le}\ (\text{stmt, mg-state}) = \text{mg-state}\ ('\text{normal}, \ \\
\text{set-alist-value (car (call-actuals (stmt))),} \ \\
\text{mg-bool (ileq (untag (get-m-value (cadr (call-actuals (stmt)))),} \ \\
\text{mg-alist (mg-state))),} \ \\
\text{untag (get-m-value (caddr (call-actuals (stmt)))),} \ \\
\text{mg-alist (mg-state))}},) \ \\
\text{mg-alist (mg-state)),} \ \\
\text{mg-psw (mg-state))}
\]

;; x := -y

**Definition:**

\[
\text{mg-meaning-mg-integer-unary-minus}\ (\text{stmt, mg-state}) = \text{let value be inegate (untag (get-m-value (cadr (call-actuals (stmt)))),} \ \\
\text{mg-alist (mg-state)) in} \ \\
\text{if small-integerp (value, MG-WORD-SIZE) then mg-state ('normal,} \ \\
\text{set-alist-value (car (call-actuals (stmt))),} \ \\
\text{tag ('int-mg, value),} \ \\
\text{mg-alist (mg-state),} \ \\
\text{mg-psw (mg-state))} \ \\
\text{else set-condition (mg-state, 'routineerror) endif endlet}
\]

;; x := y + z
Definition:
mg-meaning-mg-integer-add \((stmt, mg-state)\)
\[= \text{let } sum \text{ be } \text{iplus (untag (get-m-value (cadr (call-actuals \(stmt\))}, \text{mg-alist \(mg-state\))}, \text{untag (get-m-value (caddr (call-actuals \(stmt\))}, \text{mg-alist \(mg-state\)})})\]
\[\text{in}\]
\[\text{if small-integerp} (sum, MG-WORD-SIZE) \text{then } mg-state (’normal, set-alist-value (car (call-actuals \(stmt\)), tag (’int-mg, sum), mg-alist \(mg-state\)), mg-psw \(mg-state\))\]
\[\text{else } set-condition (mg-state, ’routineerror) endif endlet\]

;; x := y - z

Definition:
mg-meaning-mg-integer-subtract \((stmt, mg-state)\)
\[= \text{let } diff \text{ be } \text{idifference (untag (get-m-value (cadr (call-actuals \(stmt\))}, \text{mg-alist \(mg-state\))}, \text{untag (get-m-value (caddr (call-actuals \(stmt\))}, \text{mg-alist \(mg-state\)})})\]
\[\text{in}\]
\[\text{if small-integerp} (diff, MG-WORD-SIZE) \text{then } mg-state (’normal, set-alist-value (car (call-actuals \(stmt\)), tag (’int-mg, diff), mg-alist \(mg-state\)), mg-psw \(mg-state\))\]
\[\text{else } set-condition (mg-state, ’routineerror) endif endlet\]

;; b := b1 or b2

Definition:
mg-meaning-mg-boolean-or \((stmt, mg-state)\)
\[= \text{mg-state (’normal, set-alist-value (car (call-actuals \(stmt\)), tag (’boolean-mg, mg-or-bool (untag (get-m-value (cadr (call-actuals \(stmt\))}, \text{mg-alist \(mg-state\))}, \text{untag (get-m-value (caddr (call-actuals \(stmt\))}, \text{mg-alist \(mg-state\)})})}, \text{mg-alist \(mg-state\)})})\]

34
\[ \text{let } \text{index } = \text{untag (get-m-value (caddr (call-actuals (stmt), mg-alist (mg-state)))}, \]
\[ \text{in } \]
\[ \text{if } (\text{index } \in \mathbb{N}) \]
\[ \land \ (\text{index } < \text{array-length (get-m-type (cadr (call-actuals (stmt)), mg-alist (mg-state))))) \]
\[ \text{mg-meaning-mg-index-array (stmt, mg-state)} \]
\[ = \text{mg-state('normal, set-alist-value (car (call-actuals (stmt)), tag ('boolean-mg, mg-not-bool (untag (get-m-value (cadr (call-actuals (stmt)), mg-alist (mg-state)))))), mg-alist (mg-state))}, \]
\[ \text{mg-psw (mg-state))} \]

;; z := A[i] for A of size
;; The call is (predefined-proc-call-mg mg-index-array (z A i size))
then mg-state ('normal,
    set-alist-value (car (call-actuals (stmt)),
    fetch-array-element (cadr (call-actuals (stmt)),
        index,
        mg-alist (mg-state)),
    mg-alist (mg-state)),
    mg-psw (mg-state))
else set-condition (mg-state, 'routineerror) endif endlet

;; A[i] := v -- The actual argument list is (A i v size) where size is the
;;       array-length of the type of A. Here i and v are both variables.

Definition:
mg-meaning-mg-array-element-assignment (stmt, mg-state)
= let index be untag (get-m-value (cadr (call-actuals (stmt)),
            mg-alist (mg-state))),
    val be get-m-value (caddr (call-actuals (stmt)),
            mg-alist (mg-state))
in
if (index ∈ N)
    ∧ (index < array-length (get-m-type (car (call-actuals (stmt)),
            mg-alist (mg-state))))
then mg-state ('normal,
    set-alist-value (car (call-actuals (stmt)),
    put-array-element (car (call-actuals (stmt)),
        index,
        val,
        mg-alist (mg-state)),
    mg-alist (mg-state)),
    mg-psw (mg-state))
else set-condition (mg-state, 'routineerror) endif endlet

Definition:
mg-meaning-predefined-proc-call (stmt, mg-state)
= case on call-name (stmt):
    case = mg-simple-variable-assignment
    then mg-meaning-mg-simple-variable-assignment (stmt, mg-state)
    case = mg-simple-constant-assignment
    then mg-meaning-mg-simple-constant-assignment (stmt, mg-state)
    case = mg-simple-variable-eq
    then mg-meaning-mg-simple-variable-eq (stmt, mg-state)
    case = mg-simple-constant-eq
    then mg-meaning-mg-simple-constant-eq (stmt, mg-state)
case = mg-integer-le
then mg-meaning-mg-integer-le (stmt, mg-state)
case = mg-integer-unary-minus
then mg-meaning-mg-integer-unary-minus (stmt, mg-state)
case = mg-integer-add
then mg-meaning-mg-integer-add (stmt, mg-state)
case = mg-integer-subtract
then mg-meaning-mg-integer-subtract (stmt, mg-state)
case = mg-boolean-or
then mg-meaning-mg-boolean-or (stmt, mg-state)
case = mg-boolean-and
then mg-meaning-mg-boolean-and (stmt, mg-state)
case = mg-boolean-not
then mg-meaning-mg-boolean-not (stmt, mg-state)
case = mg-index-array
then mg-meaning-mg-index-array (stmt, mg-state)
case = mg-array-element-assignment
then mg-meaning-mg-array-element-assignment (stmt, mg-state)
otherwise mg-state endcase

Event: Disable mg-meaning-predefined.proc-call.

Definition:
mg-meaning (stmt, proc-list, mg-state, n) =
  if n ≃ 0 then signal-system-error (mg-state, 'timed-out)
  elseif ¬ normal (mg-state) then mg-state
  else case on car (stmt):
    case = no-op-mg
    then mg-state
    case = signal-mg
    then set-condition (mg-state, signalled-condition (stmt))
    case = prog2-mg
    then mg-meaning (prog2-right-branch (stmt), proc-list, mg-meaning (prog2-left-branch (stmt), proc-list, mg-state, n − 1), n − 1)
  case = loop-mg
  then remove-leave (mg-meaning (stmt, proc-list, mg-meaning (loop-body (stmt), proc-list, mg-state), mg-state),
\[ \text{case } = \text{if-mg} \]
\[ \text{then if mg-expression-falsep (if-condition} \ (\text{stmt}), \ mg\text{-state}) \]
\[ \text{then mg-meaning (if-false-branch} \ (\text{stmt}), \]
\[ \quad \text{proc-list}, \]
\[ \quad mg\text{-state}, \]
\[ \quad n - 1) \]
\[ \text{else mg-meaning (if-true-branch} \ (\text{stmt}), \]
\[ \quad \text{proc-list}, \]
\[ \quad mg\text{-state}, \]
\[ \quad n - 1) \text{endif} \]
\[ \text{case } = \text{begin-mg} \]
\[ \text{then if cc (mg-meaning (begin-body} \ (\text{stmt}), \]
\[ \quad \text{proc-list}, \]
\[ \quad mg\text{-state}, \]
\[ \quad n - 1)) \]
\[ \quad \in \text{when-labels} \ (\text{stmt}) \]
\[ \text{then mg-meaning (when-handler} \ (\text{stmt}), \]
\[ \quad \text{proc-list}, \]
\[ \quad \text{set-condition (mg-meaning (begin-body} \ (\text{stmt}), \]
\[ \quad \text{proc-list}, \]
\[ \quad mg\text{-state}, \]
\[ \quad n - 1), \]
\[ \quad '\text{normal}), \]
\[ \quad n - 1) \]
\[ \text{else mg-meaning (begin-body} \ (\text{stmt}), \]
\[ \quad \text{proc-list}, \]
\[ \quad mg\text{-state}, \]
\[ \quad n - 1) \text{endif} \]
\[ \text{case } = \text{proc-call-mg} \]
\[ \text{then map-call-effects (mg-meaning (def-body} \ (\text{fetch-called-def} \ (\text{stmt}, \]
\[ \quad \text{proc-list})), \]
\[ \quad \text{proc-list}, \]
\[ \quad \text{make-call-environment (mg-state}, \]
\[ \quad \text{stmt}, \]
\[ \quad \text{fetch-called-def} \ (\text{stmt}, \]
\[ \quad \text{proc-list})), \]
\[ \quad n - 1), \]
\[ \quad \text{fetch-called-def} \ (\text{stmt}, \]
\[ \quad \text{proc-list}), \]
\[ \quad \text{stmt}, \]
\[ \quad mg\text{-state}) \]
\[ \text{case } = \text{predefined-proc-call-mg} \]
\[ \text{then mg-meaning-predefined-proc-call} \ (\text{stmt}, \]
\[ \quad mg\text{-state}) \]
This is the version of mg-meaning with resource errors. It should be a theorem that in the absence of resource-error, it behaves exactly as mg-meaning.

The resource descriptor is a pair <temp-stk-size, ctrl-stk-size> where the two components are numberps which characterize the number of free slots on the top of the stacks. We cause a resource error if there is not enough space to continue without stack overflow. If the resource requirements exceed the available space, the cc is set to 'resource-error.

This computes the amount of space required for the storage of the locals. It is the number of simple variables plus the sum of the lengths of the arrays.

**Definition:**

\[
data-length \ (locals) = \begin{cases} 
0 & \text{if} \ locals \ \succapprox \ \text{nil} \\
1 + \text{data-length} \ (\text{cdr} \ (\text{car} \ (\text{locals}))) & \text{if} \ \text{simple-mg-type-refp} \ (\text{cadr} \ (\text{car} \ (\text{locals}))) \\
\text{array-length} \ (\text{cadr} \ (\text{car} \ (\text{locals}))) + \text{data-length} \ (\text{cdr} \ (\text{locals})) & \text{otherwise}
\end{cases}
\]

**Theorem:** data-length-not-zerop
\[
(\text{ok-mg-local-data-plistp} \ (\text{locals}) \ \land \ \text{listp} \ (\text{locals})) \ \rightarrow \ (\text{data-length} \ (\text{locals}) \ \not\approx \ 0)
\]

**Event:** Disable data-length-not-zerop.

**Theorem:** data-length-not-zerop2
\[
(\text{ok-mg-local-data-plistp} \ (\text{locals}) \ \land \ \text{listp} \ (\text{locals})) \ \rightarrow \ ((\text{data-length} \ (\text{locals}) \ \in \ \mathbb{N}) \ \land \ (\text{data-length} \ (\text{locals}) \ \neq \ 0))
\]

**Definition:**

\[
predefined-proc-call-temp-stk-requirement \ (\text{name}) = \begin{cases} 
\text{mg-simple-variable-assignment} & \text{if} \ \text{name} \\
\end{cases}
\]
then 2
  case = mg-simple-constant-assignment
  then 2
  case = mg-simple-variable-eq
  then 3
  case = mg-simple-constant-eq
  then 3
  case = mg-integer-le
  then 3
  case = mg-integer-unary-minus
  then 2
  case = mg-integer-add
  then 3
  case = mg-integer-subtract
  then 3
  case = mg-boolean-or
  then 3
  case = mg-boolean-and
  then 3
  case = mg-boolean-not
  then 2
  case = mg-index-array
  then 4
  case = mg-array-element-assignment
  then 4
otherwise 0 endcase

;; The number associated with each predefined procedure
;; represents the number of formals plus locals in the
;; Piton implementation. This is required because the
;; ctrl-stk requirements for the call p-frame is
;; (plus 2
;; (length (formal-vars def))
;; (length (temp-var-dcls def)))

DEFINITION:
predefined-proc-call-bindings-count (name) =
  case on name:
    case = mg-simple-variable-assignment
    then 2
    case = mg-simple-constant-assignment
    then 2
    case = mg-simple-variable-eq
then 3
case = mg-simple-constant-eq
  then 3
case = mg-integer-le
  then 3
case = mg-integer-unary-minus
  then 4
case = mg-integer-add
  then 4
case = mg-integer-subtract
  then 4
case = mg-boolean-or
  then 3
case = mg-boolean-and
  then 3
case = mg-boolean-not
  then 3
case = mg-index-array
  then 5
case = mg-array-element-assignment
  then 5
otherwise 0 endcase

Definition:
predefined-proc-call-p-frame-size (name)
= (1 + (1 + predefined-proc-call-bindings-count (name)))

;; I'm implementing the resources-available as a pair
;; <t-size c-size> of numberps.

Definition:  t-size (x) = car (x)

Definition:  c-size (x) = cadr (x)

;; An interesting fact is that the requirements for execution of a statement are not
;; dependent on the state. (Except in the case of begin where a when-label is
;; signalled.) This makes the computation of the resource requirements independent of
;; mg-meaning and may allow a much cleaner treatment.

Definition:
temp-stk-requirements (stmt, proc-list)
=  case on car (stmt):
    case = no-op-mg

then 0
  case = signal-mg
    then 1
  case = prog2-mg
    then 0
  case = loop-mg
    then 1
  case = if-mg
    then 1
  case = begin-mg
    then 1
  case = proc-call-mg
    then max (data-length (def-locals (fetch-called-def (stmt, proc-list)))
               + length (def-locals (fetch-called-def (stmt, proc-list))),
               1)
  case = predefined-proc-call-mg
    then predefined-proc-call-temp-stk-requirement (call-name (stmt))
otherwise 0 endcase

Definition:
ctrl-stk-requirements (stmt, proc-list)
  = case on car (stmt):
    case = no-op-mg
      then 0
    case = signal-mg
      then 0
    case = prog2-mg
      then 0
    case = loop-mg
      then 0
    case = if-mg
      then 0
    case = begin-mg
      then 0
    case = proc-call-mg
      then 2
        + length (def-locals (fetch-called-def (stmt, proc-list)))
        + length (def-formals (fetch-called-def (stmt, proc-list)))
    case = predefined-proc-call-mg
      then predefined-proc-call-p-frame-size (call-name (stmt))
otherwise 0 endcase

Event: Disable temp-stk-requirements.
EVENT: Disable ctrl-stk-requirements.

;; Resources are inadequate if I can’t perform the current operation without
;; running out of space. This can alternatively be phrased as follows.
;; In this version, the size-pair contains the
;; <current temp-stk length, current ctrl-stk length>

DEFINITION:
resources-inadequatep (stmt, proc-list, size-pair)
= ((temp-stk-requirements (stmt, proc-list)
   ⋄ (MG-MAX-TEMP-STK-SIZE – t-size (size-pair)))
∪ (ctrl-stk-requirements (stmt, proc-list)
   ⋄ (MG-MAX-CTRL-STK-SIZE – c-size (size-pair))))

EVENT: Disable resources-inadequatep.

DEFINITION:
mg-meaning-r (stmt, proc-list, mg-state, n, sizes)
= if n ≃ 0 then signal-system-error (mg-state, ’timed-out)
   elseif ¬ normal (mg-state) then mg-state
   elseif resources-inadequatep (stmt, proc-list, sizes)
    then signal-system-error (mg-state, ’resource-error)
   else case on car (stmt):
         case = no-op-mg
             then mg-state
         case = signal-mg
             then set-condition (mg-state, signalled-condition (stmt))
         case = prog2-mg
             then mg-meaning-r (prog2-right-branch (stmt),
                           proc-list,
                           mg-meaning-r (prog2-left-branch (stmt),
                           proc-list,
                           mg-state,
                           n – 1,
                           sizes),
                           n – 1,
                           sizes)
         case = loop-mg
             then remove-leave (mg-meaning-r (stmt,
                            proc-list,
                            mg-meaning-r (loop-body (stmt),
                            proc-list,
                            mg-state,
                            n – 1,
                            sizes),
                            n – 1,
                            sizes)
\[
\text{case } = \text{ if-mg} \\
\text{then if mg-expression-falsep (if-condition } (\text{stmt} ), \text{ mg-state)} \\
\text{then mg-meaning-r (if-false-branch } (\text{stmt} ), \text{ mg-state, proc-list, n - 1, sizes)} \\
\text{else mg-meaning-r (if-true-branch } (\text{stmt} ), \text{ proc-list, mg-state, n - 1, sizes)} \text{ endif} \\
\]

\[
\text{case } = \text{ begin-mg} \\
\text{then if cc (mg-meaning-r (begin-body } (\text{stmt} ), \text{ proc-list, mg-state, n - 1, sizes))} \in \text{ when-labels } (\text{stmt}) \\
\text{then mg-meaning-r (when-handler } (\text{stmt} ), \text{ proc-list, set-condition (mg-meaning-r (begin-body } (\text{stmt} ), \text{ proc-list, mg-state, n - 1, sizes)}, \text{ 'normal}), n - 1, sizes)} \\
\text{else mg-meaning-r (begin-body } (\text{stmt} ), \text{ proc-list, mg-state, n - 1, sizes)} \text{ endif} \\
\]

\[
\text{case } = \text{ proc-call-mg} \\
\text{then map-call-effects (mg-meaning-r (def-body (fetch-called-def } (\text{stmt}, \text{ proc-list))}, \text{ proc-list, make-call-environment (mg-state,)} \\
\]
stmt,
fetch-called-def (stmt,
proc-list)),

n − 1,
list (t-size (sizes)
+ data-length (def-locals (fetch-called-def (stmt,
proc-list))),
c-size (sizes)
+ (2
+ length (def-locals (fetch-called-def (stmt,
proc-list))))
+ length (def-formals (fetch-called-def (stmt,
proc-list))))),

fetch-called-def (stmt, proc-list),
stmt,
mg-state)

case = predefined-proc-call-mg
then mg-meaning-predefined-proc-call (stmt, mg-state)
otherwise mg-state endcase endif

THEOREM: map-call-effects-preserves-resource-errorp
resource-errorp (new-state)
→ resource-errorp (map-call-effects (new-state, def, stmt, old-state))

THEOREM: map-call-effects-preserves-mg-psw
mg-psw (map-call-effects (new-state, def, stmt, old-state))
= mg-psw (new-state)

THEOREM: mg-meaning-predefined-proc-call-preserves-resource-error
mg-psw (mg-meaning-predefined-proc-call (stmt, mg-state)) = mg-psw (mg-state)

THEOREM: resource-errors-propogate
resource-errorp (mg-state)
→ resource-errorp (mg-meaning-r (stmt, proc-list, mg-state, n, sizes))

THEOREM: resource-errors-propogate2
(mg-psw (mg-state) ≠ 'run)
→ (mg-psw (mg-meaning-r (stmt, proc-list, mg-state, n, sizes)) ≠ 'run)

;; This lemma shows that in the absence of resource errors, the two interpreters
;; are equivalent.

THEOREM: mg-meaning-equivalence
(¬ resource-errorp (mg-meaning-r (stmt, proc-list, mg-state, n, sizes)))
→ (mg-meaning-r (stmt, proc-list, mg-state, n, sizes)
= mg-meaning (stmt, proc-list, mg-state, n))
**Theorem**: \text{lessp-preserves-difference-lessp}
\[ ((y < (a - t\text{-size}_1)) \land (t\text{-size}_1 \not< t\text{-size}_2)) \rightarrow ((y < (a - t\text{-size}_2)) = t) \]

**Theorem**: \text{map-call-effects-preserves-resource-errorp2}
\[
\text{resource-errorp}(\text{map-call-effects} (\text{new-state, stmt, def, old-state})) = \text{resource-errorp}(\text{new-state})
\]

;; KEY POINT, t-size and c-size are the amount used, not the amount left

**Theorem**: \text{more-resources-preserves-resources-adequatep2}
\[
((t\text{-size}(\text{sizes}_1) \not< t\text{-size}(\text{sizes}_2)) \land (c\text{-size}(\text{sizes}_1) \not< c\text{-size}(\text{sizes}_2)) \land (\neg \text{resources-inadequatep}(\text{stmt, proc-list, sizes}_1))) \rightarrow (\text{resources-inadequatep}(\text{stmt, proc-list, sizes}_2) = f)
\]

**Event**: Disable mg-meaning-equivalence.

**Definition**:
\[
\text{meaning-induction-hint0}(\text{stmt, proc-list, mg-state, n, sizes}_1, \text{sizes}_2) = \begin{cases} 
\text{if } n \simeq 0 \text{ then } t \\
\text{elseif } \text{resources-inadequatep}(\text{stmt, proc-list, sizes}_1) \text{ then } t \\
\text{elseif } \neg \text{normal}(\text{mg-state}) \text{ then } t \\
\text{elseif } \text{'no-op-mg} = \text{car}(\text{stmt}) \text{ then } t \\
\text{elseif } \text{'signal-mg} = \text{car}(\text{stmt}) \text{ then } t \\
\text{elseif } \text{'prog2-mg} = \text{car}(\text{stmt}) \text{ then } \\
\text{meaning-induction-hint0}(\text{prog2-left-branch( stmt)}, \text{proc-list}, \text{mg-state}, n - 1, \text{sizes}_1, \text{sizes}_2) \\
\text{}\land\text{ meaning-induction-hint0}(\text{prog2-right-branch( stmt)}, \text{proc-list}, \text{mg-state}, n - 1, \text{sizes}_1, \text{sizes}_2) \\
\text{elseif } \text{'loop-mg} = \text{car}(\text{stmt})
\end{cases}
\]
then meaning-induction-hint0 (loop-body (stmt),
   proc-list,
   mg-state,
   n - 1,
   sizes1,
   sizes2)
∧ meaning-induction-hint0 (stmt,
   proc-list,
   mg-meaning-r (loop-body (stmt),
   proc-list,
   mg-state,
   n - 1,
   sizes1),
   n - 1,
   sizes1,
   sizes2)
elseif 'if-mg = car (stmt)
then meaning-induction-hint0 (if-false-branch (stmt),
   proc-list,
   mg-state,
   n - 1,
   sizes1,
   sizes2)
∧ meaning-induction-hint0 (if-true-branch (stmt),
   proc-list,
   mg-state,
   n - 1,
   sizes1,
   sizes2)
elseif 'begin-mg = car (stmt)
then meaning-induction-hint0 (begin-body (stmt),
   proc-list,
   mg-state,
   n - 1,
   sizes1,
   sizes2)
∧ meaning-induction-hint0 (when-handler (stmt),
   proc-list,
   set-condition (mg-meaning-r (begin-body (stmt),
   proc-list,
   mg-state,
   n - 1,
   sizes1),
   'normal),
\( n - 1, \\
\text{sizes1,} \\
\text{sizes2} \)

\textbf{elseif} 'proc-call-mg = \text{car(}\text{stmt}) \\
\textbf{then} \text{meaning-induction-hint0} \left( \text{def-body} \left( \text{fetch-called-def} \left( \text{stmt, proc-list} \right) \right), \\
\text{proc-list}, \\
\text{make-call-environment} \left( \text{mg-state,} \\
\text{stmt,} \\
\text{fetch-called-def} \left( \text{stmt, proc-list} \right) \right) \right), \\
\text{list} \left( \text{t-size} \left( \text{sizes1} \right) \\
+ \text{data-length} \left( \text{def-locals} \left( \text{fetch-called-def} \left( \text{stmt, proc-list} \right) \right) \right), \\
\text{c-size} \left( \text{sizes1} \right) \\
+ \left( 2 \\
+ \text{length} \left( \text{def-locals} \left( \text{fetch-called-def} \left( \text{stmt, proc-list} \right) \right) \right) \\
+ \text{length} \left( \text{def-formals} \left( \text{fetch-called-def} \left( \text{stmt, proc-list} \right) \right) \right) \right), \\
\text{list} \left( \text{t-size} \left( \text{sizes2} \right) \\
+ \text{data-length} \left( \text{def-locals} \left( \text{fetch-called-def} \left( \text{stmt, proc-list} \right) \right) \right), \\
\text{c-size} \left( \text{sizes2} \right) \\
+ \left( 2 \\
+ \text{length} \left( \text{def-locals} \left( \text{fetch-called-def} \left( \text{stmt, proc-list} \right) \right) \right) \\
+ \text{length} \left( \text{def-formals} \left( \text{fetch-called-def} \left( \text{stmt, proc-list} \right) \right) \right) \right) \right) \\
\text{elseif} '\text{predefined-proc-call-mg} = \text{car(}\text{stmt}) \text{ then t} \\
\text{else f endif}

\textbf{Theorem:} mg-meaning-equivalence3 \\
\left( \neg \text{resource-errorp} \left( \text{mg-meaning-r} \left( \left( \text{stmt, proc-list, mg-state, n, sizes1} \right) \right) \right) \right) \\
\land \left( \text{t-size} \left( \text{sizes1} \right) \nless \text{t-size} \left( \text{sizes2} \right) \right) \\
\land \left( \text{c-size} \left( \text{sizes1} \right) \nless \text{c-size} \left( \text{sizes2} \right) \right) \\
\rightarrow \left( \text{mg-meaning-r} \left( \left( \text{stmt, proc-list, mg-state, n, sizes1} \right) \right) = \text{mg-meaning-r} \left( \left( \text{stmt, proc-list, mg-state, n, sizes2} \right) \right) \right)

\textbf{Event:} Disable mg-meaning-equivalence3.

\textbf{Theorem:} mg-meaning-equivalence4 \\
\left( \neg \text{resource-errorp} \left( \text{mg-meaning-r} \left( \left( \text{stmt, proc-list, mg-state, n, sizes1} \right) \right) \right) \right) \\
\land \left( \text{t-size} \left( \text{sizes1} \right) \nless \text{t-size} \left( \text{sizes2} \right) \right)

48
\[\forall \ (c\text{-size} (\text{sizes}_1) \nless c\text{-size} (\text{sizes}_2)) \]
\[\rightarrow \ (\text{mg}\text{-meaning-r} (\text{stmt, proc-list, mg-state, n, sizes}_2) = \text{mg}\text{-meaning-r} (\text{stmt, proc-list, mg-state, n, sizes}_1))\]

**EVENT:** Disable mg-meaning-equivalence4.

**THEOREM:** more-resources-preserves-not-resource-errorp0
\[
((t\text{-size}_1 \nless t\text{-size}_2) \\
\wedge \ (c\text{-size}_1 \nless c\text{-size}_2) \\
\wedge \ \text{resource-errorp} (\text{mg}\text{-meaning-r} (\text{stmt, proc-list, mg-state, n, list (t\text{-size}_2, c\text{-size}_2))}) \\
\rightarrow \ \text{resource-errorp} (\text{mg}\text{-meaning-r} (\text{stmt, proc-list, mg-state, n, list (t\text{-size}_1, c\text{-size}_1))})
\]

**THEOREM:** more-resources-preserves-not-resource-errorp
\[
((t\text{-size}_1 \nless t\text{-size}_2) \\
\wedge \ (c\text{-size}_1 \nless c\text{-size}_2) \\
\wedge \ \neg \text{resource-errorp} (\text{mg}\text{-meaning-r} (\text{stmt, proc-list, mg-state, n, list (t\text{-size}_1, c\text{-size}_1))}) \\
\rightarrow \ \neg \text{resource-errorp} (\text{mg}\text{-meaning-r} (\text{stmt, proc-list, mg-state, n, list (t\text{-size}_2, c\text{-size}_2))})
\]

**THEOREM:** mg-meaning-equivalence2
\[
((\neg \text{resource-errorp} (\text{mg}\text{-meaning-r} (\text{stmt, proc-list, mg-state, n, list (t\text{-size}_1, c\text{-size}_1))}) \\
\wedge \ (t\text{-size}_1 \nless t\text{-size}_2) \\
\wedge \ (c\text{-size}_1 \nless c\text{-size}_2)) \\
\rightarrow \ (\text{mg}\text{-meaning-r} (\text{stmt, proc-list, mg-state, n, list (t\text{-size}_2, c\text{-size}_2))} = \text{mg}\text{-meaning} (\text{stmt, proc-list, mg-state, n}))
\]
THEOREM: mg-meaning-mg-meaning-r-resource-error-equivalence
(¬ resource-errorp (mg-meaning-r (stmt, proc-list, mg-state, n, sizes)))
→ (¬ resource-errorp (mg-meaning (stmt, proc-list, mg-state, n)))

THEOREM: zerop-n-mg-meaning
(n ≃ 0)
→ (mg-meaning (stmt, proc-list, mg-state, n)
= signal-system-error (mg-state, 'timed-out))

THEOREM: not-normal-mg-meaning
((n ≠ 0) ∧ (¬ normal (mg-state)))
→ (mg-meaning (stmt, proc-list, mg-state, n) = mg-state)

THEOREM: proc-call-meaning-2
(car (stmt) = 'proc-call-mg)
→ (mg-meaning (stmt, proc-list, mg-state, n)
= if n ≃ 0 then signal-system-error (mg-state, 'timed-out)
elseif ¬ normal (mg-state) then mg-state
else map-call-effects (mg-meaning (def-body (fetch-called-def (stmt, proc-list)),
proc-list,
make-call-environment (mg-state, stmt, fetch-called-def (stmt, proc-list)),
n − 1),
fetch-called-def (stmt, proc-list), stmt, mg-state) endif)

;; The versions for mg-meaning-r

THEOREM: zerop-n-mg-meaning-r
(n ≃ 0)
→ (mg-meaning-r (stmt, proc-list, mg-state, n, sizes)
= signal-system-error (mg-state, 'timed-out))

THEOREM: not-normal-mg-meaning-r
((n ≠ 0) ∧ (¬ normal (mg-state)))
→ (mg-meaning-r (stmt, proc-list, mg-state, n, sizes) = mg-state)

THEOREM: resources-inadequatep-mg-meaning-r
((n ≠ 0)
∧ normal (mg-state)
∧ resources-inadequatep (stmt, proc-list, sizes))
→ (mg-meaning-r (stmt, proc-list, mg-state, n, sizes)
  = signal-system-error (mg-state, 'resource-error))

EVENT: Disable mg-meaning-r.

THEOREM: call-cond-lists-lengths-match
((proc-call-mg = car (stmt))
∧ ok-mg-statement (stmt, r-cond-list, name-alist, proc-list))
→ (length (def-conds (fetch-called-def (stmt, proc-list)))
  = length (call-conds (stmt)))

EVENT: Disable call-cond-lists-lengths-match.

THEOREM: set-alist-value-preserves-mg-alistp
(mg-alistp (alist) ∧ ok-mg-valuep (val, cadr (assoc (name, alist))))
→ mg-alistp (set-alist-value (name, val, alist))

THEOREM: mg-alistps-append
(mg-alistp (lst1) ∧ mg-alistp (lst2)) → mg-alistp (append (lst1, lst2))

;; The recognizer has a structure called the name-alist of the following
;; form:
;; (... (namei typei other-stuff) ...)
;; This allows the identification of the types of variables. It should be
;; the case that the values of the variables in the meaning alist correspond
;; to their types on the name-alist.

;; This says that a variable on the name-alist has the same name and type as
;; on the variable alist. I need to know this to guarantee that the checks for
;; legality are visible in the execution world.

;; The only time a name-alist is ever created in the recognizer is from the formal
;; and locals lists. There is really no reason why the same structure couldn’t
;; be adhered to in the interpreter. That is, the order of the variables could
;; be maintained. >> Where would the initial values of the variables come from
;; in that case?
;; I could have an initial alist which serves both as the name-alist and var-alist
;; for the execution of the stmt. That is, the initial alist only has to be an
;; assignment of values to the vars which is consistent with the types.

;; Notice that this checks a very strong correspondence between the two alists.
;; Each element agrees in name and type. The idea here is that the name and var
;; alists are really identical except that some values have been added on the
;; Notice that this correspondence really defines an equivalence relation. I
;; don’t know that I’ll need the full power of this.

**Theorem**: signatures-match-preserves-get-m-type

```
signatures-match (alist1, alist2)
→ (cadr (assoc x, alist1)) = cadr (assoc x, alist2))
```

**Event**: Disable signatures-match-preserves-get-m-type.

**Theorem**: signatures-match-preserves-definedp

```
signatures-match (alist1, alist2)
→ (definedp (x, alist1) = definedp (x, alist2))
```

**Theorem**: signatures-match-preserves-ok-actual-params-list

```
(signatures-match (alist1, alist2) ∧ ok-actual-params-list (lst, alist1))
→ ok-actual-params-list (lst, alist2)
```

**Theorem**: set-alist-value-preserves-plistp

```
plistp (lst) → plistp (set-alist-value (name, val, lst))
```

**Theorem**: signatures-match-preserves-boolean-identifierp

```
(signatures-match (alist1, alist2) ∧ boolean-identifierp (b, alist1))
→ boolean-identifierp (b, alist2)
```

**Event**: Disable signatures-match-preserves-boolean-identifierp.

**Theorem**: signatures-match-preserves-int-identifierp

```
(signatures-match (alist1, alist2) ∧ int-identifierp (x, alist1))
→ int-identifierp (x, alist2)
```

**Event**: Disable signatures-match-preserves-int-identifierp.

**Theorem**: signatures-match-preserves-character-identifierp

```
(signatures-match (alist1, alist2) ∧ character-identifierp (x, alist1))
→ character-identifierp (x, alist2)
```

**Event**: Disable signatures-match-preserves-character-identifierp.

**Theorem**: signatures-match-preserves-array-identifierp

```
(signatures-match (alist1, alist2) ∧ array-identifierp (x, alist1))
→ array-identifierp (x, alist2)
```

52
Event: Disable signatures-match-preserves-array-identifierp.

**Theorem:** signatures-match-preserves-simple-identifierp
\[(\text{signatures-match } (\text{alist1}, \text{alist2}) \land \text{simple-identifierp } (x, \text{alist1})) \rightarrow \text{simple-identifierp } (x, \text{alist2})\]

**Theorem:** signatures-match-preserves-simple-typed-identifierp
\[(\text{signatures-match } (\text{alist1}, \text{alist2}) \land \text{simple-typed-identifierp } (x, \text{type}, \text{alist1})) \rightarrow \text{simple-typed-identifierp } (x, \text{type}, \text{alist2})\]

**Theorem:** signatures-match-preserves-data-param-lists-match
\[(\text{signatures-match } (\text{alist1}, \text{alist2}) \land \text{data-param-lists-match } (\text{actuals}, \text{formals}, \text{alist1})) \rightarrow \text{data-param-lists-match } (\text{actuals}, \text{formals}, \text{alist2})\]

**Theorem:** signatures-match-preserves-ok-predefined-proc-args
\[(\text{signatures-match } (\text{alist1}, \text{alist2}) \land \text{ok-predefined-proc-args } (\text{name}, \text{actuals}, \text{alist1})) \rightarrow \text{ok-predefined-proc-args } (\text{name}, \text{actuals}, \text{alist2})\]

**Theorem:** signatures-match-preserves-ok-predefined-proc-call
\[(\text{signatures-match } (\text{alist1}, \text{alist2}) \land \text{ok-predefined-proc-call } (\text{stmt}, \text{alist1})) \rightarrow \text{ok-predefined-proc-call } (\text{stmt}, \text{alist2})\]

**Theorem:** signatures-match-preserves-ok-mg-statement
\[(\text{signatures-match } (\text{alist1}, \text{alist2}) \land \text{ok-mg-statement } (\text{stmt}, \text{r-cond-list}, \text{alist1}, \text{proc-list})) \rightarrow \text{ok-mg-statement } (\text{stmt}, \text{r-cond-list}, \text{alist2}, \text{proc-list})\]

**Theorem:** mg-meaning-predefined-proc-call-preserves-signatures-match
\[\text{plistp } (\text{mg-alist } (\text{mg-state})) \rightarrow \text{signatures-match } (\text{mg-alist } (\text{mg-state}), \text{mg-alist } (\text{mg-meaning-predefined-proc-call } (\text{stmt}, \text{mg-state})))\]

**Theorem:** copy-out-params-preserves-signatures-match
\[\text{plistp } (v) \rightarrow \text{signatures-match } (v, \text{copy-out-params } (x, y, z, v))\]

**Theorem:** signatures-match-preserves-plistp
\[(\text{plistp } (x) \land \text{signatures-match } (x, y)) \rightarrow \text{plistp } (y)\]

**Theorem:** mg-meaning-preserves-signatures-match
\[\text{plistp } (\text{mg-alist } (\text{mg-state})) \rightarrow \text{signatures-match } (\text{mg-alist } (\text{mg-state}), \text{mg-alist } (\text{mg-meaning } (\text{stmt}, \text{proc-list}, \text{mg-state}, n)))\]
Theorem: cadr-litatom-implies-definedp
\((\text{cadr (assoc } x, \text{alist}) \neq 0) \rightarrow \text{definedp } (x, \text{alist})\)

Event: Disable cadr-litatom-implies-definedp.

Theorem: call-param-alist-mg-alistp
\((\text{mg-alistp } (mg-alist) \land \text{ok-mg-formal-data-params-plistp } (formals) \land \text{data-param-lists-match } (actu}}als, \text{formals, name-alist}) \land \text{signatures-match } (mg-alist, \text{name-alist})) \rightarrow \text{mg-alistp } (\text{make-call-param-alist } (formals, \text{actuals, mg-alist}))\)

Theorem: call-locals-alist-mg-alistp
\((\text{ok-mg-local-data-plistp } (\text{locals-list}) \rightarrow \text{mg-alistp } (\text{locals-list}))\)

Theorem: make-call-var-alist-mg-alistp
\((\text{proc-call-mg = car } (\text{stmt}) \land \text{ok-mg-statement } (\text{stmt, r-cond-list, name-alist, proc-list}) \land \text{ok-mg-def-plistp } (\text{proc-list}) \land \text{mg-alistp } (\text{mg-alist}) \land \text{signatures-match } (mg-alist, \text{name-alist})) \rightarrow \text{mg-alistp } (\text{make-call-var-alist } (mg-alist, \text{stmt, fetch-called-def } (\text{stmt, proc-list})))\)

Theorem: ok-mg-statep-preserved-call-case
\((\text{proc-call-mg = car } (\text{stmt}) \land \text{ok-mg-statep } (\text{mg-state, r-cond-list}) \land \text{ok-mg-statement } (\text{stmt, r-cond-list, name-alist, proc-list}) \land \text{signatures-match } (mg-alist, \text{name-alist}) \land \text{ok-mg-def-plistp } (\text{proc-list})) \rightarrow \text{ok-mg-statep } (\text{make-call-environment } (mg-state, \text{stmt, fetch-called-def } (\text{stmt, proc-list}), \text{make-cond-list } (\text{fetch-called-def } (\text{stmt, proc-list}))))\)

Theorem: call-formal-signatures-match
\(\text{signatures-match } (\text{make-alist-from-formals } (\text{formals}), \text{make-call-param-alist } (\text{formals, actuals, mg-alist}))\)

Theorem: call-formal-signatures-match2
\(\text{signatures-match } (\text{make-call-param-alist } (\text{formals, actuals, mg-alist}), \text{make-alist-from-formals } (\text{formals}))\)

Theorem: call-local-signatures-match
\(\text{plistp } (\text{locals}) \rightarrow \text{signatures-match } (\text{make-alist-from-formals } (\text{locals}), \text{locals})\)

54
**Theorem:** call-local-signatures-match2
signatures-match (locals, make-alist-from-formals (locals))

**Theorem:** call-signatures-match1
plistp (def-locals (def))
→ signatures-match (make-name-alist (def),
make-call-var-alist (mg-alist, stmt, def))

**Theorem:** call-signatures-match2
signatures-match (make-call-var-alist (mg-alist, stmt, def), make-name-alist (def))

**Theorem:** call-signatures-match3
signatures-match (mg-alist (make-call-environment (mg-state, stmt, fetch-called-def (stmt, proc-list))),
make-name-alist (fetch-called-def (stmt, proc-list)))

**Definition:**
formal-types-preserved (formals, alist) =
if formals ≍ nil then t
else definedp (caar (formals), alist)
∧ (cadar (formals) = cadr (assoc (caar (formals), alist)))
∧ formal-types-preserved (cdr (formals), alist) endif

**Theorem:** formal-types-preserved-append
formal-types-preserved (formals, lst1)
→ formal-types-preserved (formals, append (lst1, lst2))

**Theorem:** formal-types-unaffected-by-extra-binding
(car (x) \notin listcars (y))
→ (formal-types-preserved (y, cons (x, z)) = formal-types-preserved (y, z))

**Theorem:** formal-types-preserved-in-call-param-alist
(ok-mg-formal-data-params-plistp (formals)
∧ no-duplicates (listcars (formals)))
→ formal-types-preserved (formals,
make-call-param-alist (formals, actuals, mg-alist))

**Theorem:** formal-types-preserved-in-call-environment
((proc-call-mg = car (stmt))
∧ ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)
∧ ok-mg-def-plistp (proc-list)
∧ mg-alistp (mg-alist (mg-state)))
formal-types-preserved (def-formals (fetch-called-def (stmt, proc-list)),
mg-alist (make-call-environment (mg-state,
stmt,
fetch-called-def (stmt, proc-list))))

Theorem: copy-out-params-preserves-mg-alistp
(mg-alistp (old-alist)
∧ mg-alistp (new-alist)
∧ formal-types-preserved (formals, new-alist)
∧ data-param-lists-match (actuals, formals, name-alist)
∧ signatures-match (old-alist, name-alist)
∧ ok-mg-formal-data-params-plistp (formals))
→ mg-alistp (copy-out-params (formals, actuals, new-alist, old-alist))

Theorem: formal-types-preserved-in-matching-signatures
(mg-name-alistp (old-alist)
∧ formal-types-preserved (formals, old-alist)
∧ signatures-match (old-alist, new-alist))
→ formal-types-preserved (formals, new-alist)

;; This is the case needed for map-call-effects-preserves-ok-state.

Theorem: map-call-effects-preserves-mg-alistp
((‘proc-call-mg = car (stmt))
∧ mg-alistp (mg-alist (mg-state))
∧ ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)
∧ signatures-match (mg-alist (mg-state), name-alist)
∧ ok-mg-def-plistp (proc-list)
∧ ok-mg-def (fetch-called-def (stmt, proc-list), proc-list)
∧ mg-alistp (mg-alist (mg-meaning (def-body (fetch-called-def (stmt,
proc-list)),
proc-list,
make-call-environment (mg-state,
stmt,
fetch-called-def (stmt, proc-list)),
n − 1)))))
→ mg-alistp (copy-out-params (def-formals (fetch-called-def (stmt, proc-list)),
call-actuals (stmt),
mg-alist (mg-meaning (def-body (fetch-called-def (stmt,
proc-list)),
proc-list,
make-call-environment (mg-state,
Theorem: convert-condition1-membership
\[ \text{length (def-conds)} = \text{length (call-conds)} \]
\[ \rightarrow (\text{convert-condition1 (cc, def-conds, call-conds)} \]
\[ \in \text{cons ('routineerror, call-conds}) \]

Theorem: cond-identifier-plistp-preserves-membership
\[ ((\text{cc} \in \text{lst1}) \]
\[ \land \text{cond-identifier-plistp (lst1, lst2}) \]
\[ \land (\text{cc} \neq \text{'routineerror}) \]
\[ \rightarrow (\text{cc} \in \text{lst2}) \]

Event: Disable cond-identifier-plistp-preserves-membership.

Theorem: cons-preserves-membership
\[ ((x \in \text{cons (y, z))}) \land (x \neq y) \rightarrow (x \in z) \]

Event: Disable cons-preserves-membership.

Theorem: cond-identifier-conversion-litatom
\[ ((\text{length (def-conds)} = \text{length (call-conds)}) \]
\[ \land \text{cond-identifier-plistp (call-conds, cond-list}) \]
\[ \rightarrow \text{litatom (convert-condition1 (cc, def-conds, call-conds}) \]

Event: Disable cond-identifier-conversion-litatom.

Theorem: map-call-effects-preserves-ok-state
\[ (\text{'proc-call-mg} = \text{car (stmt)}) \]
\[ \land \text{ok-mg-statep (mg-state, r-cond-list}) \]
\[ \land \text{ok-mg-statement (stmt, r-cond-list, name-alist, proc-list}) \]
\[ \land \text{signatures-match (mg-alist (mg-state), name-alist}) \]
\[ \land \text{ok-mg-def-plistp (proc-list}) \]
\[ \land \text{ok-mg-statep (mg-meaning (def-body (fetch-called-def (stmt, proc-list)), proc-list, make-call-environment (mg-state, stmt, fetch-called-def (stmt, proc-list)), mg-alist (mg-state))))} \]

57
\[ n - 1, \]
\[ \text{make-cond-list (fetch-called-def (stmt, proc-list)))} \]
\[ \rightarrow \text{ok-mg-statep (map-call-effects (mg-meaning (def-body (fetch-called-def (stmt, proc-list)), proc-list)), make-call-environment (mg-state, stmt, fetch-called-def (stmt, proc-list))), n - 1, \]
\[ \text{fetch-called-def (stmt, proc-list)}, stmt, mg-state, r-cond-list) \]

**Theorem:** simple-typed-literalp-ok-valuep

simple-typed-literalp (exp, type) \(\rightarrow\) ok-mg-valuep (exp, type)

**Event:** Enable ok-predefined-proc-call.

**Event:** Enable ok-predefined-proc-args.

**Event:** Enable ok-mg-statement.

**Event:** Enable simple-identifierp-implies-definedp.

**Event:** Enable int-identifierp-implies-definedp.

**Event:** Enable boolean-identifierp-implies-definedp.

**Event:** Enable character-identifierp-implies-definedp.

**Event:** Enable boolean-identifierp.

**Event:** Enable character-identifierp.

**Event:** Enable int-identifierp.

**Event:** Disable mg-bool.

58
EVENT: Disable mg-or-bool.

EVENT: Disable mg-and-bool.

EVENT: Disable mg-not-bool.

THEOREM: tag-length-plistp
length-plistp (tag (x, y), 2)

THEOREM: car-tag
(car (tag (x, y)) = x) ∧ (cdr (tag (x, y)) = list (y))

THEOREM: simple-typed-literalp-boolean-literals
simple-typed-literalp (tag ('boolean-mg, 'true-mg), 'boolean-mg)∧
simple-typed-literalp (tag ('boolean-mg, 'false-mg), 'boolean-mg)

THEOREM: simple-typed-literalp-boolean-mg-bool
simple-typed-literalp (mg-bool (x), 'boolean-mg)

THEOREM: simple-typed-literalp-boolean-mg-bool-not
simple-typed-literalp (tag ('boolean-mg, mg-not-bool (x)), 'boolean-mg)

THEOREM: ok-mg-valuep-int-mg
ok-mg-valuep (tag ('int-mg, x), 'int-mg) = small-integerp (x, MG-WORD-SIZE)

THEOREM: boolean-literalp-tag-untag
boolean-literalp (x) → boolean-literalp (tag ('boolean-mg, untag (x)))

THEOREM: boolean-identifier-boolean-literal-value
(boolean-identifierp (x, name-alist)∧
mg-alistp (mg-alist)∧
signatures-match (mg-alist, name-alist))→
boolean-literalp (caddr (assoc (x, mg-alist)))

THEOREM: mg-meaning-mg-simple-variable-assignment-preserves-ok-mg-statep
((car (stmt) = 'predefined-proc-call-mg)∧
(call-name (stmt) = 'mg-simple-variable-assignment)∧
ok-mg-statep (mg-state, r-cond-list))
\(\land\) ok-mg-statement \((\text{stmt}, \text{r-cond-list}, \text{name-alist}, \text{proc-list})\)  
\(\land\) signatures-match \((\text{mg-alist} (\text{mg-state}), \text{name-alist})\)  
\(\rightarrow\) ok-mg-statep \((\text{mg-meaning-predefined-proc-call (stmt, mg-state)}, \text{r-cond-list})\)

**Theorem:** mg-meaning-mg-simple-constant-assignment-preserves-ok-mg-statep  
\((\text{car (stmt)} = \text{'predefined-proc-call-mg}) \land \text{(call-name (stmt) = 'mg-simple-constant-assignment})\land \text{ok-mg-statep (mg-state, r-cond-list)}) \land \text{ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)}) \land \text{signatures-match (mg-alist (mg-state), name-alist)}) \rightarrow \text{ok-mg-statep (mg-meaning-predefined-proc-call (stmt, mg-state), r-cond-list)})

**Theorem:** mg-meaning-mg-simple-variable-eq-preserves-ok-mg-statep  
\((\text{car (stmt)} = \text{'predefined-proc-call-mg}) \land \text{(call-name (stmt) = 'mg-simple-variable-eq})\land \text{ok-mg-statep (mg-state, r-cond-list)}) \land \text{ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)}) \land \text{signatures-match (mg-alist (mg-state), name-alist)}) \rightarrow \text{ok-mg-statep (mg-meaning-predefined-proc-call (stmt, mg-state), r-cond-list)})

**Theorem:** mg-meaning-mg-simple-constant-eq-preserves-ok-mg-statep  
\((\text{car (stmt)} = \text{'predefined-proc-call-mg}) \land \text{(call-name (stmt) = 'mg-simple-constant-eq})\land \text{ok-mg-statep (mg-state, r-cond-list)}) \land \text{ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)}) \land \text{signatures-match (mg-alist (mg-state), name-alist)}) \rightarrow \text{ok-mg-statep (mg-meaning-predefined-proc-call (stmt, mg-state), r-cond-list)})

**Theorem:** mg-meaning-mg-integer-le-preserves-ok-mg-statep  
\((\text{car (stmt)} = \text{'predefined-proc-call-mg}) \land \text{(call-name (stmt) = 'mg-integer-le})\land \text{ok-mg-statep (mg-state, r-cond-list)}) \land \text{ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)}) \land \text{signatures-match (mg-alist (mg-state), name-alist)}) \rightarrow \text{ok-mg-statep (mg-meaning-predefined-proc-call (stmt, mg-state), r-cond-list)})

**Theorem:** mg-meaning-mg-integer-unary-minus-preserves-ok-mg-statep  
\((\text{car (stmt)} = \text{'predefined-proc-call-mg}) \land \text{(call-name (stmt) = 'mg-integer-unary-minus})\land \text{ok-mg-statep (mg-state, r-cond-list)}) \land \text{ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)}) \land \text{signatures-match (mg-alist (mg-state), name-alist)}) \rightarrow \text{ok-mg-statep (mg-meaning-predefined-proc-call (stmt, mg-state), r-cond-list)})
\[\begin{align*}
\text{∀ } & \text{ok-mg-statement}(\text{stmt, r-cond-list, name-alist, proc-list}) \\
\text{∀ } & \text{signatures-match}(\text{mg-alist}(\text{mg-state}), \text{name-alist}) \\
\rightarrow & \text{ok-mg-statep}(\text{mg-meaning-predefined-proc-call}(\text{stmt, mg-state}), r\text{-cond-list})
\end{align*}\]

**Theorem:** \(\text{mg-meaning-mg-integer-add-preserves-ok-mg-statep}\)

\[
((\text{car}(\text{stmt}) = \text{'predefined-proc-call-mg}) \\
\land (\text{call-name}(\text{stmt}) = \text{'mg-integer-add}) \\
\land \text{ok-mg-statep}(\text{mg-state, r-cond-list}) \\
\land \text{ok-mg-statement}(\text{stmt, r-cond-list, name-alist, proc-list}) \\
\land \text{signatures-match}(\text{mg-alist}(\text{mg-state}), \text{name-alist})) \\
\rightarrow \text{ok-mg-statep}(\text{mg-meaning-predefined-proc-call}(\text{stmt, mg-state}), r\text{-cond-list})
\]

**Theorem:** \(\text{mg-meaning-mg-integer-subtract-preserves-ok-mg-statep}\)

\[
((\text{car}(\text{stmt}) = \text{'predefined-proc-call-mg}) \\
\land (\text{call-name}(\text{stmt}) = \text{'mg-integer-subtract}) \\
\land \text{ok-mg-statep}(\text{mg-state, r-cond-list}) \\
\land \text{ok-mg-statement}(\text{stmt, r-cond-list, name-alist, proc-list}) \\
\land \text{signatures-match}(\text{mg-alist}(\text{mg-state}), \text{name-alist})) \\
\rightarrow \text{ok-mg-statep}(\text{mg-meaning-predefined-proc-call}(\text{stmt, mg-state}), r\text{-cond-list})
\]

**Theorem:** \(\text{mg-meaning-mg-boolean-or-preserves-ok-mg-statep}\)

\[
((\text{car}(\text{stmt}) = \text{'predefined-proc-call-mg}) \\
\land (\text{call-name}(\text{stmt}) = \text{'mg-boolean-or}) \\
\land \text{ok-mg-statep}(\text{mg-state, r-cond-list}) \\
\land \text{ok-mg-statement}(\text{stmt, r-cond-list, name-alist, proc-list}) \\
\land \text{signatures-match}(\text{mg-alist}(\text{mg-state}), \text{name-alist})) \\
\rightarrow \text{ok-mg-statep}(\text{mg-meaning-predefined-proc-call}(\text{stmt, mg-state}), r\text{-cond-list})
\]

**Theorem:** \(\text{mg-meaning-mg-boolean-and-preserves-ok-mg-statep}\)

\[
((\text{car}(\text{stmt}) = \text{'predefined-proc-call-mg}) \\
\land (\text{call-name}(\text{stmt}) = \text{'mg-boolean-and}) \\
\land \text{ok-mg-statep}(\text{mg-state, r-cond-list}) \\
\land \text{ok-mg-statement}(\text{stmt, r-cond-list, name-alist, proc-list}) \\
\land \text{signatures-match}(\text{mg-alist}(\text{mg-state}), \text{name-alist})) \\
\rightarrow \text{ok-mg-statep}(\text{mg-meaning-predefined-proc-call}(\text{stmt, mg-state}), r\text{-cond-list})
\]

**Theorem:** \(\text{mg-meaning-mg-boolean-not-preserves-ok-mg-statep}\)

\[
((\text{car}(\text{stmt}) = \text{'predefined-proc-call-mg}) \\
\land (\text{call-name}(\text{stmt}) = \text{'mg-boolean-not}) \\
\land \text{ok-mg-statep}(\text{mg-state, r-cond-list})
\]

61
\[ \land \text{ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)} \]
\[ \land \text{signatures-match (mg-alist (mg-state), name-alist)} \]
\[ \rightarrow \text{ok-mg-statep (mg-meaning-predefined-proc-call (stmt, mg-state), r-cond-list)} \]

**Theorem**: simple-typed-literal-list-elements

\[ \text{(simple-typed-literal-plistp (lst, type) \land (i < \text{length (lst))})} \]
\[ \rightarrow \text{simple-typed-literalp (get (i, lst, type)} \]

**Theorem**: index-array-mg-alist-elementp

\[ \text{(mg-alistp (mg-alist) \land array-identifierp (a, mg-alist) \land (i < \text{array-length (cadr (assoc (a, mg-alist))}))} \]
\[ \rightarrow \text{simple-typed-literalp (get (i, caddr (assoc (a, mg-alist))), array-elemtyppe (cadr (assoc (a, mg-alist))))} \]

**Theorem**: put-preserves-simple-typed-literal-plistp

\[ \text{(simple-typed-literal-plistp (lst, type) \land (i < \text{length (lst)})} \]
\[ \land \text{simple-typed-literalp (val, type))} \]
\[ \rightarrow \text{simple-typed-literal-plistp (put (val, i, lst, type)} \]

**Theorem**: simple-type-literal-plistp-ok-valuep

\[ \text{(array-mg-type-refp (type) \land simple-typed-literal-plistp (exp, array-elemtype (type)) \land (length (exp) = \text{array-length (type)})} \]
\[ \rightarrow \text{ok-mg-valuep (exp, type)} \]

**Theorem**: mg-meaning-mg-index-array-preserves-ok-mg-statep

\[ ((\text{car (stmt) = predefined-proc-call-mg}}) \land (\text{call-name (stmt) = mg-index-array}) \land \text{ok-mg-statep (mg-state, r-cond-list)} \land \text{ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)} \land \text{signatures-match (mg-alist (mg-state), name-alist))} \]
\[ \rightarrow \text{ok-mg-statep (mg-meaning-predefined-proc-call (stmt, mg-state), r-cond-list)} \]

**Theorem**: array-identifiers-have-array-types

\[ (\text{array-identifierp (x, mg-alist) \land mg-alistp (mg-alist))} \]
\[ \rightarrow \text{array-mg-type-refp (cadr (assoc (x, mg-alist))))} \]

**Theorem**: array-identifiers-have-array-types2

\[ (\text{array-identifierp (a, mg-alist) \land mg-alistp (mg-alist))} \]
\[ \rightarrow \text{simple-typed-literal-plistp (caddr (assoc (a, mg-alist)), array-elemtyppe (cadr (assoc (a, mg-alist))))} \]
Theorem: array-identifier-lengths-match
(array-identifierp \( a, mg-alist \) \( \land \) mg-alistp (mg-alist))
\( \rightarrow \) (length (caddr (assoc \( a, mg-alist \))))
\( = \) array-length (cadr (assoc \( a, mg-alist \))))

Theorem: simple-typed-identifier-has-simple-typed-literal-value
(mg-alistp (mg-alist) \( \land \) simple-typed-identifierp \( x, type, mg-alist \))
\( \rightarrow \) simple-typed-literalp (caddr (assoc \( x, mg-alist \)), type)

Theorem: mg-meaning-mg-array-element-assignment-preserves-ok-mg-statep
((car \( stmt \)) = 'predefined-proc-call-mg)
\( \land \) (call-name \( stmt \) = 'mg-array-element-assignment)
\( \land \) ok-mg-statep (mg-state, r-cond-list)
\( \land \) ok-mg-statement \( stmt, r-cond-list, name-alist, proc-list \)
\( \land \) signatures-match (mg-alist (mg-state), name-alist))
\( \rightarrow \) ok-mg-statep (mg-meaning-predefined-proc-call \( stmt, mg-state \), r-cond-list)

Theorem: mg-meaning-predefined-proc-call-preserves-ok-mg-statep
((car \( stmt \)) = 'predefined-proc-call-mg)
\( \land \) ok-mg-statep (mg-state, r-cond-list)
\( \land \) ok-mg-statement \( stmt, r-cond-list, name-alist, proc-list \)
\( \land \) signatures-match (mg-alist (mg-state), name-alist))
\( \rightarrow \) ok-mg-statep (mg-meaning-predefined-proc-call \( stmt, mg-state \), r-cond-list)

Theorem: set-condition-normal-preserves-ok-mg-statep
ok-mg-statep (state, cond-list1)
\( \rightarrow \) ok-mg-statep (set-condition \( state, normal \), cond-list2)

Theorem: append-conditions-preserves-ok-mg-statep
(ok-mg-statep \( state, append \( lst, lst2 \) \) \( \land \) (cc \( state \) \( \not\in \) lst))
\( \rightarrow \) ok-mg-statep \( state, lst2 \)

Definition:
meaning-induction-hint \( stmt, proc-list, mg-state, n, name-alist, r-cond-list \)
\( = \) if \( n \simeq 0 \) then t
elseif ~ normal (mg-state) then t
elseif 'no-op-mg = car \( stmt \) then t
elseif 'signal-mg = car \( stmt \) then t
elseif 'prog2-mg = car \( stmt \) then
    meaning-induction-hint \( prog2-left-branch \( stmt \), proc-list, mg-state, n - 1 \)

\[\text{name-alist},\]
\[\text{r-cond-list}\]
\[\land \text{meaning-induction-hint (prog2-right-branch (stmt),}
\text{proc-list,}
\text{mg-meaning (prog2-left-branch (stmt),}
\text{proc-list,}
\text{mg-state,}
\text{n - 1),}
\text{n - 1,}
\text{name-alist,}
\text{r-cond-list)}\]
\text{elseif 'loop-mg = car (stmt) then meaning-induction-hint (loop-body (stmt),}
\text{proc-list,}
\text{mg-state,}
\text{n - 1,}
\text{name-alist,}
\text{cons ('leave, r-cond-list))}
\[\land \text{meaning-induction-hint (stmt,}
\text{proc-list,}
\text{mg-meaning (loop-body (stmt),}
\text{proc-list,}
\text{mg-state,}
\text{n - 1),}
\text{n - 1,}
\text{name-alist,}
\text{r-cond-list)}\]
\text{elseif 'if-mg = car (stmt) then meaning-induction-hint (if-false-branch (stmt),}
\text{proc-list,}
\text{mg-state,}
\text{n - 1,}
\text{name-alist,}
\text{r-cond-list)}\]
\[\land \text{meaning-induction-hint (if-true-branch (stmt),}
\text{proc-list,}
\text{mg-state,}
\text{n - 1,}
\text{name-alist,}
\text{r-cond-list)}\]
\text{elseif 'begin-mg = car (stmt) then meaning-induction-hint (begin-body (stmt),}
\text{proc-list,}
\text{mg-state,}
\text{n - 1,}
\text{name-alist,}
\text{r-cond-list)}\]
\[ n - 1, \]
\[ name-alist, \]
append (when-labels (stmt), r-cond-list))
∧ meaning-induction-hint (when-handler (stmt),)
\[ proc-list, \]
set-condition (mg-meaning (begin-body (stmt),
\[ proc-list, \]
\[ mg-state, \]
\[ n - 1), \]
\[ 'normal), \]
\[ n - 1, \]
\[ name-alist, \]
\[ r-cond-list) \]
\textbf{elseif} 'proc-call-mg = car (stmt)
\textbf{then} meaning-induction-hint (def-body (fetch-called-def (stmt, proc-list)),
\[ proc-list, \]
make-call-environment (mg-state, stmt,
\[ fetch-called-def (stmt, \]
\[ proc-list)), \]
\[ n - 1, \]
make-name-alist (fetch-called-def (stmt, proc-list)),
\[ make-cond-list (fetch-called-def (stmt, \]
\[ proc-list))), \]
\textbf{elseif} 'predefined-proc-call-mg = car (stmt) \textbf{then} t
\textbf{else endif}

**Event**: Disable ok-predefined-proc-call.

**Event**: Disable ok-predefined-proc-args.

**Event**: Disable ok-mg-statement.

**Event**: Disable simple-identifierp-implies-definedp.

**Event**: Disable int-identifierp-implies-definedp.

**Event**: Disable boolean-identifierp-implies-definedp.

**Event**: Disable character-identifierp-implies-definedp.
EVENT: Disable boolean-identifierp.

EVENT: Disable character-identifierp.

EVENT: Disable int-identifierp.

EVENT: Disable signatures-match-preserves-ok-mg-statement.

EVENT: Disable signatures-match-preserves-ok-predefined-proc-call.

EVENT: Disable signatures-match-preserves-ok-predefined-proc-args.

EVENT: Disable signatures-match-preserves-simple-identifierp.

EVENT: Disable not-member-listcars-not-assoc.

EVENT: Disable ok-mg-statep.

EVENT: Disable call-locals-alist-mg-alistp.

THEOREM: removing-condition-preserves-ok-mg-statep
\[(\text{ok-mg-statep}(\text{state}, \text{cons}(x, \text{lst})) \land (\text{cc}(\text{state}) \neq x)) \rightarrow \text{ok-mg-statep}(\text{state}, \text{lst})\]

EVENT: Disable removing-condition-preserves-ok-mg-statep.

THEOREM: adding-condition-preserves-ok-mg-statep
\[\text{ok-mg-statep}(\text{state}, \text{lst2}) \rightarrow \text{ok-mg-statep}(\text{state}, \text{append}(\text{lst}, \text{lst2}))\]

EVENT: Disable adding-condition-preserves-ok-mg-statep.

THEOREM: mg-meaning-preserves-ok-mg-statep
\[(\text{ok-mg-statep}(\text{mg-state}, \text{r-cond-list}) \land \text{ok-mg-statement}(\text{stmt}, \text{r-cond-list}, \text{name-alist}, \text{proc-list}) \land \text{signatures-match}(\text{mg-alist}(\text{mg-state}), \text{name-alist}) \land \text{ok-mg-def-plistp}(\text{proc-list})) \rightarrow \text{ok-mg-statep}(\text{mg-meaning}(\text{stmt}, \text{proc-list}, \text{mg-state}, n), \text{r-cond-list})\]

EVENT: Disable mg-meaning-preserves-ok-mg-statep.
THEOREM: mg-meaning-preserves-ok-cc
(\(\text{ok-mg-statep}(\text{mg-state}, \text{r-cond-list})\))
\(\land\) \(\text{ok-mg-statement}(\text{stmt}, \text{r-cond-list}, \text{name-alist}, \text{proc-list})\)
\(\land\) \(\text{signatures-match}(\text{mg-alist}(\text{mg-state}), \text{name-alist})\)
\(\land\) \(\text{ok-mg-def-plistp}(\text{proc-list})\)
\(\rightarrow\) \(\text{ok-cc}(\text{cc}(\text{mg-meaning}(\text{stmt}, \text{proc-list}, \text{mg-state}, n), \text{r-cond-list}))\)

EVENT: Disable mg-meaning-preserves-ok-cc.

THEOREM: mg-meaning-preserves-mg-alistp
(\(\text{ok-mg-statep}(\text{mg-state}, \text{r-cond-list})\))
\(\land\) \(\text{ok-mg-statement}(\text{stmt}, \text{r-cond-list}, \text{name-alist}, \text{proc-list})\)
\(\land\) \(\text{signatures-match}(\text{mg-alist}(\text{mg-state}), \text{name-alist})\)
\(\land\) \(\text{ok-mg-def-plistp}(\text{proc-list})\)
\(\rightarrow\) \(\text{mg-alistp}(\text{mg-alist}(\text{mg-meaning}(\text{stmt}, \text{proc-list}, \text{mg-state}, n)))\)

EVENT: Disable mg-meaning-preserves-mg-alistp.

THEOREM: mg-meaning-condition-member-cond-list1
(\(\text{ok-mg-statep}(\text{mg-state}, \text{r-cond-list})\))
\(\land\) \(\text{ok-mg-statement}(\text{stmt}, \text{r-cond-list}, \text{name-alist}, \text{proc-list})\)
\(\land\) \(\text{ok-mg-def-plistp}(\text{proc-list})\)
\(\land\) \(\text{signatures-match}(\text{mg-alist}(\text{mg-state}), \text{name-alist})\)
\(\land\) \((\neg\ \text{resource-errorp}(\text{mg-meaning}(\text{stmt}, \text{proc-list}, \text{mg-state}, n))))\)
\(\rightarrow\) \(\text{cc}(\text{mg-meaning}(\text{stmt}, \text{proc-list}, \text{mg-state}, n)))\)
\(\in\) \(\text{cons}(\text{‘normal}, \text{cons}(\text{‘routineerror}, \text{r-cond-list}))))\)

EVENT: Disable mg-meaning-condition-member-cond-list1.

EVENT: Make the library "c3".
### Index

<table>
<thead>
<tr>
<th>Condition</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>adding-condition-preserves-ok-mg-statep</td>
<td>66</td>
</tr>
<tr>
<td>adding-element-preserves-cond-identifier-plistp</td>
<td>10</td>
</tr>
<tr>
<td>all-cars-unique</td>
<td>24, 29</td>
</tr>
<tr>
<td>append-conditions-preserves-ok-mg-statep</td>
<td>63</td>
</tr>
<tr>
<td>array-elemtype</td>
<td>3, 5, 6, 16, 62</td>
</tr>
<tr>
<td>array-identifier-lengths-match</td>
<td>63</td>
</tr>
<tr>
<td>array-identifierp</td>
<td>7, 9, 15, 16, 52, 62, 63</td>
</tr>
<tr>
<td>array-identifierp-implies-definedp</td>
<td>9</td>
</tr>
<tr>
<td>array-identifiers-have-array-type s</td>
<td>62</td>
</tr>
<tr>
<td>array-length</td>
<td>3, 5, 6, 16, 35, 36, 39, 62, 63</td>
</tr>
<tr>
<td>array-literalp</td>
<td>6</td>
</tr>
<tr>
<td>array-mg-type-refp</td>
<td>5, 6, 62</td>
</tr>
<tr>
<td>assoc-def-ok1</td>
<td>24</td>
</tr>
<tr>
<td>begin-body</td>
<td>1, 18, 20, 21, 38, 44, 47, 64, 65</td>
</tr>
<tr>
<td>boolean-identifier-boolean-lite ral-value</td>
<td>59</td>
</tr>
<tr>
<td>boolean-identifierp</td>
<td>7, 8, 13–15, 18, 52, 59</td>
</tr>
<tr>
<td>boolean-identifierp-implies-definedp</td>
<td>8</td>
</tr>
<tr>
<td>boolean-identifierp-simple</td>
<td>8</td>
</tr>
<tr>
<td>boolean-literalp</td>
<td>4, 5, 59</td>
</tr>
<tr>
<td>boolean-literalp-tag-untag</td>
<td>59</td>
</tr>
<tr>
<td>c-size</td>
<td>41, 43, 45, 46, 48, 49</td>
</tr>
<tr>
<td>cadr-litatom-implies-definedp</td>
<td>54</td>
</tr>
<tr>
<td>call-actuals</td>
<td>1, 17, 24, 29, 30, 32–36, 42, 56</td>
</tr>
<tr>
<td>call-cond-lists-lengths-match</td>
<td>51</td>
</tr>
<tr>
<td>call-conds</td>
<td>1, 17, 29, 51</td>
</tr>
<tr>
<td>call-formal-signatures-match</td>
<td>54</td>
</tr>
<tr>
<td>call-formal-signatures-match2</td>
<td>54</td>
</tr>
<tr>
<td>call-local-signatures-match</td>
<td>54</td>
</tr>
<tr>
<td>call-local-signatures-match2</td>
<td>55</td>
</tr>
<tr>
<td>call-locals-alist-mg-alistp</td>
<td>54</td>
</tr>
<tr>
<td>call-name</td>
<td>1, 2, 17, 36, 42, 59–63</td>
</tr>
<tr>
<td>call-name-expansion</td>
<td>1</td>
</tr>
<tr>
<td>call-param-alist-mg-alistp</td>
<td>54</td>
</tr>
<tr>
<td>call-signatures-match1</td>
<td>55</td>
</tr>
<tr>
<td>call-signatures-match2</td>
<td>55</td>
</tr>
<tr>
<td>call-signatures-match3</td>
<td>55</td>
</tr>
<tr>
<td>called-def-formals-ok</td>
<td>24</td>
</tr>
<tr>
<td>called-def-ok</td>
<td>24</td>
</tr>
<tr>
<td>car-tag</td>
<td>59</td>
</tr>
<tr>
<td>cc, 25, 27, 29–31, 38, 44, 63, 66, 67</td>
<td></td>
</tr>
<tr>
<td>cc-set-condition</td>
<td>27</td>
</tr>
<tr>
<td>character-identifierp</td>
<td>7, 8, 52</td>
</tr>
<tr>
<td>character-identifierp-implies-definedp</td>
<td>8</td>
</tr>
<tr>
<td>character-identifierp-simple</td>
<td>8</td>
</tr>
<tr>
<td>character-literalp</td>
<td>5</td>
</tr>
<tr>
<td>collect-local-names</td>
<td>23</td>
</tr>
<tr>
<td>cond-identifier-conversion-litatom</td>
<td>57</td>
</tr>
<tr>
<td>cond-identifier-plistp</td>
<td>10, 17, 57</td>
</tr>
<tr>
<td>cond-identifier-plistp-cond-subsetp</td>
<td>10</td>
</tr>
<tr>
<td>cond-identifier-plistp-preserve s-membership</td>
<td>57</td>
</tr>
<tr>
<td>cond-identifierp</td>
<td>9, 10</td>
</tr>
<tr>
<td>cond-params-match</td>
<td>12, 17</td>
</tr>
<tr>
<td>cond-subsetp</td>
<td>10</td>
</tr>
<tr>
<td>cons-preserves-membership</td>
<td>57</td>
</tr>
<tr>
<td>cons-preserves-ok-mg-statep</td>
<td>27</td>
</tr>
<tr>
<td>convert-condition</td>
<td>28, 29</td>
</tr>
<tr>
<td>convert-condition-non-member</td>
<td>28</td>
</tr>
<tr>
<td>convert-condition1</td>
<td>28, 57</td>
</tr>
<tr>
<td>convert-condition1-membership</td>
<td>57</td>
</tr>
<tr>
<td>copy-out-params</td>
<td>29, 53, 56, 57</td>
</tr>
<tr>
<td>copy-out-params-preserves-mg-alistp</td>
<td>56</td>
</tr>
</tbody>
</table>
make-call-param-alist, 30, 54, 55
make-call-param-alist-plistp, 30
make-call-param-alist-preserves-listcars, 30
make-call-var-alist, 30, 31, 54, 55
make-cond-list, 22–24, 54, 58, 65
make-cond-list-legal-length, 23
make-name-alist, 22, 23, 55, 65
map-call-effects, 29, 30, 38, 45, 46, 50, 58
map-call-effects-preserves-mg-alistp, 56
map-call-effects-preserves-mg-psw, 45
map-call-effects-preserves-normal, 30
map-call-effects-preserves-ok-state, 57
map-call-effects-preserves-resource-error, 45
map-call-effects-preserves-route-error, 45
map-call-effects-preserves-route-errorp, 46
maxint, 4, 16
meaning-induction-hint, 63–65
meaning-induction-hint0, 46–48
mg-alist-elementp, 26
mg-alist-elements-have-ok-values, 26
mg-alist-member-mg-alist-elementp, 26
mg-alist-name-alistp, 26
mg-alist-set-condition, 27
mg-alistp, 26–29, 51, 54–57, 59, 62, 63, 67
mg-alistp-cdr, 26
mg-alistp-cons, 26
mg-alistp-distributes, 26
mg-alistp-distributes2, 26
mg-alistp-implies-mg-state, 27
mg-alistp-plistp, 26
mg-alistps-append, 51
mg-and-bool, 31, 35
mg-bool, 31–33, 59
mg-expression-falsep, 27, 38, 44
mg-max-ctrl-stk-size, 4, 43
mg-max-ctrl-stk-size-small-natural, 4
mg-max-temp-stk-size, 4, 43
mg-max-temp-stk-size-numberp, 4
mg-meaning, 37, 38, 45, 49, 50, 53, 56–58, 64–67
mg-meaning-condition-member-condition-member-condition-member-condition-member-condition-member-cond-list1, 67
mg-meaning-equivalence, 45
mg-meaning-equivalence2, 49
mg-meaning-equivalence3, 48
mg-meaning-equivalence4, 48
mg-meaning-mg-array-element-assignment, 36, 37
mg-meaning-mg-array-element-assignment-preserves-ok-mg-statep, 63
mg-meaning-mg-boolean-and, 35, 37
mg-meaning-mg-boolean-and-preserves-ok-mg-statep, 61
mg-meaning-mg-boolean-not, 35, 37
mg-meaning-mg-boolean-not-preserves-ok-mg-statep, 61
mg-meaning-mg-boolean-or, 34, 37
mg-meaning-mg-boolean-or-preserves-ok-mg-statep, 61
mg-meaning-mg-index-array, 35, 37
mg-meaning-mg-index-array-preserves-ok-mg-statep, 62
mg-meaning-mg-integer-add, 34, 37
mg-meaning-mg-integer-add-preserves-ok-mg-statep, 61
mg-meaning-mg-integer-le, 33, 37
mg-meaning-mg-integer-le-preserves-ok-mg-statep, 60
mg-meaning-mg-integer-subtract, 34, 37
mg-meaning-mg-integer-subtract-preserves-ok-mg-statep, 61
mg-meaning-mg-integer-unary-min
us, 33, 37
us-preserves-ok-mg-statep, 60
mg-meaning-mg-meaning-r-resource
  -error-equivalence, 50
mg-meaning-mg-simple-constant-assignment, 32, 36
  assignment-preserves-ok-mg-statep, 60
mg-meaning-mg-simple-constant-eq, 33, 36
  -preserves-ok-mg-statep, 60
mg-meaning-mg-simple-variable-assignment, 32, 36
  assignment-preserves-ok-mg-statep, 59
mg-meaning-mg-simple-variable-eq, 32, 36
  -preserves-ok-mg-statep, 59
mg-meaning-predefined-proc-call, 36, 38, 45, 53, 60–63
  -preserves-resource-error, 45
  -preserves-signatures-match, 53
mg-meaning-preserves-mg-alistp, 67
mg-meaning-preserves-ok-cc, 67
mg-meaning-preserves-ok-mg-statep, 66
mg-meaning-preserves-signatures-match, 53
mg-meaning-r, 43–51
mg-name-alist-elementp, 6
mg-name-alistp, 6, 26, 56
mg-not-bool, 31, 35, 59
mg-or-bool, 31, 34
mg-psw, 25, 27, 29, 31–36, 45
mg-state, 25, 27, 29, 31–36
mg-statep, 27
mg-type-refp, 5, 6, 21, 22, 26
mg-word-size, 4, 23, 33, 34, 59
minint, 4
more-resources-preserves-not-resource-errorp, 49
  source-errorp, 49
more-resources-preserves-resource-errorp0, 49
name, 22
new-value-mg-alist-elementp, 26
no-duplicates, 17, 23, 24, 55
nonempty-cond-identifier-plistp, 10, 18, 20
normal, 25, 30, 37, 43, 46, 50, 63
normal-not-in-cond-identifier-plistp, 10
not-normal-mg-meaning, 50
not-normal-mg-meaning-r, 50
ok-actual-params-list, 11, 17, 28, 52
ok-begin-expansion, 20
ok-begin-statement, 21
ok-cc, 27, 67
ok-condition, 10, 11, 18, 20
ok-condition-litatom, 11
ok-identifier-actual, 11
ok-if-statement, 20
ok-locals-plistp, 24
ok-loop-statement, 20
ok-mg-array-element-assignment-args, 16, 17
ok-mg-array-value, 6
ok-mg-boolean-and-args, 15, 17
ok-mg-boolean-not-args, 15, 17
ok-mg-boolean-or-args, 14–16
ok-mg-def, 23, 24, 56
ok-mg-def-plistp, 23–25, 54–57, 66, 67
ok-mg-def-plistp1, 23, 24
ok-mg-formal-data-param, 21
ok-mg-formal-data-params-plistp, 21, 23, 24, 54–56
ok-mg-index-array-args, 15, 17
ok-mg-integer-add-args, 14, 16
ok-mg-integer-add-args, 14, 16
ok-mg-integer-le-args, 13, 14, 16
ok-mg-integer-subtract-args, 14, 16
ok-mg-integer-unary-minus-args, 14, 16
ok-mg-local-data-decl, 22

71