EVENT: Start with the library "c-proc-call2".

;; I know that in the absence of resource-errors the two MG interpreters
;; and the two clocks are the same. Therefore, after I have proved everything
;; with the resource-error versions, I automatically gain the non-resource-error
;; versions.

THEOREM: not-resource-errorp-not-zero-p
(ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)
∧ (¬ resource-errorp (mg-meaning-r (stmt,
proc-list,
mg-state,
n,
list (length (temp-stk),
p-ctrl-stk-size (ctrl-stk))))))
→ (n ≠ 0)

(defun exact-time-induction-hint (cinfo r-cond-list t-cond-list stmt
proc-list mg-state n code2 subr ctrl-stk temp-stk name-alist)
  (if (zerop n)
   t
   (if (resources-inadequatep stmt proc-list
(list (length temp-stk) (p-ctrl-stk-size ctrl-stk)))
   t
   (if (equal (car stmt) 'no-op-mg)
     t
   (if (equal (car stmt) 'signal-mg)
     t
   (if (equal (car stmt) 'prog2-mg)
     (and (exact-time-induction-hint
    cinfo
r-cond-list
t-cond-list
1
(prog2-left-branch stmt)
proc-list
mg-state
(sub1 n)
(append (code (translate (nullify (translate (nullify cinfo)
  t-cond-list
  (prog2-left-branch stmt)
  proc-list))
  t-cond-list
  (prog2-right-branch stmt)
  proc-list))
code2)
subr ctrl-stk temp-stk name-alist)
  (exact-time-induction-hint
  (translate cinfo t-cond-list (prog2-left-branch stmt) proc-list)
  r-cond-list
t-cond-list
  (prog2-right-branch stmt)
proc-list
  (mg-meaning-r (prog2-left-branch stmt) proc-list mg-state (sub1 n)
  (list (length temp-stk) (p-ctrl-stk-size ctrl-stk)))
  (sub1 n)
code2
subr ctrl-stk temp-stk name-alist))
  (if (equal (car stmt) 'loop-mg)
    (and (exact-time-induction-hint
      (make-cinfo (append (code cinfo)
        (list (list 'dl (label-cnt cinfo) nil '(no-op))))
        (cons (cons 'leave (add1 (label-cnt cinfo)))
          (label-alist cinfo))
        (add1 (add1 (label-cnt cinfo)))))
      (cons 'leave r-cond-list) t-cond-list
    (loop-body stmt)
proc-list
mg-state
(sub1 n)
(cons (list 'jump (label-cnt cinfo))
  (cons (list 'dl (add1 (label-cnt cinfo)) nil '(push-constant (nat 2)))
    (cons '(pop-global c-c)
      code2)))
subr ctrl-stk temp-stk name-alist)
  (exact-time-induction-hint
  cinfo
    (cons 'leave r-cond-list) t-cond-list
  2
stmt proc-list
  (mg-meaning-r (loop-body stmt) proc-list mg-state (sub1 n))
  (list (length temp-stk) (p-ctrl-stk-size ctrl-stk)))
  (sub1 n)
  code2
  subr ctrl-stk temp-stk name-alist))
  (if (equal (car stmt) 'if-mg)
    (and (exact-time-induction-hint
    (make-cinfo
    (append (code cinfo)
    (list (list 'push-local (if-condition stmt))
    '(fetch-temp-stk)
    (list 'test-bool-and-jump 'false (label-cnt cinfo))))))
    (label-alist cinfo)
    (add1 (add1 (label-cnt cinfo))))
    r-cond-list t-cond-list
  (if-true-branch stmt)
    proc-list
    mg-state
    (sub1 n)
    (cons (list 'jump (add1 (label-cnt cinfo)))
    (cons (list 'dl (label-cnt cinfo) nil '(no-op))
    (append (code (translate
    (nullify
    (translate
    (make-cinfo
    nil
    (label-alist cinfo)
    (add1 (add1 (label-cnt cinfo))))
    t-cond-list
    (if-true-branch stmt)
    proc-list))
    t-cond-list
    (if-false-branch stmt)
    proc-list))
    (cons (list 'dl (add1 (label-cnt cinfo)) nil '(no-op)
    code2))))
  subr ctrl-stk temp-stk name-alist)
    (exact-time-induction-hint
    (add-code
    (translate
    (make-cinfo
    (append (code cinfo)
(list (list 'push-local (if-condition stmt))
  '(fetch-temp-stk)
  (list 'test-bool-and-jump 'false (label-cnt cinfo))))
(label-alist cinfo)
(add1 (add1 (label-cnt cinfo)))))
t-cond-list
(if-true-branch stmt)
proc-list)
(list (list 'jump (add1 (label-cnt cinfo)))
(list 'dl (label-cnt cinfo) nil '(no-op))))
r-cond-list t-cond-list
(if-false-branch stmt)
proc-list
mg-state
(sub1 n)
(cons (list 'dl (add1 (label-cnt cinfo)) nil '(no-op))
  code2)
subr ctrl-stk temp-stk name-alist))
(if (equal (car stmt) 'begin-mg)
  (and
    (exact-time-induction-hint
      (make-cinfo (code cinfo))
      (append (make-label-alist (when-labels stmt)
        (label-cnt cinfo))
        (label-alist cinfo))
      (add1 (add1 (label-cnt cinfo))))
      (append (when-labels stmt) r-cond-list)
      r-cond-list)
    t-cond-list)
(begin-body stmt)
proc-list
mg-state
(sub1 n)
(cons (list 'jump (add1 (label-cnt cinfo)))
  (cons (list 'dl (label-cnt cinfo) nil '(push-constant (nat 2)))
    (cons '(pop-global c-c)
      (append
        (code (translate
          (nullify
            (set-label-alist
              (translate
                (make-cinfo (code cinfo))
                (append (make-label-alist (when-labels stmt)
                  (label-cnt cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo))
                  (label-alist cinfo)}
(add1 (add1 (label-cnt cinfo)))
t-cond-list
(begin-body stmt)
proc-list)
(label-alist cinfo)))

  t-cond-list
  (when-handler stmt)
  proc-list)))
  (cons (list 'dl (add1 (label-cnt cinfo)) nil '(no-op))
    code2))))
subr ctrl-stk temp-stk name-alist)
  (exact-time-induction-hint
    (add-code
      (set-label-alist
        (translate
          (make-cinfo (code cinfo))
        (append (make-label-alist (when-labels stmt)
          (label-cnt cinfo))
          (label-alist cinfo))
        (add1 (add1 (label-cnt cinfo))))
      t-cond-list
      (begin-body stmt)
      proc-list)
      (label-alist cinfo))
    (list (list 'jump (add1 (label-cnt cinfo)))
      (list 'dl (label-cnt cinfo) nil '(push-constant (nat 2)))
      '(pop-global c-c)))
  r-cond-list
t-cond-list
(when-handler stmt) proc-list
(set-condition (mg-meaning-r (begin-body stmt) proc-list mg-state (sub1 n)
  (list (length temp-stk) (p-ctrl-stk-size ctrl-stk)))
  'normal)
(sub1 n)
(cons (list 'dl (add1 (label-cnt cinfo)) nil '(no-op))
  code2)
  subr ctrl-stk temp-stk name-alist))
  (if (equal (car stmt) 'proc-call-mg)
    (exact-time-induction-hint
      (make-cinfo nil
        (cons (cons 'routineerror 0)
          (make-label-alist (make-cond-list (fetch-called-def stmt proc-list)) 0))
      1)
    (make-cond-list (fetch-called-def stmt proc-list)))
(make-cond-list (fetch-called-def stmt proc-list))
  (def-body (fetch-called-def stmt proc-list))
proc-list
(make-call-environment mg-state stmt (fetch-called-def stmt proc-list))
(sub1 n)
(cons '(dl 0 nil (no-op))
  (cons (list 'pop* (data-length (def-locals (fetch-called-def stmt proc-list))))
    '((ret))))
(call-name stmt)
  (cons (p-frame
    (make-frame-alist (fetch-called-def stmt proc-list) stmt ctrl-stk temp-stk)
    (tag 'pc (cons subr
      (add1 (PLUS (LENGTH (CODE CINFO))
        (data-length (DEF-LOCALS (FETCH-CALLED-DEF
          STM PROC-LIST)))))
      (length (DEF-LOCALS (FETCH-CALLED-DEF
        STM PROC-LIST))))
      (LENGTH (CALL-ACTUALS STM)))))
    ctrl-stk)
  (append (reverse (mg-to-p-local-values (def-locals (fetch-called-def stmt proc-list)))
    (map-down-values (mg-alist mg-state)
      (bindings (top ctrl-stk))
      temp-stk)))
(make-name-alist (fetch-called-def stmt proc-list)))
  (if (equal (car stmt) 'predefined-proc-call-mg)
    t
    f))))))))
  ((lessp (COUNT n)))
  (INSTRUCTIONS (BASH (ENABLE WHEN-HANDLER BEGIN-BODY IF-TRUE-BRANCH
    IF-FALSE-BRANCH LOOP-BODY PROG2-RIGHT-BRANCH
    PROG2-LEFT-BRANCH))))

;; The cond-list is required in the translation so that
;; I can convert between MG and Piton conditions; in the recognizer it is only
;; required that I have some set of conditions which could be signalled. It had better
;; be that any signalled are on the translator cond-list or I won’t be able to do the
;; mapping. However, I can sometimes signal ‘leave and ‘routineerror even though
;; these are not on the translator cond-list. This is because their map functions are
;; computed independently of the list. Therefore the appropriate relation between the
;; recognizer-cond-alist and translator-cond-alist is that (cond-subsetp rec-list trans-list)

THEOREM: exact-time-lemma
(ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)

6
∧ ok-mg-def-plistp (proc-list)
∧ ok-translation-parameters (cinfo, t-cond-list, stmt, proc-list, code2)
∧ ok-mg-statep (mg-state, r-cond-list)
∧ cond-subsetp (r-cond-list, t-cond-list)
∧ (code (translate-def-body (assoc (subr, proc-list), proc-list))
   = append (code (translate (cinfo, t-cond-list, stmt, proc-list)),
             code2))
∧ user-defined-procp (subr, proc-list)
∧ plistp (temp-stk)
∧ listp (ctrl-stk)
∧ mg-vars-list-ok-in-p-state (mg-alist (mg-state),
                              bindings (top (ctrl-stk)),
                              temp-stk)
∧ no-p-aliasing (bindings (top (ctrl-stk)), mg-alist (mg-state))
∧ signatures-match (mg-alist (mg-state), name-alist)
∧ normal (mg-state)
∧ all-cars-unique (mg-alist (mg-state))
∧ (¬ resource-errorp (mg-meaning-r (stmt,
                              proc-list,
                              mg-state,
                              n,
                              list (length (temp-stk),
                              p-ctrl-stk-size (ctrl-stk))))))
→ (p (map-down (mg-state,
               proc-list,
               ctrl-stk,
               temp-stk,
               tag (’pc, cons (subr, length (code (cinfo))))),
               t-cond-list),
       clock (stmt, proc-list, mg-state, n))
  = p-state (tag (’pc,
               cons (subr,
               if normal (mg-meaning-r (stmt,
                              proc-list,
                              mg-state,
                              n,
                              list (length (temp-stk),
                              p-ctrl-stk-size (ctrl-stk))))
               then length (code (translate (cinfo,
                              t-cond-list,
                              stmt,
                              proc-list)))
               else find-label (fetch-label (cc (mg-meaning-r (stmt,
                              proc-list),
                              mg-state),
                              proc-list),
                              proc-list))}
EVENT: Disable exact-time-lemma.

THEOREM: exact-time-lemma2
(\text{ok-mg-statement} (\text{stmt}, r-cond-list, name-alist, proc-list) \\
\land \text{ok-mg-def-plistp} (proc-list) \\
\land \text{ok-translation-parameters} (cinfo, t-cond-list, stmt, proc-list, code2) \\
\land \text{ok-mg-statep} (mg-state, r-cond-list) \\
\land \text{cond-subsetp} (r-cond-list, t-cond-list)
\[\wedge (\text{code (translate-def-body (assoc (subr, proc-list), proc-list))} = \text{append (code (translate (cinfo, t-cond-list, stmt, proc-list)), code2)})\]
\[\wedge \text{user-defined-procp (subr, proc-list)}\]
\[\wedge \text{plistp (temp-stk)}\]
\[\wedge \text{listp (ctrl-stk)}\]
\[\wedge \text{mg-vars-list-ok-in-p-state (mg-alist (mg-state), bindings (top (ctrl-stk)), temp-stk)}\]
\[\wedge \text{no-p-aliasing (bindings (top (ctrl-stk)), mg-alist (mg-state))}\]
\[\wedge \text{signatures-match (mg-alist (mg-state), name-alist)}\]
\[\wedge \text{normal (mg-state)}\]
\[\wedge \text{all-cars-unique (mg-alist (mg-state))}\]
\[\wedge (\neg \text{resource-errorp (mg-meaning-r (stmt, proc-list, mg-state, n, list (length (temp-stk), p-ctrl-stk-size (ctrl-stk))))})\]
\[\wedge (\text{offset = length (code (cinfo))})\]
\[\rightarrow (\text{p (map-down (mg-state, proc-list, ctrl-stk, temp-stk, tag ('pc, cons (subr, offset)), t-cond-list), clock (stmt, proc-list, mg-state, n)) = p-state (tag ('pc, cons (subr, if normal (mg-meaning-r (stmt, proc-list, mg-state, n, list (length (temp-stk), p-ctrl-stk-size (ctrl-stk)))) then length (code (translate (cinfo, t-cond-list, stmt, proc-list))) else find-label (fetch-label (cc (mg-meaning-r (stmt, proc-list, mg-state, n, list (length (temp-stk), p-ctrl-stk-size (ctrl-stk))))))}})\]
p-ctrl-stk-size (ctrl-stk))),
label-alist (translate (cinfo, t-cond-list, stmt, proc-list)),
append (code (translate (cinfo, t-cond-list, stmt, proc-list)),
code2) endif),
ctrl-stk,
map-down-values (mg-alist (mg-meaning-r (stmt, proc-list, mg-state, n, list (length (temp-stk), p-ctrl-stk-size (ctrl-stk)))), bindings (top (ctrl-stk)), temp-stk),
translate-proc-list (proc-list), list (list ('c-c, mg-cond-to-p-nat (cc (mg-meaning-r (stmt, proc-list, mg-state, n, list (length (temp-stk), p-ctrl-stk-size (ctrl-stk))))), t-cond-list))),
MG-MAX-CTRL-STK-SIZE, MG-MAX-TEMP-STK-SIZE, MG-WORD-SIZE, 'run))

EVENT: Disable exact-time-lemma2.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; ;
;; ;
;; ;
;; ;
;; ;
;; ;
;; ;
;; ;
;; ;
;; ;
;; ;

;;;;;;;;;;;;;; ;
;; ;
;; ;
;; ;
;; ;
;; ;
;; ;
;; ;
;; ;
;; ;

;; Notice that exact-time-hyps refers to two cond-lists (which are related by ;; the function cond-subsetp) while translation-is-correct only refers to a ;; single list. The two lists are necessary in exact-time-hyps to make the
induction work. The user supplies only a single list with his input statement. That list must be an identifier-plistp and hence may not contain 'leave.

The hypotheses of this theorem guarantee the following:

1. mg-state is of the form <alist current-condition>
   a. the alist is of the form < <name type value> ... >
   b. the condition is 'normal, 'routineerror, 'timed-out, or is in cond-list
2. proc-list is a syntactically legal list of micro-gypsy procedures (which may be mutually recursive)
3. stmt is a syntactically legal micro-Gypsy statement with respect to proc-list with conditions from cond-list and variables from (mg-alist mg-state)
4. the translation environment described by cinfo is legitimate
   a. cinfo is of the form <code label-alist label-cnt cond-list>, where cond-list and code are proper lists
   b. cond-list is a proper list containing only legal identifiers, 'leave, or 'routineerror
   c. translation stmt with cinfo does not generate any labels which would duplicate labels in (code cinfo) or code2
5. subr is the name of some procedure in proc-list
6. the body of procedure subr is the context in which stmt "lives"; that is the translation of the body of subr is equal to the append of the
   the translation of stmt with code2
7. execution of stmt does not yield the condition timed-out.

THEOREM: cons-preserves-cond-subsetp
cond-subsetp \((y, z)\) \(\rightarrow\) cond-subsetp \((y, \text{cons}(x, z))\)

THEOREM: cond-subsetp-reflexive
ok-cond-list \((\text{cond-list})\) \(\rightarrow\) cond-subsetp \((\text{cond-list}, \text{cond-list})\)

THEOREM: mg-meaning-preserves-signatures-match3
(ok-ng-statement \((\text{stmt}, \text{cond-list}, \text{name-alist}, \text{proc-list})\)
  \(\wedge\) ok-ng-def-plistp \((\text{proc-list})\)
  \(\wedge\) ok-ng-statep \((\text{mg-state}, \text{cond-list})\)
  \(\wedge\) signatures-match \((\text{mg-alist} (\text{mg-state}), \text{name-alist})\)
  \(\rightarrow\) signatures-match \((\text{mg-alist} \text{mg-state}),\)
  \(\text{mg-alist} \text{mg-meaning} (\text{stmt, proc-list, mg-state, n}))\)

THEOREM: mg-state-decomposition
\(\neg \text{resource-errorp} (\text{state})\)
\(\rightarrow\) (mg-state \((\text{cc state}), \text{mg-alist (state)}, 'run = state)\)
THEOREM: signatures-match-implies-signatures-equal
signatures-match (alist1, alist2) → (signature (alist1) = signature (alist2))

EVENT: Disable signatures-match-implies-signatures-equal.

THEOREM: translation-is-correct2
(ok-mg-statement (stmt, cond-list, name-alist, proc-list)
  ∧ ok-mg-def-plistp (proc-list)
  ∧ ok-translation-parameters (cinfo, cond-list, stmt, proc-list, code2)
  ∧ ok-mg-statep (mg-state, cond-list)
  ∧ (code (translate-def-body (assoc (subr, proc-list), proc-list))
      = append (code (translate (cinfo, cond-list, stmt, proc-list)), code2))
  ∧ user-defined-procp (subr, proc-list)
  ∧ plistp (temp-stk)
  ∧ listp (ctrl-stk)
  ∧ mg-vars-list-ok-in-p-state (mg-alist (mg-state),
      bindings (top (ctrl-stk)),
      temp-stk)
  ∧ no-p-aliasing (bindings (top (ctrl-stk)), mg-alist (mg-state))
  ∧ signatures-match (mg-alist (mg-state), name-alist)
  ∧ all-cars-unique (mg-alist (mg-state))
  ∧ (¬ resource-errorp (mg-meaning-r (stmt,
      proc-list,
      mg-state,
      n,
      list (length (temp-stk),
      p-ctrl-stk-size (ctrl-stk)))))
  ∧ (pc-offset = length (code (cinfo))))
  ∧ (cc (mg-state) ≠ 'leave))
→ (map-up (p (map-down (mg-state,
      proc-list,
      ctrl-stk,
      temp-stk,
      tag ('pc, cons (subr, pc-offset)),
      cond-list),
      clock (stmt, proc-list, mg-state, n)),
    signature (mg-alist (mg-state)),
    cond-list)
    = mg-meaning (stmt, proc-list, mg-state, n))

;; The following few functions are for making the "initial story" about how you invoke the
;; MG compiler at the highest level. That is, we want to be able to compute the meaning of
;; an MG statement. To do so, I build a procedure around it and show by the lemma
This takes an alist from the mg-state and turns it into
a list of local-var-decls as might appear in a Micro-Gypsy
procedure.

Does this do anything; the alist is already in the right form.

**Definition:**

\[
\text{make-mg-locals-list (mg-alist)} \equiv
\begin{cases}
\text{nil} & \text{if mg-alist \( \approx \) nil} \\
\text{cons (list (name (car (mg-alist))), m-type (car (mg-alist)), m-value (car (mg-alist))), make-mg-locals-list (cdr (mg-alist)))} & \text{else}
\end{cases}
\]

**Theorem:** make-mg-local-list-preserves-listcars

\[
\text{listcars (make-mg-locals-list (lst))} = \text{listcars (lst)}
\]

**Theorem:** make-mg-locals-list-ok-mg-local-data-plistp

\[
\text{mg-alistp (mg-alist)} \rightarrow \text{ok-mg-local-data-plistp (make-mg-locals-list (mg-alist))}
\]

The point of this is to cons up a new user-defined-procedure
definition from the following components:
- alist: the Micro-Gypsy variable alist in which the statement is
to be interpreted.
- subr: a user-supplied name;
- stmt: the statement which we are interpreting
- cond-list: a list of conditions which we will allow
to be raised.

**Definition:**

\[
\text{make-mg-proc (alist, subr, stmt, cond-list)} \equiv
\text{list (subr, nil, cond-list, make-mg-locals-list (alist), nil, stmt)}
\]

Initial-temp-stk places the values of the mg-alist variables onto the temp-stk;
initial-bindings creates the Piton bindings corresponding to that initial
temp-stk.
\[\text{initial-temp-stk-reversed}(mg\text{-alist}) = \begin{cases} \text{nil} & \text{if } mg\text{-alist} \approx \text{nil} \\ \text{cons}(\text{mg-to-p-simple-literal}(\text{caddr}(\text{car}(mg\text{-alist}))), \text{initial-temp-stk-reversed}(\text{cdr}(mg\text{-alist}))) & \text{elseif } \text{simple-mg-type-refp}(\text{cadr}(\text{car}(mg\text{-alist}))) \\ \text{append}(\text{mg-to-p-simple-literal-list}(\text{caddr}(\text{car}(mg\text{-alist}))), \text{initial-temp-stk-reversed}(\text{cdr}(mg\text{-alist}))) & \text{else} \end{cases}\] 

**Theorem:** initial-temp-stk-reversed-plistp
plistp(initial-temp-stk-reversed(x))

**Definition:**
initial-temp-stk(mg\text{-alist}) = reverse(initial-temp-stk-reversed(mg\text{-alist}))

**Definition:**
initial-bindings(mg\text{-alist}, n) = \begin{cases} \text{nil} & \text{if } mg\text{-alist} \approx \text{nil} \\ \text{cons}(\text{cons}(\text{caar}(mg\text{-alist}), \text{tag}('\text{nat}, n)), \text{initial-bindings}(\text{cdr}(mg\text{-alist}), 1 + n)) & \text{elseif } \text{simple-mg-type-refp}(\text{cadr}(\text{car}(mg\text{-alist}))) \\ \text{cons}(\text{cons}(\text{caar}(mg\text{-alist}), \text{tag}('\text{nat}, n)), \text{initial-bindings}(\text{cdr}(mg\text{-alist}), n + \text{array-length}(\text{cadr}(\text{car}(mg\text{-alist})))))) & \text{else} \end{cases}\] 

**Theorem:** length-initial-bindings
length(initial-bindings(alist, n)) = length(alist)

;; 10/4/88 changing the final return pc from nil to (pc (subr . 0))
;; It is ignored but must be a legal pc value.

**Definition:**
map-down1(mg\text{-state}, proc-list, cond-list, subr, stmt)
= map-down(mg\text{-state},
\text{cons}(\text{make-mg-proc}(mg\text{-alist}(mg\text{-state}), subr, stmt, cond-list),
\text{proc-list}),
\text{list}(\text{cons}(\text{initial-bindings}(mg\text{-alist}(mg\text{-state}), 0),
\text{list}(\text{tag}('\text{pc}, \text{cons}(\text{subr, 0})))),
\text{initial-temp-stk}(mg\text{-alist}(mg\text{-state})),
\text{tag}('\text{pc}, \text{cons}(\text{subr, 0})),
\text{cond-list})

;; This theorem shows that I can compute the meaning of a statement if you will give me
;; the following and guarantee the following:
;; proc-list: must be a legitimate MG procedure list;
;;; cond-list: a list of conditions you will allow to be raised;
;;; mg-state: a legitimate MG state with current condition constrained by cond-list,
;;;     names on the list must all be unique;
;;; stmt: must be a legal statement with respect to proc-list and cond-list;
;;; n: the clock parameter saying how long to let the thing run;
;;; : an integer telling me the size of the implementation;
;;; subr: a litatom which is not the name of any procedure on proc-list;
;;;
;;; Supplied with these things, I can show you the meaning of a statement (provided that
;;; the computation does not run out of time or space.

;;; (defn ok-cond-list1 (cond-list)
;;;     (and (ok-cond-list cond-list)
;;;     (not (member 'leave cond-list)))

**Definition:**

\[
\text{new-proc-name} (x, \text{proc-list}) = (\text{ok-mg-namep} (x) \land \neg \text{defined-procp} (x, \text{proc-list}))
\]

**Theorem:** new-proc-doesnt-affect-fetch-called-def

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

**Theorem:** new-proc-doesnt-affect-mg-meaning-proc-call-case

\[
((n \neq 0)
\land \text{normal} (\text{mg-state})
\land (\text{car} (\text{stmt}) = '\text{proc-call-mg})
\land ((\text{ok-mg-def-plistp} (\text{proc-list})
\land \text{ok-mg-statement} (\text{def-body} (\text{fetch-called-def} (\text{stmt}, \text{proc-list})),
\quad \text{make-cond-list} (\text{fetch-called-def} (\text{stmt}, \text{proc-list})),
\quad \text{make-name-alist} (\text{fetch-called-def} (\text{stmt}, \text{proc-list})),
\quad \text{proc-list})
\land \text{new-proc-name} (\text{car} (\text{new-proc}), \text{proc-list}))
\rightarrow (\text{mg-meaning} (\text{def-body} (\text{fetch-called-def} (\text{stmt}, \text{proc-list})),
\quad \text{proc-list},
\quad \text{make-call-environment} (\text{mg-state},
\quad \text{stmt},
\quad \text{fetch-called-def} (\text{stmt}, \text{proc-list}))),\]

15
\[ n - 1 \]
\[ = \text{mg-meaning (def-body (fetch-called-def (stmt, proc-list)),} \]
\[ \quad \text{cons (new-proc, proc-list),} \]
\[ \quad \text{make-call-environment (mg-state,} \]
\[ \quad \quad \text{stmt,} \]
\[ \quad \quad \text{fetch-called-def (stmt,} \]
\[ \quad \quad \quad \text{proc-list)),} \]
\[ \quad n - 1)) \]
\[ \land \ \text{ok-mg-def-plistp (proc-list)} \]
\[ \land \ \text{ok-mg-statement (stmt, cond-list, name-alist, proc-list)} \]
\[ \land \ \text{new-proc-name (car (new-proc), proc-list))} \]
\[ \rightarrow \ (\text{mg-meaning (stmt, proc-list, mg-state, n)} \]
\[ \quad = \ \text{mg-meaning (stmt, cons (new-proc, proc-list), mg-state, n))} \]

**Definition:**

\[
\text{meaning-induction-hint3 (stmt,)}
\]
\[
\quad \text{proc-list,} \]
\[
\quad \text{mg-state,} \]
\[
\quad n, \]
\[
\quad \text{name-alist,} \]
\[
\quad \text{cond-list,} \]
\[
\quad \text{new-proc)} \]
\[
\begin{array}{l}
\text{= if } n \approx 0 \text{ then } t \\
\quad \text{elseif } \neg \text{normal (mg-state) then } t \\
\quad \text{elseif car (stmt) = 'no-op-mg then } t \\
\quad \text{elseif car (stmt) = 'signal-mg then } t \\
\quad \text{elseif car (stmt) = 'prog2-mg then } \text{meaning-induction-hint3 (prog2-left-branch (stmt),}
\quad \quad \text{proc-list,} \]
\[
\quad \quad \text{mg-state,} \]
\[
\quad \quad n - 1, \]
\[
\quad \quad \text{name-alist,} \]
\[
\quad \quad \text{cond-list,} \]
\[
\quad \quad \text{new-proc)} \]
\[
\land \ \text{meaning-induction-hint3 (prog2-right-branch (stmt),}
\quad \quad \text{proc-list,} \]
\[
\quad \quad \text{mg-meaning (prog2-left-branch (stmt),}
\quad \quad \quad \text{cons (new-proc,} \]
\[
\quad \quad \quad \quad \text{proc-list,} \]
\[
\quad \quad \quad \quad \text{mg-state,} \]
\[
\quad \quad \quad \quad n - 1),} \]
\[
\quad n - 1, \]
\[
\quad \text{name-alist,} \]
\[
\quad \text{cond-list,} \]
\end{array}
\]
\textbf{elseif} \texttt{car(stmt) = 'loop-mg} \textbf{then} \text{meaning-induction-hint3} (\text{loop-body (stmt)}, \\text{proc-list}, \\text{mg-state}, \\text{n - 1}, \\text{name-alist}, \\text{cons ('}leave\text{, cond-list)}, \\text{new-proc}) \\land \text{meaning-induction-hint3} (\text{stmt}, \\text{proc-list}, \\text{mg-meaning (loop-body (stmt)}, \\text{cons (new-proc, proc-list), mg-state}, \\text{n - 1}), \\text{n - 1}, \\text{name-alist}, \text{cond-list, new-proc})

\textbf{elseif} \texttt{car(stmt) = 'if-mg} \textbf{then} \text{meaning-induction-hint3} (\text{if-false-branch (stmt)}, \\text{proc-list}, \\text{mg-state}, \\text{n - 1}, \\text{name-alist}, \text{cond-list, new-proc}) \\land \text{meaning-induction-hint3} (\text{if-true-branch (stmt)}, \\text{proc-list}, \\text{mg-state}, \\text{n - 1}, \\text{name-alist}, \text{cond-list, new-proc})

\textbf{elseif} \texttt{car(stmt) = 'begin-mg} \textbf{then} \text{meaning-induction-hint3} (\text{begin-body (stmt)}, \\text{proc-list}, \\text{mg-state}, \\text{n - 1}, \\text{name-alist}, \text{cond-list, new-proc}) \\land \text{meaning-induction-hint3} (\text{when-handler (stmt)}, \text{new-proc})

\textbf{elseif} \texttt{car(stmt) = 'begin-mg} \textbf{then} \text{meaning-induction-hint3} (\text{begin-body (stmt)}, \\text{proc-list}, \\text{mg-state}, \\text{n - 1}, \\text{name-alist}, \text{append (when-labels (stmt), cond-list), new-proc}) \\land \text{meaning-induction-hint3} (\text{when-handler (stmt)}, \text{new-proc})
proc-list,
set-condition (mg-meaning (begin-body (stmt),
    cons (new-proc,
        proc-list),
    mg-state,
    n - 1),
    'normal),

    n - 1,
name-alist,
cond-list,
new-proc)

elseif car (stmt) = 'proc-call-mg
then meaning-induction-hint3 (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)),
new-proc)

elseif car (stmt) = 'predefined-proc-call-mg then t
else flendif

;; >>> For an automatic proof, these rules should be oriented in the other direction. Otherwise,
;;    they could cause looping.

THEOREM: new-proc-doesnt-affect-mg-meaning
(ok-mg-def-plistp (proc-list)
∧ ok-mg-statement (stmt, cond-list, name-alist, proc-list)
∧ new-proc-name (car (new-proc), proc-list))
→ (mg-meaning (stmt, proc-list, mg-state, n)
     = mg-meaning (stmt, cons (new-proc, proc-list), mg-state, n))

DEFINITION:
meaning-induction-hint4 (stmt,
    proc-list,
mg-state,
    n,
name-alist,
cond-list,
new-proc, sizes)

= if \( n \simeq 0 \) then \( \text{t} \)
  
elseif \( \text{normal(mg-state)} \) then \( \text{t} \)
  
elseif \( \text{resources-inadequatep(stmt, proc-list, sizes)} \) then \( \text{t} \)
  
elseif \( \text{car(stmt) = 'no-op-mg} \) then \( \text{t} \)
  
elseif \( \text{car(stmt) = 'signal-mg} \) then \( \text{t} \)
  
elseif \( \text{car(stmt) = 'prog2-mg} \) then
    meaning-induction-hint4(prog2-left-branch(stmt),
    proc-list, mg-state, \( n - 1 \),
    name-alist, cond-list, new-proc, sizes)
  
\( \land \) meaning-induction-hint4(prog2-right-branch(stmt),
    proc-list, mg-meaning-r(prog2-left-branch(stmt),
    cons(new-proc, proc-list),
    mg-state, \( n - 1 \),
    name-alist, cond-list, new-proc, sizes)

  
elseif \( \text{car(stmt) = 'loop-mg} \) then
    meaning-induction-hint4(loop-body(stmt),
    proc-list, mg-state, \( n - 1 \),
    name-alist, cons('leave, cond-list),
    new-proc, sizes)
  
\( \land \) meaning-induction-hint4(stmt,
    proc-list, mg-meaning-r(loop-body(stmt),
    cons(new-proc, proc-list),
    mg-state,
 elseif car(stmt) = 'if-mg
then meaning-induction-hint4 (if-false-branch (stmt),
proc-list,
mg-state,
n - 1,
name-alist,
cond-list,
new-proc,
sizes)
∧ meaning-induction-hint4 (if-true-branch (stmt),
proc-list,
mg-state,
n - 1,
name-alist,
cond-list,
new-proc,
sizes)
elseif car(stmt) = 'begin-mg
then meaning-induction-hint4 (begin-body (stmt),
proc-list,
mg-state,
n - 1,
name-alist,
cond-list,
new-proc,
sizes)
∧ meaning-induction-hint4 (when-handler (stmt),
proc-list,
set-condition (mg-meaning-r (begin-body (stmt),
cons (new-proc,
proc-list),
mg-state,
n - 1,
sizes),
'normal),
n - 1,
name-alist,
elseif car \( \text{stmt} \) = 'proc-call-mg
then

meaning-induction-hint4 (def-body (fetch-called-def \( \text{stmt} \), \( \text{proc-list} \)),
proc-list,
make-call-environment (mg-state, \( \text{stmt} \),
fetch-called-def \( \text{stmt} \), \( \text{proc-list} \)),
\( n - 1 \),
make-name-alist (fetch-called-def \( \text{stmt} \), \( \text{proc-list} \)),
make-cond-list (fetch-called-def \( \text{stmt} \), \( \text{proc-list} \)),
new-proc,
list (t-size \( \text{sizes} \) + data-length (def-locals (fetch-called-def \( \text{stmt} \), \( \text{proc-list} \))),
c-size \( \text{sizes} \) + (2
+ length (def-locals (fetch-called-def \( \text{stmt} \), \( \text{proc-list} \)))
+ length (def-formals (fetch-called-def \( \text{stmt} \), \( \text{proc-list} \)))))

elseif car \( \text{stmt} \) = 'predefined-proc-call-mg then t
else f endif

Theorem: new-proc-doesnt-affect-resources-inadequatep
(ok-mg-def-plistp \( \text{proc-list} \)
∧ ok-mg-statement \( \text{stmt} \), \( \text{cond-list} \), \( \text{name-alist} \), \( \text{proc-list} \))
∧ new-proc-name (car \( \text{new-proc} \), \( \text{proc-list} \))
→ (resources-inadequatep \( \text{stmt} \), cons \( \text{new-proc} \), \( \text{proc-list} \), \( \text{sizes} \))
= resources-inadequatep \( \text{stmt} \), \( \text{proc-list} \), \( \text{sizes} \))

Theorem: new-proc-doesnt-affect-mg-meaning-r
(ok-mg-def-plistp \( \text{proc-list} \)
∧ ok-mg-statement \( \text{stmt} \), \( \text{cond-list} \), \( \text{name-alist} \), \( \text{proc-list} \))
∧ new-proc-name (car \( \text{new-proc} \), \( \text{proc-list} \))
→ (mg-meaning-r \( \text{stmt} \), \( \text{proc-list} \), \( \text{mg-state} \), \( n \), \( \text{sizes} \))
= mg-meaning-r \( \text{stmt} \), cons \( \text{new-proc} \), \( \text{proc-list} \), \( \text{mg-state} \), \( n \), \( \text{sizes} \))

Theorem: new-proc-doesnt-affect-mg-meaning-r-2
(ok-mg-def-plistp \( \text{proc-list} \)
∧ ok-mg-statement \( \text{stmt} \), \( \text{cond-list} \), \( \text{name-alist} \), \( \text{proc-list} \))
\[\forall \text{ new-proc-name (car (new-proc, proc-list))} \rightarrow (\text{mg-meaning-r (stmt, cons (new-proc, proc-list), mg-state, n, sizes)} = \text{mg-meaning-r (stmt, proc-list, mg-state, n, sizes)})\]

**Theorem:** new-proc-doesnt-affect-clock-prog2-case

\[(n \neq 0) \land \text{normal (mg-state)} \land (\text{car (stmt) = 'prog2-mg}) \land ((\text{ok-mg-def-plistp (proc-list)} \land \text{ok-mg-statement (prog2-right-branch (stmt), cond-list, name-alist, proc-list)}) \land \text{new-proc-name (car (new-proc, proc-list))}) \rightarrow (\text{clock (prog2-right-branch (stmt), proc-list, mg-meaning (prog2-left-branch (stmt), cons (new-proc, proc-list), mg-state, n - 1)}) = \text{clock (prog2-right-branch (stmt), cons (new-proc, proc-list), mg-meaning (prog2-left-branch (stmt), cons (new-proc, proc-list), mg-state, n - 1)})}\]

\[\land ((\text{ok-mg-def-plistp (proc-list)} \land \text{ok-mg-statement (prog2-left-branch (stmt), cond-list, name-alist, proc-list)}) \land \text{new-proc-name (car (new-proc, proc-list))}) \rightarrow (\text{clock (prog2-left-branch (stmt), proc-list, mg-state, n - 1)}) = \text{clock (prog2-left-branch (stmt), cons (new-proc, proc-list), mg-state, n - 1)})\]

\[\land ((\text{ok-mg-def-plistp (proc-list)} \land \text{ok-mg-statement (stmt, cond-list, name-alist, proc-list)} \land \text{new-proc-name (car (new-proc, proc-list))}) \rightarrow (\text{clock (stmt, proc-list, mg-state, n)}) = \text{clock (stmt, cons (new-proc, proc-list), mg-state, n)})\]

22
THEOREM: new-proc-doesnt-affect-clock-loop-case
\((n \not\equiv 0)\)
\(\land\) normal \((mg-state)\)
\(\land\) \((\text{car} \, \text{stmt}) = \text{'loop-mg}\)
\(\land\) \((\text{ok-mg-def-plistp} \, \text{proc-list})\)
\(\land\) \((\text{ok-mg-statement} \, \text{stmt}, \text{cond-list}, \text{name-alist}, \text{proc-list})\)
\(\land\) \((\text{new-proc-name} \, \text{car} \, \text{(new-proc), proc-list})\)
\(\rightarrow\) \((\text{clock} \, \text{stmt}, \text{proc-list}, \text{mg-meaning} \, \text{loop-body} \, \text{stmt}, \text{cons} \, \text{(new-proc, proc-list)}, \text{mg-state}, n - 1)\) = \((\text{clock} \, \text{stmt}, \text{cons} \, \text{(new-proc, proc-list)}, \text{mg-meaning} \, \text{loop-body} \, \text{stmt}, \text{cons} \, \text{(new-proc, proc-list)}, \text{mg-state}, n - 1)\)

\(\land\) \((\text{ok-mg-def-plistp} \, \text{proc-list})\)
\(\land\) \((\text{ok-mg-statement} \, \text{stmt}, \text{cond-list}, \text{name-alist}, \text{proc-list})\)
\(\land\) \((\text{new-proc-name} \, \text{car} \, \text{(new-proc), proc-list})\)
\(\rightarrow\) \((\text{clock} \, \text{loop-body} \, \text{stmt}, \text{proc-list}, \text{mg-state}, n - 1)\) = \((\text{clock} \, \text{stmt}, \text{proc-list}, \text{mg-state}, n - 1)\)

THEOREM: new-proc-doesnt-affect-clock-if-case
\((n \not\equiv 0)\)
\(\land\) normal \((mg-state)\)
\(\land\) \((\text{car} \, \text{stmt}) = \text{'if-mg}\)
\(\land\) \((\text{ok-mg-def-plistp} \, \text{proc-list})\)
\(\land\) \((\text{ok-mg-statement} \, \text{if-true-branch} \, \text{stmt}, \text{cond-list})\)
\(\land\) \((\text{new-proc-name} \, \text{car} \, \text{(new-proc), proc-list})\)
\(\rightarrow\) \((\text{clock} \, \text{stmt}, \text{proc-list}, \text{mg-state}, \text{n})\) = \((\text{clock} \, \text{stmt}, \text{cons} \, \text{(new-proc, proc-list), mg-state, n})\)
\begin{align*}
\text{cond-list,} \\
\text{name-alist,} \\
\text{proc-list}) \\
\land \quad \text{new-proc-name (car (new-proc, proc-list))} \\
\to \quad (\text{clock (if-true-branch (stmt), proc-list, mg-state, n - 1)}) \\
\quad = \quad \text{clock (if-true-branch (stmt),} \\
\quad \quad \text{cons (new-proc, proc-list),} \\
\quad \quad \text{mg-state,} \\
\quad \quad n - 1))
\end{align*}

\textbf{Theorem:} new-proc-doesnt-affect-clock-begin-case

\((n \not\equiv 0) \land\)
\begin{align*}
\text{normal (mg-state)} \\
\land \quad (\text{car (stmt) = 'begin-mg)} \\
\land \quad ((\text{ok-mg-def-plistp (proc-list)} \\
\land \quad \text{ok-mg-statement (if-false-branch (stmt),} \\
\quad \text{cond-list,} \\
\quad \text{name-alist,} \\
\quad \text{proc-list)} \\
\land \quad \text{new-proc-name (car (new-proc, proc-list))} \\
\to \quad (\text{clock (stmt, proc-list, mg-state, n)}) \\
\quad = \quad \text{clock (stmt, cons (new-proc, proc-list), mg-state, n))}
\end{align*}

\textbf{Theorem:} new-proc-doesnt-affect-clock-begin-case

\((n \not\equiv 0) \land\)
\begin{align*}
\text{normal (mg-state)} \\
\land \quad (\text{car (stmt) = 'begin-mg)} \\
\land \quad ((\text{ok-mg-def-plistp (proc-list)} \\
\land \quad \text{ok-mg-statement (when-handler (stmt),} \\
\quad \text{cond-list,} \\
\quad \text{name-alist,} \\
\quad \text{proc-list)} \\
\land \quad \text{new-proc-name (car (new-proc, proc-list))} \\
\to \quad (\text{clock (when-handler (stmt),} \\
\quad \text{proc-list,} \\
\quad \text{set-condition (mg-meaning (begin-body (stmt),} \\
\quad \text{cons (new-proc, proc-list),} \\
\quad \text{mg-state,} \\
\quad \text{n - 1),} \\
\quad \quad \text{'normal),} \\
\quad \quad n - 1))}
\end{align*}
\[= \text{clock (when-
handler (stmt),} \]
\[\text{cons (new-proc, proc-list),} \]
\[\text{set-condition (mg-meaning (begin-body (stmt),} \]
\[\text{cons (new-proc,} \]
\[\text{proc-list),} \]
\[\text{mg-state,} \]
\[n - 1),} \]
\[\text{\textquote{normal},} \]
\[n - 1))} \]
\[\land \quad \text{(ok-mg-def-plistp (proc-list)} \]
\[\land \quad \text{ok-mg-statement (begin-body (stmt),} \]
\[\text{append (when-labels (stmt), cond-list),} \]
\[\text{name-alist,} \]
\[\text{proc-list)} \]
\[\land \quad \text{new-proc-name (car (new-proc), proc-list))} \]
\[\rightarrow \quad \text{(clock (begin-body (stmt), proc-list, mg-state, n - 1)} \]
\[= \quad \text{clock (begin-body (stmt),} \]
\[\text{cons (new-proc, proc-list),} \]
\[\text{mg-state,} \]
\[n - 1))} \]
\[\land \quad \text{ok-mg-def-plistp (proc-list)} \]
\[\land \quad \text{ok-mg-statement (stmt, cond-list, name-alist, proc-list)} \]
\[\land \quad \text{new-proc-name (car (new-proc), proc-list))} \]
\[\rightarrow \quad \text{(clock (stmt, proc-list, mg-state, n)} \]
\[= \quad \text{clock (stmt, cons (new-proc, proc-list), mg-state, n))} \]

**Theorem:** new-proc-doesnt-affect-clock-proc-call-case

\[((n \not\equiv 0) \]
\[\land \quad \text{normal (mg-state)} \]
\[\land \quad \text{(car stmt) = \textquote{proc-call-mg}} \]
\[\land \quad \text{((ok-mg-def-plistp (proc-list)} \]
\[\land \quad \text{ok-mg-statement (def-body (fetch-called-def (stmt, proc-list)),} \]
\[\text{make-cond-list (fetch-called-def (stmt, proc-list)),} \]
\[\text{make-name-alist (fetch-called-def (stmt, proc-list)),} \]
\[\text{proc-list)} \]
\[\land \quad \text{new-proc-name (car (new-proc), proc-list))} \]
\[\rightarrow \quad \text{(clock (def-body (fetch-called-def (stmt, proc-list)),} \]
\[\text{proc-list,} \]
\[\text{make-call-environment (mg-state,} \]
\[\text{stmt,} \]
\[\text{fetch-called-def (stmt, proc-list))},} \]
\[n - 1)\]
\[
\text{clock (def-body (fetch-called-def (stmt, proc-list)),}
\text{cons (new-proc, proc-list),}
\text{make-call-environment (mg-state,}
\text{stmt,}
\text{fetch-called-def (stmt,}
\text{proc-list)),}
\]
\[
\text{\( n - 1 \))}
\]
\[
\wedge \text{ok-mg-def-plistp (proc-list)}
\]
\[
\wedge \text{ok-mg-statement (stmt, cond-list, name-alist, proc-list)}
\]
\[
\wedge \text{new-proc-name (car (new-proc), proc-list)}
\]
\[
\rightarrow \text{ (clock (stmt, proc-list, mg-state, n)}
\]
\[
\quad \text{= clock (stmt, cons (new-proc, proc-list), mg-state, n))}
\]

**THEOREM: new-proc-doesnt-affect-clock**
\[
\text{(ok-mg-def-plistp (proc-list)}
\]
\[
\wedge \text{ok-mg-statement (stmt, cond-list, name-alist, proc-list)}
\]
\[
\wedge \text{new-proc-name (car (new-proc), proc-list)}
\]
\[
\rightarrow \text{ (clock (stmt, proc-list, mg-state, n)}
\]
\[
\quad \text{= clock (stmt, cons (new-proc, proc-list), mg-state, n))}
\]

**THEOREM: new-proc-doesnt-affect-ok-mg-statement-proc-call-case**
\[
\text{((car (stmt) = 'proc-call-mg)}
\]
\[
\wedge \text{ok-mg-def-plistp (proc-list)}
\]
\[
\wedge \text{ok-mg-statement (stmt, cond-list, name-alist, proc-list)}
\]
\[
\wedge \text{new-proc-name (car (new-proc), proc-list)}
\]
\[
\rightarrow \text{ ok-mg-statement (stmt, cond-list, name-alist, cons (new-proc, proc-list))}
\]

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

**THEOREM: new-proc-doesnt-affect-ok-mg-statement**
\[
\text{(ok-mg-def-plistp (proc-list)}
\]
\[
\wedge \text{ok-mg-statement (stmt, cond-list, name-alist, proc-list)}
\]
\[
\wedge \text{new-proc-name (car (new-proc), proc-list)}
\]
\[
\rightarrow \text{ ok-mg-statement (stmt, cond-list, name-alist, cons (new-proc, proc-list))}
\]

**THEOREM: new-proc-preserves-ok-mg-def**
\[
\text{(new-proc-name (car (new-proc), pl)}
\]
\[
\wedge \text{ok-mg-def-plistp (pl)}
\]
\[
\wedge \text{ok-mg-def (def, pl))}
\]
\[
\rightarrow \text{ ok-mg-def (def, cons (new-proc, pl))}
\]

**THEOREM: new-proc-preserves-ok-mg-def-plistp1**
\[
\text{(new-proc-name (car (new-proc), pl2)}
\]
\[
\wedge \text{ok-mg-def-plistp (pl2)}
\]
\[
\wedge \text{ok-mg-def-plistp1 (pl1, pl2))}
\]
\[
\rightarrow \text{ ok-mg-def-plistp1 (pl1, cons (new-proc, pl2))}
\]
Theorem: make-alist-make-locals-list-preserves-signatures-match
\[ \text{mg-alistp (mg-alist)} \rightarrow \text{signatures-match (mg-alist, make-alist-from-formals (make-mg-locals-list (mg-alist))}) \]

;; To do this one I needed to change the hyp the cond-list is a cond-identifierp-plistp to an identifier plistp.

Theorem: new-proc-preserves-ok-mg-def-plistp
\[ \text{ok-mg-statement (stmt, cond-list, mg-alist (mg-state), proc-list)} \wedge \text{ok-mg-def-plistp (proc-list)} \wedge \text{ok-mg-statep (mg-state, cond-list)} \wedge \text{identifier-plistp (cond-list)} \wedge \text{all-cars-unique (mg-alist (mg-state))} \wedge \text{new-proc-name (subr, proc-list)} \wedge (\text{length (cond-list)} < (((\exp (2, \text{MG-WORD-SIZE}) - 1) - 1) - 1)) \rightarrow \text{ok-mg-def-plistp (cons (make-mg-proc (mg-alist (mg-state), subr, stmt, cond-list), proc-list))} \]

Theorem: mg-to-p-simple-literal-list-listp
\[ \text{listp (mg-to-p-simple-literal-list (x)) = listp (x)} \]

Theorem: initial-temp-stk-reversed-listp
\[ \text{mg-alistp (x) \wedge listp (x)} \rightarrow \text{listp (initial-temp-stk-reversed (x))} \]

Theorem: length-initial-temp-stk-reversed
\[ \text{mg-alistp (alist)} \rightarrow (\text{length (initial-temp-stk-reversed (alist))} = \text{data-length (alist)}) \]

Definition:
\[ \text{initial-bindings-induction-hint (mg-alist, n, lst)} = \text{if mg-alist \simeq nil then t}\]
\[ \text{elseif simple-mg-type-refp (cadar (mg-alist))}\]
\[ \text{then initial-bindings-induction-hint (cdr (mg-alist), 1 + n, cons (mg-to-p-simple-literal (caddar (mg-alist)), lst))}\]
\[ \text{else initial-bindings-induction-hint (cdr (mg-alist), n + array-length (cadar (mg-alist)), append (reverse (mg-to-p-simple-literal-list (caddar (mg-alist))), lst)) endif} \]
THEOREM: initial-bindings-ok-in-initial-temp-stk1
\[\text{mg-alistp (mg-alist)} \land (n = \text{length (lst)}) \land \text{all-cars-unique (mg-alist)}\]
\[\rightarrow \text{mg-vars-list-ok-in-p-state (mg-alist, initial-bindings (mg-alist, n), append (initial-temp-stk (mg-alist), lst))}\]

THEOREM: initial-bindings-ok-in-initial-temp-stk
\[\text{ok-mg-statep (mg-state, cond-list)} \land \text{all-cars-unique (mg-alist (mg-state))}\]
\[\rightarrow \text{mg-vars-list-ok-in-p-state (mg-alist (mg-state), initial-bindings (mg-alist, 0), initial-temp-stk (mg-alist (mg-state)))}\]

THEOREM: initial-bindings-all-pointers-bigger
\[\text{all-cars-unique (alist)}\]
\[\rightarrow \text{all-pointers-bigger (collect-pointers (initial-bindings (alist, n, alist), n)}\]

THEOREM: no-p-aliasing-in-initial-bindings
\[\text{all-cars-unique (mg-alist)} \land \text{mg-alistp (mg-alist) \land (n \in \mathbb{N})}\]
\[\rightarrow \text{no-p-aliasing (initial-bindings (mg-alist, n, mg-alist)}\]

THEOREM: leave-not-state-cc
\[\text{ok-mg-statep (mg-state, cond-list)} \land \text{identifier-plistp (cond-list)}\]
\[\rightarrow (\text{cc (mg-state)} \neq \text{'leave)}\]

THEOREM: translation-is-correct3
\[\text{ok-mg-statement (stmt, cond-list, mg-alist (mg-state), proc-list)}\]
\[\land \text{ok-mg-def-plistp (proc-list)}\]
\[\land \text{ok-mg-statep (mg-state, cond-list)}\]
\[\land \text{identifier-plistp (cond-list)}\]
\[\land \text{all-cars-unique (mg-alist (mg-state))}\]
\[\land \text{new-proc-name (subr, proc-list)}\]
\[\land (\text{length (cond-list)} < (((\text{exp (2, MG-WORD-SIZE)} - 1) - 1) - 1))\]
\[\land (\neg \text{resource-errorp (mg-meaning-r (stmt, proc-list, mg-state, n, list (length (initial-temp-stk (mg-alist (mg-state)))), p-ctrl-stk-size (list (cons (initial-bindings (mg-alist (mg-state), 0), list (tag ('pc, cons (subr, 0))))))))})\]
\[\rightarrow (\text{map-up (p (map-down1 (mg-state, proc-list, cond-list, subr, stmt), clock (stmt, proc-list, mg-state, n)), 28})\]
signature (mg-alist (mg-state)),
cond-list
= mg-meaning (stmt, proc-list, mg-state, n))

;; This is just a slightly cleaned up version of the previous lemma.

THEOREM: translation-is-correct4
(ok-mg-statement (stmt, cond-list, mg-alist (mg-state), proc-list)
∧ ok-mg-def-plistp (proc-list)
∧ ok-mg-statep (mg-state, cond-list)
∧ identifier-plistp (cond-list)
∧ all-cars-unique (mg-alist (mg-state))
∧ new-proc-name (subr, proc-list)
∧ (length (cond-list) < (((exp (2, MG-WORD-SIZE) − 1) − 1) − 1))
∧ (¬ resource-errorp (mg-meaning-r (stmt,
proc-list,
mg-state,
stmt, proc-list, mg-state, n),
list (data-length (mg-alist (mg-state)),
2 + length (mg-alist (mg-state))))))
→ (map-up (p (map-down1 (mg-state, proc-list, cond-list, subr, stmt),
clock (stmt, proc-list, mg-state, n)),
signature (mg-alist (mg-state)),
cond-list)
= mg-meaning (stmt, proc-list, mg-state, n))

DEFINITION:
ok-execution-environment (stmt, cond-list, proc-list, mg-state, subr, n)
= (ok-mg-statement (stmt, cond-list, mg-alist (mg-state), proc-list)
∧ ok-mg-def-plistp (proc-list)
∧ ok-mg-statep (mg-state, cond-list)
∧ identifier-plistp (cond-list)
∧ all-cars-unique (mg-alist (mg-state))
∧ new-proc-name (subr, proc-list)
∧ (length (cond-list) < (((exp (2, MG-WORD-SIZE) − 1) − 1) − 1)))

THEOREM: translation-is-correct5
(ok-execution-environment (stmt, cond-list, proc-list, mg-state, subr, n)
∧ (¬ resource-errorp (mg-meaning-r (stmt,
proc-list,
mg-state,
stmt, proc-list, mg-state, n),
list (data-length (mg-alist (mg-state)),
2 + length (mg-alist (mg-state)))))))
\[ \text{Event: Make the library "ca10".} \]
Index

all-cars-unique, 7, 9, 12, 27–29
all-pointers-bigger, 28
array-length, 14, 27

begin-body, 17, 18, 20, 24, 25
bindings, 7–10, 12

c-size, 21
cc, 8, 10–12, 28
clock, 7, 9, 12, 22–26, 28–30
code, 7–10, 12
collect-pointers, 28
cond-subsetp, 7, 8, 11
cond-subsetp-reflexive, 11
cons-preserves-cond-subsetp, 11

data-length, 21, 27, 29
def-body, 15, 16, 18, 21, 25, 26
def-formals, 21
def-locals, 21
defined-procp, 15

exact-time-lemma, 6
exact-time-lemma2, 8
exp, 27–29

fetch-called-def, 15, 16, 18, 21, 25, 26
fetch-label, 8, 10
find-label, 8, 10

identifier-plistp, 27–29
if-false-branch, 17, 20, 24
if-true-branch, 17, 20, 23, 24
initial-bindings, 14, 28
initial-bindings-all-pointers-bigger, 28
initial-bindings-induction-hint, 27
initial-bindings-ok-in-initial-temp-stk, 28

initial-temp-stk, 14, 28
initial-temp-stk-reversed, 13, 14, 27
initial-temp-stk-reversed-listp, 27
initial-temp-stk-reversed-plistp, 14

label-alist, 8, 10
leave-not-state-cc, 28
length, 1, 7–10, 12, 14, 21, 27–29
length-initial-bindings, 14
length-initial-temp-stk-reverse

d, 27
listcars, 13
loop-body, 17, 19, 23

m-type, 13
m-value, 13
make-alist-from-formals, 27
make-alist-make-locals-list-preserves-signatures-match, 27
make-call-environment, 15, 16, 18, 21, 25, 26
make-cond-list, 15, 18, 21, 25
make-mg-local-list-preserves-listcars, 13

make-mg-locals-list, 13, 27
make-mg-locals-list-ok-mg-local-data-plistp, 13
make-mg-proc, 13, 14, 27
make-name-alist, 15, 18, 21, 25
map-down, 7, 9, 12, 14
map-down-values, 8, 10
map-down1, 14, 28–30
map-up, 12, 29, 30

meaning-induction-hint3, 16–18
meaning-induction-hint4, 18–21
mg-alist, 7–12, 14, 27–30
mg-alistp, 13, 27, 28
mg-cond-to-p-nat, 8, 10

mg-max-ctrl-stk-size, 8, 10
mg-max-temp-stk-size, 8, 10
mg-meaning, 11, 12, 16–18, 22–25, 29, 30
mg-meaning-preserves-signatures
-match3, 11
mg-meaning-r, 1, 7–10, 12, 19–22, 28, 29
mg-state, 11
mg-state-decomposition, 11
mg-to-p-simple-literal, 14, 27
mg-to-p-simple-literal-list, 14, 27
mg-to-p-simple-literal-listp, 27
mg-vars-list-ok-in-p-state, 7, 9, 12, 28
mg-word-size, 8, 10, 27–29
name, 13
new-proc-doesnt-affect-clock, 26
new-proc-doesnt-affect-clock-begin-case, 24
new-proc-doesnt-affect-clock-if-case, 23
new-proc-doesnt-affect-clock-loop-case, 23
new-proc-doesnt-affect-clock-proc-call-case, 25
og2-case, 22
new-proc-doesnt-affect-fetch-called-def, 15
new-proc-doesnt-affect-mg-meaning, 18
mg-proc-call-case, 15
mg-r, 21
mg-r-2, 21
new-proc-doesnt-affect-ok-mg-statement, 26
atement-proc-call-case, 26
new-proc-doesnt-affect-resources-inadequatep, 21
new-proc-name, 15, 16, 18, 21–29
new-proc-preserves-ok-mg-def, 26
new-proc-preserves-ok-mg-def-plstp, 27
st1, 26
no-p-aliasing, 7, 9, 12, 28
no-p-aliasing-in-initial-bindings, 28
normal, 7, 9, 15, 16, 19, 22–25
not-resource-errorp-not-zerop-n, 1
ok-cond-list, 11
ok-execution-environment, 29
ok-mg-def, 26
ok-mg-def-plistp, 7, 8, 11, 12, 15, 16, 18, 21–29
ok-mg-def-plisp1, 26
ok-mg-local-data-plistp, 13
ok-mg-namep, 15
ok-mg-statement, 1, 6, 8, 11, 12, 15, 16, 18, 21–29
ok-mg-statep, 7, 8, 11, 12, 27–29
ok-translation-parameters, 7, 8, 12
p, 7, 9, 12, 28–30
p-ctrl-stk-size, 1, 7–10, 12, 28
p-state, 8, 10
plistp, 7, 9, 12, 14
prog2-left-branch, 16, 19, 22
prog2-right-branch, 16, 19, 22
resource-errorp, 1, 7, 9, 11, 12, 28, 29
resources-inadequatep, 19, 21
reverse, 14, 27
set-condition, 18, 20, 24, 25
signature, 12, 29, 30
signatures-match, 7, 9, 11, 12, 27
signatures-match-implies-signatures-equal, 12
simple-mg-type-refp, 14, 27
t-size, 21
tag, 7–10, 12, 14, 28
top, 7–10, 12
translate, 7–10, 12
translate-def-body, 7, 9, 12
translate-proc-list, 8, 10
translation-is-correct2, 12
translation-is-correct3, 28
translation-is-correct4, 29
translation-is-correct5, 29
user-defined-proc, 7, 9, 12

when-handler, 17, 20, 24, 25
when-labels, 17, 20, 25