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.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                           ;;
;;                              EXACT TIME LEMMA                             ;;
;;                                                                           ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: not-resource-errorp-not-zerop-n

(ok-mg-statement (*stmt*, *r-cond-list*, *name-alist*, *proc-list*)
 $\wedge$ ($\neg$ resource-errorp (mg-meaning-r (*stmt*,
proc-list,
mg-state,
n,
list (length (*temp-stk*),
p-ctrl-stk-size (*ctrl-stk*))))))
$\rightarrow$ ($n \not\simeq 0$)

```
(defn 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
```

```
  (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
```

```
  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)
                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))
```

```
     (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
       STMT PROC-LIST)))
   (length (DEF-LOCALS (FETCH-CALLED-DEF
STMT PROC-LIST)))
   (LENGTH (CALL-ACTUALS STMT)))))))
       ctrl-stk)
                 (append (reverse (mg-to-p-local-values (def-locals (fetch-called-def stmt p
 (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-lis
```

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

$\wedge$ ok-mg-def-plistp (*proc-list*)
$\wedge$ ok-translation-parameters (*cinfo*, *t-cond-list*, *stmt*, *proc-list*, *code2*)
$\wedge$ ok-mg-statep (*mg-state*, *r-cond-list*)
$\wedge$ cond-subsetp (*r-cond-list*, *t-cond-list*)
$\wedge$ (code (translate-def-body (assoc (*subr*, *proc-list*), *proc-list*))
= append (code (translate (*cinfo*, *t-cond-list*, *stmt*, *proc-list*)),
code2))
$\wedge$ user-defined-procp (*subr*, *proc-list*)
$\wedge$ plistp (*temp-stk*)
$\wedge$ listp (*ctrl-stk*)
$\wedge$ mg-vars-list-ok-in-p-state (mg-alist (*mg-state*),
bindings (top (*ctrl-stk*)),
temp-stk)
$\wedge$ no-p-aliasing (bindings (top (*ctrl-stk*)), mg-alist (*mg-state*))
$\wedge$ signatures-match (mg-alist (*mg-state*), *name-alist*)
$\wedge$ normal (*mg-state*)
$\wedge$ all-cars-unique (mg-alist (*mg-state*))
$\wedge$ ($\neg$ resource-errorp (mg-meaning-r (*stmt*,
proc-list,
mg-state,
n,
list (length (*temp-stk*),
p-ctrl-stk-size (*ctrl-stk*))))))
$\rightarrow$ (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,
n,
list (length (*temp-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-lemma.

THEOREM: exact-time-lemma2
(ok-mg-statement (*stmt*, *r-cond-list*, *name-alist*, *proc-list*)
∧ 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*)

$$
\begin{array}{ll}
\wedge & (\text{code}\,(\text{translate-def-body}\,(\text{assoc}\,(\mathit{subr},\ \mathit{proc\text{-}list}),\ \mathit{proc\text{-}list})) \\
& \quad = \quad \text{append}\,(\text{code}\,(\text{translate}\,(\mathit{cinfo},\ \mathit{t\text{-}cond\text{-}list},\ \mathit{stmt},\ \mathit{proc\text{-}list})), \\
& \qquad\qquad \mathit{code2})) \\
\wedge & \text{user-defined-procp}\,(\mathit{subr},\ \mathit{proc\text{-}list}) \\
\wedge & \text{plistp}\,(\mathit{temp\text{-}stk}) \\
\wedge & \text{listp}\,(\mathit{ctrl\text{-}stk}) \\
\wedge & \text{mg-vars-list-ok-in-p-state}\,(\text{mg-alist}\,(\mathit{mg\text{-}state}), \\
& \qquad\qquad\qquad\qquad\qquad \text{bindings}\,(\text{top}\,(\mathit{ctrl\text{-}stk})), \\
& \qquad\qquad\qquad\qquad\qquad \mathit{temp\text{-}stk}) \\
\wedge & \text{no-p-aliasing}\,(\text{bindings}\,(\text{top}\,(\mathit{ctrl\text{-}stk})),\ \text{mg-alist}\,(\mathit{mg\text{-}state})) \\
\wedge & \text{signatures-match}\,(\text{mg-alist}\,(\mathit{mg\text{-}state}),\ \mathit{name\text{-}alist}) \\
\wedge & \text{normal}\,(\mathit{mg\text{-}state}) \\
\wedge & \text{all-cars-unique}\,(\text{mg-alist}\,(\mathit{mg\text{-}state})) \\
\wedge & (\neg\ \text{resource-errorp}\,(\text{mg-meaning-r}\,(\mathit{stmt}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{proc\text{-}list}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{mg\text{-}state}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad n, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{list}\,(\text{length}\,(\mathit{temp\text{-}stk}), \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{p-ctrl-stk-size}\,(\mathit{ctrl\text{-}stk}))))) \\
\wedge & (\mathit{offset} = \text{length}\,(\text{code}\,(\mathit{cinfo})))) \\
\rightarrow & (\text{p}\,(\text{map-down}\,(\mathit{mg\text{-}state}, \\
& \qquad\qquad\qquad\qquad\quad \mathit{proc\text{-}list}, \\
& \qquad\qquad\qquad\qquad\quad \mathit{ctrl\text{-}stk}, \\
& \qquad\qquad\qquad\qquad\quad \mathit{temp\text{-}stk}, \\
& \qquad\qquad\qquad\qquad\quad \text{tag}\,(\texttt{'pc},\ \text{cons}\,(\mathit{subr},\ \mathit{offset})), \\
& \qquad\qquad\qquad\qquad\quad \mathit{t\text{-}cond\text{-}list}), \\
& \qquad \text{clock}\,(\mathit{stmt},\ \mathit{proc\text{-}list},\ \mathit{mg\text{-}state},\ n)) \\
& \quad = \quad \text{p-state}\,(\text{tag}\,(\texttt{'pc}, \\
& \qquad\qquad\qquad\qquad\quad \text{cons}\,(\mathit{subr}, \\
& \qquad\qquad\qquad\qquad\qquad\quad \textbf{if}\ \text{normal}\,(\text{mg-meaning-r}\,(\mathit{stmt}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{proc\text{-}list}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{mg\text{-}state}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad n, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{list}\,(\text{length}\,(\mathit{temp\text{-}stk}), \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{p-ctrl-stk-size}\,(\mathit{ctrl\text{-}stk})))) \\
& \qquad\qquad\qquad\qquad\qquad\quad \textbf{then}\ \text{length}\,(\text{code}\,(\text{translate}\,(\mathit{cinfo}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{t\text{-}cond\text{-}list}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{stmt}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{proc\text{-}list}))) \\
& \qquad\qquad\qquad\qquad\qquad\quad \textbf{else}\ \text{find-label}\,(\text{fetch-label}\,(\text{cc}\,(\text{mg-meaning-r}\,(\mathit{stmt}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{proc\text{-}list}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{mg\text{-}state}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad n, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{list}\,(\text{length}\,(\mathit{temp\text{-}stk}),
\end{array}
$$

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.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                              ;;
;;                          TRANSLATION CORRECTNESS                             ;;
;;                                                                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; 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) → cond-subsetp (y, cons (x, z))

THEOREM: cond-subsetp-reflexive
ok-cond-list (*cond-list*) → cond-subsetp (*cond-list*, *cond-list*)

THEOREM: mg-meaning-preserves-signatures-match3
(ok-mg-statement (*stmt*, *cond-list*, *name-alist*, *proc-list*)
 ∧ ok-mg-def-plistp (*proc-list*)
 ∧ ok-mg-statep (*mg-state*, *cond-list*)
 ∧ signatures-match (mg-alist (*mg-state*), *name-alist*))
→ signatures-match (mg-alist (*mg-state*),
                    mg-alist (mg-meaning (*stmt*, *proc-list*, *mg-state*, n)))

THEOREM: mg-state-decomposition
(¬ resource-errorp (*state*))
→ (mg-state (cc (*state*), 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
```

```
;; translation-is-correct2 that the meaning of the statement is the execution of that
;; procedure.
;;
;; 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:
make-mg-locals-list (*mg-alist*)
= **if** *mg-alist* $\simeq$ **nil then nil**
else cons (list (name (car (*mg-alist*)),
m-type (car (*mg-alist*)),
m-value (car (*mg-alist*))),
make-mg-locals-list (cdr (*mg-alist*))) **endif**

THEOREM: make-mg-local-list-preserves-listcars
listcars (make-mg-locals-list (*lst*)) = listcars (*lst*)

THEOREM: make-mg-locals-list-ok-mg-local-data-plistp
mg-alistp (*mg-alist*)
→ 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:
make-mg-proc (*alist*, *subr*, *stmt*, *cond-list*)
= 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.
```

DEFINITION:

initial-temp-stk-reversed (*mg-alist*)
= **if** *mg-alist* ≃ **nil then nil**
elseif simple-mg-type-refp (cadr (car (*mg-alist*)))
then cons (mg-to-p-simple-literal (caddr (car (*mg-alist*))),
initial-temp-stk-reversed (cdr (*mg-alist*)))
else append (mg-to-p-simple-literal-list (caddr (car (*mg-alist*))),
initial-temp-stk-reversed (cdr (*mg-alist*))) **endif**

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

DEFINITION:
initial-temp-stk (*mg-alist*) = reverse (initial-temp-stk-reversed (*mg-alist*))

DEFINITION:
initial-bindings (*mg-alist*, *n*)
= **if** *mg-alist* ≃ **nil then nil**
elseif simple-mg-type-refp (cadr (car (*mg-alist*)))
then cons (cons (caar (*mg-alist*), tag (`'nat`, *n*)),
initial-bindings (cdr (*mg-alist*), 1 + *n*))
else cons (cons (caar (*mg-alist*), tag (`'nat`, *n*)),
initial-bindings (cdr (*mg-alist*),
n + array-length (cadr (car (*mg-alist*))))) **endif**

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-state*, *proc-list*, *cond-list*, *subr*, *stmt*)
= map-down (*mg-state*,
cons (make-mg-proc (mg-alist (*mg-state*), *subr*, *stmt*, *cond-list*),
proc-list),
list (cons (initial-bindings (mg-alist (*mg-state*), `0`),
list (tag (`'pc`, cons (*subr*, `0`))))),
initial-temp-stk (mg-alist (*mg-state*)),
tag (`'pc`, cons (*subr*, `0`)),
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 alist 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:
new-proc-name (x, *proc-list*)
= (ok-mg-namep (x) $\wedge$ ($\neg$ defined-procp (x, *proc-list*)))

THEOREM: new-proc-doesnt-affect-fetch-called-def
((car (*stmt*) = 'proc-call-mg)
$\wedge$ new-proc-name (car (*new-proc*), *proc-list*)
$\wedge$ ok-mg-statement (*stmt*, *cond-list*, *name-alist*, *proc-list*))
$\rightarrow$ (fetch-called-def (*stmt*, cons (*new-proc*, *proc-list*))
= fetch-called-def (*stmt*, *proc-list*))

THEOREM: new-proc-doesnt-affect-mg-meaning-proc-call-case
(($n \not\simeq$ 0)
$\wedge$ normal (*mg-state*)
$\wedge$ (car (*stmt*) = 'proc-call-mg)
$\wedge$ ((ok-mg-def-plistp (*proc-list*)
$\wedge$ ok-mg-statement (def-body (fetch-called-def (*stmt*, *proc-list*)),
make-cond-list (fetch-called-def (*stmt*,
proc-list)),
make-name-alist (fetch-called-def (*stmt*,
proc-list)),
proc-list)
$\wedge$ new-proc-name (car (*new-proc*), *proc-list*))
$\rightarrow$ (mg-meaning (def-body (fetch-called-def (*stmt*, *proc-list*)),
proc-list,
make-call-environment (*mg-state*,
stmt,
fetch-called-def (*stmt*,
proc-list)),

$$
\begin{array}{ll}
 & n-1) \\
= & \text{mg-meaning}\,(\text{def-body}\,(\text{fetch-called-def}\,(\textit{stmt},\ \textit{proc-list})), \\
 & \qquad \text{cons}\,(\textit{new-proc},\ \textit{proc-list}), \\
 & \qquad \text{make-call-environment}\,(\textit{mg-state}, \\
 & \qquad\qquad \textit{stmt}, \\
 & \qquad\qquad \text{fetch-called-def}\,(\textit{stmt}, \\
 & \qquad\qquad\qquad \textit{proc-list})), \\
 & \qquad n-1))) \\
\wedge & \text{ok-mg-def-plistp}\,(\textit{proc-list}) \\
\wedge & \text{ok-mg-statement}\,(\textit{stmt},\ \textit{cond-list},\ \textit{name-alist},\ \textit{proc-list}) \\
\wedge & \text{new-proc-name}\,(\text{car}\,(\textit{new-proc}),\ \textit{proc-list})) \\
\rightarrow & (\text{mg-meaning}\,(\textit{stmt},\ \textit{proc-list},\ \textit{mg-state},\ n) \\
 & \quad = \text{mg-meaning}\,(\textit{stmt},\ \text{cons}\,(\textit{new-proc},\ \textit{proc-list}),\ \textit{mg-state},\ n))
\end{array}
$$

DEFINITION:

$$
\begin{array}{ll}
 & \text{meaning-induction-hint3}\,(\textit{stmt}, \\
 & \qquad \textit{proc-list}, \\
 & \qquad \textit{mg-state}, \\
 & \qquad n, \\
 & \qquad \textit{name-alist}, \\
 & \qquad \textit{cond-list}, \\
 & \qquad \textit{new-proc}) \\
= & \quad \textbf{if}\ n \simeq \texttt{0}\ \textbf{then t} \\
 & \quad \textbf{elseif}\ \neg\ \text{normal}\,(\textit{mg-state})\ \textbf{then t} \\
 & \quad \textbf{elseif}\ \text{car}\,(\textit{stmt}) = \texttt{'no-op-mg}\ \textbf{then t} \\
 & \quad \textbf{elseif}\ \text{car}\,(\textit{stmt}) = \texttt{'signal-mg}\ \textbf{then t} \\
 & \quad \textbf{elseif}\ \text{car}\,(\textit{stmt}) = \texttt{'prog2-mg} \\
 & \quad \textbf{then}\ \text{meaning-induction-hint3}\,(\text{prog2-left-branch}\,(\textit{stmt}), \\
 & \qquad\qquad\qquad\qquad \textit{proc-list}, \\
 & \qquad\qquad\qquad\qquad \textit{mg-state}, \\
 & \qquad\qquad\qquad\qquad n-1, \\
 & \qquad\qquad\qquad\qquad \textit{name-alist}, \\
 & \qquad\qquad\qquad\qquad \textit{cond-list}, \\
 & \qquad\qquad\qquad\qquad \textit{new-proc}) \\
 & \qquad \wedge\ \text{meaning-induction-hint3}\,(\text{prog2-right-branch}\,(\textit{stmt}), \\
 & \qquad\qquad\qquad\qquad\qquad \textit{proc-list}, \\
 & \qquad\qquad\qquad\qquad\qquad \text{mg-meaning}\,(\text{prog2-left-branch}\,(\textit{stmt}), \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{cons}\,(\textit{new-proc}, \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{proc-list}), \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{mg-state}, \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad n-1), \\
 & \qquad\qquad\qquad\qquad\qquad n-1, \\
 & \qquad\qquad\qquad\qquad\qquad \textit{name-alist}, \\
 & \qquad\qquad\qquad\qquad\qquad \textit{cond-list},
\end{array}
$$

new-proc)
elseif car (*stmt*) = `'loop-mg`
then meaning-induction-hint3 (loop-body (*stmt*),
proc-list,
mg-state,
$n - 1$,
name-alist,
cons (`'leave`, *cond-list*),
new-proc)
$\wedge$ meaning-induction-hint3 (*stmt*,
proc-list,
mg-meaning (loop-body (*stmt*),
cons (*new-proc*,
proc-list),
mg-state,
$n - 1$),
$n - 1$,
name-alist,
cond-list,
new-proc)
elseif car (*stmt*) = `'if-mg`
then meaning-induction-hint3 (if-false-branch (*stmt*),
proc-list,
mg-state,
$n - 1$,
name-alist,
cond-list,
new-proc)
$\wedge$ meaning-induction-hint3 (if-true-branch (*stmt*),
proc-list,
mg-state,
$n - 1$,
name-alist,
cond-list,
new-proc)
elseif car (*stmt*) = `'begin-mg`
then meaning-induction-hint3 (begin-body (*stmt*),
proc-list,
mg-state,
$n - 1$,
name-alist,
append (when-labels (*stmt*), *cond-list*),
new-proc)
$\wedge$ meaning-induction-hint3 (when-handler (*stmt*),

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 f endif

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

THEOREM: new-proc-doesnt-affect-mg-meaning
(ok-mg-def-plistp (*proc-list*)
$\wedge$ ok-mg-statement (*stmt*, *cond-list*, *name-alist*, *proc-list*)
$\wedge$ new-proc-name (car (*new-proc*), *proc-list*))
$\rightarrow$ (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 t**
elseif ¬ normal (*mg-state*) **then t**
elseif resources-inadequatep (*stmt*, *proc-list*, *sizes*) **then t**
elseif car (*stmt*) = `'no-op-mg` **then t**
elseif car (*stmt*) = `'signal-mg` **then t**
elseif 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)
∧ meaning-induction-hint4 (prog2-right-branch (*stmt*),
proc-list,
mg-meaning-r (prog2-left-branch (*stmt*),
cons (*new-proc*,
proc-list),
mg-state,
$n - 1$,
sizes),
$n - 1$,
name-alist,
cond-list,
new-proc,
sizes)
elseif 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)
∧ meaning-induction-hint4 (*stmt*,
proc-list,
mg-meaning-r (loop-body (*stmt*),
cons (*new-proc*,
proc-list),
mg-state,

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

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)
$\wedge$ 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,
append (when-labels (*stmt*), *cond-list*),
new-proc,
sizes)
$\wedge$ 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,

cond-list,
new-proc,
sizes)
elseif car (*stmt*) = 'proc-call-mg
then meaning-induction-hint4 (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,
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))))))
elseif car (*stmt*) = 'predefined-proc-call-mg **then t**
else f endif

THEOREM: new-proc-doesnt-affect-resources-inadequatep
(ok-mg-def-plistp (*proc-list*)
∧ ok-mg-statement (*stmt*, *cond-list*, *name-alist*, *proc-list*)
∧ new-proc-name (car (*new-proc*), *proc-list*))
→ (resources-inadequatep (*stmt*, cons (*new-proc*, *proc-list*), *sizes*)
= resources-inadequatep (*stmt*, *proc-list*, *sizes*))

THEOREM: new-proc-doesnt-affect-mg-meaning-r
(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-r (*stmt*, *proc-list*, *mg-state*, *n*, *sizes*)
= mg-meaning-r (*stmt*, cons (*new-proc*, *proc-list*), *mg-state*, *n*, *sizes*))

THEOREM: new-proc-doesnt-affect-mg-meaning-r-2
(ok-mg-def-plistp (*proc-list*)
∧ ok-mg-statement (*stmt*, *cond-list*, *name-alist*, *proc-list*)

$\wedge$ new-proc-name (car (*new-proc*), *proc-list*))
$\rightarrow$ (mg-meaning-r (*stmt*, cons (*new-proc*, *proc-list*), *mg-state*, *n*, *sizes*)
= mg-meaning-r (*stmt*, *proc-list*, *mg-state*, *n*, *sizes*))

THEOREM: new-proc-doesnt-affect-clock-prog2-case
((*n* $\not\simeq$ 0)
$\wedge$ normal (*mg-state*)
$\wedge$ (car (*stmt*) = 'prog2-mg)
$\wedge$ ((ok-mg-def-plistp (*proc-list*)
$\wedge$ ok-mg-statement (prog2-right-branch (*stmt*),
cond-list,
name-alist,
proc-list)
$\wedge$ new-proc-name (car (*new-proc*), *proc-list*))
$\rightarrow$ (clock (prog2-right-branch (*stmt*),
proc-list,
mg-meaning (prog2-left-branch (*stmt*),
cons (*new-proc*, *proc-list*),
mg-state,
n − 1),
n − 1)
= 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),
n − 1)))
$\wedge$ ((ok-mg-def-plistp (*proc-list*)
$\wedge$ ok-mg-statement (prog2-left-branch (*stmt*),
cond-list,
name-alist,
proc-list)
$\wedge$ new-proc-name (car (*new-proc*), *proc-list*))
$\rightarrow$ (clock (prog2-left-branch (*stmt*), *proc-list*, *mg-state*, *n* − 1)
= clock (prog2-left-branch (*stmt*),
cons (*new-proc*, *proc-list*),
mg-state,
n − 1)))
$\wedge$ ok-mg-def-plistp (*proc-list*)
$\wedge$ ok-mg-statement (*stmt*, *cond-list*, *name-alist*, *proc-list*)
$\wedge$ new-proc-name (car (*new-proc*), *proc-list*))
$\rightarrow$ (clock (*stmt*, *proc-list*, *mg-state*, *n*)
= clock (*stmt*, cons (*new-proc*, *proc-list*), *mg-state*, *n*))

THEOREM: new-proc-doesnt-affect-clock-loop-case

```
((n ≠ 0)
 ∧  normal(mg-state)
 ∧  (car(stmt) = 'loop-mg)
 ∧  ((ok-mg-def-plistp(proc-list)
       ∧  ok-mg-statement(stmt, cond-list, name-alist, proc-list)
       ∧  new-proc-name(car(new-proc), proc-list))
      →  (clock(stmt,
                proc-list,
                mg-meaning(loop-body(stmt),
                           cons(new-proc, proc-list),
                           mg-state,
                           n − 1),
                n − 1)
          =  clock(stmt,
                   cons(new-proc, proc-list),
                   mg-meaning(loop-body(stmt),
                              cons(new-proc, proc-list),
                              mg-state,
                              n − 1),
                   n − 1)))
 ∧  ((ok-mg-def-plistp(proc-list)
       ∧  ok-mg-statement(loop-body(stmt),
                          cons('leave, cond-list),
                          name-alist,
                          proc-list)
       ∧  new-proc-name(car(new-proc), proc-list))
      →  (clock(loop-body(stmt), proc-list, mg-state, n − 1)
          =  clock(loop-body(stmt),
                   cons(new-proc, proc-list),
                   mg-state,
                   n − 1)))
 ∧  ok-mg-def-plistp(proc-list)
 ∧  ok-mg-statement(stmt, cond-list, name-alist, proc-list)
 ∧  new-proc-name(car(new-proc), proc-list))
 →  (clock(stmt, proc-list, mg-state, n)
     =  clock(stmt, cons(new-proc, proc-list), mg-state, n))
```

THEOREM: new-proc-doesnt-affect-clock-if-case

```
((n ≠ 0)
 ∧  normal(mg-state)
 ∧  (car(stmt) = 'if-mg)
 ∧  ((ok-mg-def-plistp(proc-list)
       ∧  ok-mg-statement(if-true-branch(stmt),
```

cond-list,
name-alist,
proc-list)
∧ new-proc-name (car (*new-proc*), *proc-list*))
→ (clock (if-true-branch (*stmt*), *proc-list*, *mg-state*, $n - 1$)
= clock (if-true-branch (*stmt*),
cons (*new-proc*, *proc-list*),
mg-state,
$n - 1$)))
∧ ((ok-mg-def-plistp (*proc-list*)
∧ ok-mg-statement (if-false-branch (*stmt*),
cond-list,
name-alist,
proc-list)
∧ new-proc-name (car (*new-proc*), *proc-list*))
→ (clock (if-false-branch (*stmt*), *proc-list*, *mg-state*, $n - 1$)
= clock (if-false-branch (*stmt*),
cons (*new-proc*, *proc-list*),
mg-state,
$n - 1$)))
∧ ok-mg-def-plistp (*proc-list*)
∧ ok-mg-statement (*stmt*, *cond-list*, *name-alist*, *proc-list*)
∧ new-proc-name (car (*new-proc*), *proc-list*))
→ (clock (*stmt*, *proc-list*, *mg-state*, *n*)
= clock (*stmt*, cons (*new-proc*, *proc-list*), *mg-state*, *n*))

THEOREM: new-proc-doesnt-affect-clock-begin-case
($n \not\simeq$ `0`)
∧ normal (*mg-state*)
∧ (car (*stmt*) = `'begin-mg`)
∧ ((ok-mg-def-plistp (*proc-list*)
∧ ok-mg-statement (when-handler (*stmt*),
cond-list,
name-alist,
proc-list)
∧ new-proc-name (car (*new-proc*), *proc-list*))
→ (clock (when-handler (*stmt*),
proc-list,
set-condition (mg-meaning (begin-body (*stmt*),
cons (*new-proc*, *proc-list*),
mg-state,
$n - 1$),
`'normal`),
$n - 1$)

```
        =  clock (when-handler (stmt),
                 cons (new-proc, proc-list),
                 set-condition (mg-meaning (begin-body (stmt),
                                            cons (new-proc,
                                                  proc-list),
                                            mg-state,
                                            n − 1),
                                'normal),
                 n − 1)))
∧  ((ok-mg-def-plistp (proc-list)
     ∧  ok-mg-statement (begin-body (stmt),
                         append (when-labels (stmt), cond-list),
                         name-alist,
                         proc-list)
     ∧  new-proc-name (car (new-proc), proc-list))
     →  (clock (begin-body (stmt), proc-list, mg-state, n − 1)
         =  clock (begin-body (stmt),
                   cons (new-proc, proc-list),
                   mg-state,
                   n − 1)))
∧  ok-mg-def-plistp (proc-list)
∧  ok-mg-statement (stmt, cond-list, name-alist, proc-list)
∧  new-proc-name (car (new-proc), proc-list))
→  (clock (stmt, proc-list, mg-state, n)
    =  clock (stmt, cons (new-proc, proc-list), mg-state, n))
```

THEOREM: new-proc-doesnt-affect-clock-proc-call-case

```
((n ≯ 0)
 ∧  normal (mg-state)
 ∧  (car (stmt) = 'proc-call-mg)
 ∧  ((ok-mg-def-plistp (proc-list)
      ∧  ok-mg-statement (def-body (fetch-called-def (stmt, proc-list)),
                         make-cond-list (fetch-called-def (stmt,
                                                           proc-list)),
                         make-name-alist (fetch-called-def (stmt,
                                                            proc-list)),
                         proc-list)
      ∧  new-proc-name (car (new-proc), proc-list))
      →  (clock (def-body (fetch-called-def (stmt, proc-list)),
                 proc-list,
                 make-call-environment (mg-state,
                                        stmt,
                                        fetch-called-def (stmt, proc-list)),
                 n − 1)
```

$$
\begin{array}{ll}
 & = \quad \text{clock}\,(\text{def-body}\,(\text{fetch-called-def}\,(\textit{stmt},\ \textit{proc-list})),\\
 & \qquad\quad \text{cons}\,(\textit{new-proc},\ \textit{proc-list}),\\
 & \qquad\quad \text{make-call-environment}\,(\textit{mg-state},\\
 & \qquad\qquad\qquad\qquad\qquad\quad \textit{stmt},\\
 & \qquad\qquad\qquad\qquad\qquad\quad \text{fetch-called-def}\,(\textit{stmt},\\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{proc-list})),\\
 & \qquad\quad n-1)))\\
\wedge & \text{ok-mg-def-plistp}\,(\textit{proc-list})\\
\wedge & \text{ok-mg-statement}\,(\textit{stmt},\ \textit{cond-list},\ \textit{name-alist},\ \textit{proc-list})\\
\wedge & \text{new-proc-name}\,(\text{car}\,(\textit{new-proc}),\ \textit{proc-list}))\\
\rightarrow & (\text{clock}\,(\textit{stmt},\ \textit{proc-list},\ \textit{mg-state},\ n)\\
 & \quad = \quad \text{clock}\,(\textit{stmt},\ \text{cons}\,(\textit{new-proc},\ \textit{proc-list}),\ \textit{mg-state},\ n))
\end{array}
$$

THEOREM: new-proc-doesnt-affect-clock

$$
\begin{array}{ll}
 & (\text{ok-mg-def-plistp}\,(\textit{proc-list})\\
\wedge & \text{ok-mg-statement}\,(\textit{stmt},\ \textit{cond-list},\ \textit{name-alist},\ \textit{proc-list})\\
\wedge & \text{new-proc-name}\,(\text{car}\,(\textit{new-proc}),\ \textit{proc-list}))\\
\rightarrow & (\text{clock}\,(\textit{stmt},\ \textit{proc-list},\ \textit{mg-state},\ n)\\
 & \quad = \quad \text{clock}\,(\textit{stmt},\ \text{cons}\,(\textit{new-proc},\ \textit{proc-list}),\ \textit{mg-state},\ n))
\end{array}
$$

THEOREM: new-proc-doesnt-affect-ok-mg-statement-proc-call-case

$$
\begin{array}{ll}
 & ((\text{car}\,(\textit{stmt}) = \texttt{'proc-call-mg})\\
\wedge & \text{ok-mg-def-plistp}\,(\textit{proc-list})\\
\wedge & \text{ok-mg-statement}\,(\textit{stmt},\ \textit{cond-list},\ \textit{name-alist},\ \textit{proc-list})\\
\wedge & \text{new-proc-name}\,(\text{car}\,(\textit{new-proc}),\ \textit{proc-list}))\\
\rightarrow & \text{ok-mg-statement}\,(\textit{stmt},\ \textit{cond-list},\ \textit{name-alist},\ \text{cons}\,(\textit{new-proc},\ \textit{proc-list}))
\end{array}
$$

EVENT: Enable ok-mg-statement.

THEOREM: new-proc-doesnt-affect-ok-mg-statement

$$
\begin{array}{ll}
 & (\text{ok-mg-def-plistp}\,(\textit{proc-list})\\
\wedge & \text{ok-mg-statement}\,(\textit{stmt},\ \textit{cond-list},\ \textit{name-alist},\ \textit{proc-list})\\
\wedge & \text{new-proc-name}\,(\text{car}\,(\textit{new-proc}),\ \textit{proc-list}))\\
\rightarrow & \text{ok-mg-statement}\,(\textit{stmt},\ \textit{cond-list},\ \textit{name-alist},\ \text{cons}\,(\textit{new-proc},\ \textit{proc-list}))
\end{array}
$$

THEOREM: new-proc-preserves-ok-mg-def

$$
\begin{array}{ll}
 & (\text{new-proc-name}\,(\text{car}\,(\textit{new-proc}),\ \textit{pl})\\
\wedge & \text{ok-mg-def-plistp}\,(\textit{pl})\\
\wedge & \text{ok-mg-def}\,(\textit{def},\ \textit{pl}))\\
\rightarrow & \text{ok-mg-def}\,(\textit{def},\ \text{cons}\,(\textit{new-proc},\ \textit{pl}))
\end{array}
$$

THEOREM: new-proc-preserves-ok-mg-def-plistp1

$$
\begin{array}{ll}
 & (\text{new-proc-name}\,(\text{car}\,(\textit{new-proc}),\ \textit{pl2})\\
\wedge & \text{ok-mg-def-plistp}\,(\textit{pl2})\\
\wedge & \text{ok-mg-def-plistp1}\,(\textit{pl1},\ \textit{pl2}))\\
\rightarrow & \text{ok-mg-def-plistp1}\,(\textit{pl1},\ \text{cons}\,(\textit{new-proc},\ \textit{pl2}))
\end{array}
$$

THEOREM: make-alist-make-locals-list-preserves-signatures-match
mg-alistp (*mg-alist*)
→ 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
(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)))
→ 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
listp (mg-to-p-simple-literal-list (*x*)) = listp (*x*)

THEOREM: initial-temp-stk-reversed-listp
(mg-alistp (*x*) ∧ listp (*x*)) → listp (initial-temp-stk-reversed (*x*))

THEOREM: length-initial-temp-stk-reversed
mg-alistp (*alist*)
→ (length (initial-temp-stk-reversed (*alist*)) = data-length (*alist*))

DEFINITION:
initial-bindings-induction-hint (*mg-alist*, *n*, *lst*)
= **if** *mg-alist* ≃ **nil then t**
elseif simple-mg-type-refp (cadar (*mg-alist*))
then initial-bindings-induction-hint (cdr (*mg-alist*),
1 + *n*,
cons (mg-to-p-simple-literal (caddar (*mg-alist*)),
lst))
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
(mg-alistp (*mg-alist*) ∧ (*n* = length (*lst*)) ∧ all-cars-unique (*mg-alist*))
→ 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
(ok-mg-statep (*mg-state*, *cond-list*) ∧ all-cars-unique (mg-alist (*mg-state*)))
→ mg-vars-list-ok-in-p-state (mg-alist (*mg-state*),
initial-bindings (mg-alist (*mg-state*), 0),
initial-temp-stk (mg-alist (*mg-state*)))

THEOREM: initial-bindings-all-pointers-bigger
all-cars-unique (*alist*)
→ all-pointers-bigger (collect-pointers (initial-bindings (*alist*, *n*), *alist*),
n)

THEOREM: no-p-aliasing-in-initial-bindings
(all-cars-unique (*mg-alist*) ∧ mg-alistp (*mg-alist*) ∧ (*n* ∈ **N**))
→ no-p-aliasing (initial-bindings (*mg-alist*, *n*), *mg-alist*)

THEOREM: leave-not-state-cc
(ok-mg-statep (*mg-state*, *cond-list*) ∧ identifier-plistp (*cond-list*))
→ (cc (*mg-state*) ≠ 'leave)

THEOREM: translation-is-correct3
(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,
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)))))))))))
→ (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*))

```
;; 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,
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,
n,
list (data-length (mg-alist (*mg-state*)),
2 + length (mg-alist (*mg-state*)))))))

$\rightarrow$ (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*))

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
    mp-stk1, 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-reversed, 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-list-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-be
gin-case, 24
new-proc-doesnt-affect-clock-if
-case, 23
new-proc-doesnt-affect-clock-lo
op-case, 23
new-proc-doesnt-affect-clock-pr
oc-call-case, 25
og2-case, 22
new-proc-doesnt-affect-fetch-ca
lled-def, 15
new-proc-doesnt-affect-mg-meani
ng, 18
ng-proc-call-case, 15
ng-r, 21
ng-r-2, 21
new-proc-doesnt-affect-ok-mg-st
atement, 26
atement-proc-call-case, 26
new-proc-doesnt-affect-resource
s-inadequatep, 21
new-proc-name, 15, 16, 18, 21–29
new-proc-preserves-ok-mg-def, 26
new-proc-preserves-ok-mg-def-pli
stp, 27
stp1, 26
no-p-aliasing, 7, 9, 12, 28
no-p-aliasing-in-initial-bindin
gs, 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-plistp1, 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-signat
ures-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-procp, 7, 9, 12

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