EVENT: Start with the library "c4".

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                        ;;
;;                        %Mapping Call Parameters                        ;;
;;                                                                        ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:

```
mg-to-p-local-values (locals)
=   if locals ≃ nil then nil
    elseif simple-mg-type-refp (cadr (car (locals)))
    then cons (mg-to-p-simple-literal (caddr (car (locals))),
               mg-to-p-local-values (cdr (locals)))
    else append (mg-to-p-simple-literal-list (caddr (car (locals))),
                 mg-to-p-local-values (cdr (locals))) endif
```

THEOREM: mg-to-p-local-values-plistp
plistp (mg-to-p-local-values (*lst*))

```
;; Given a list of formals with the call site actuals, this gives the list for the new stack
;; frame.  Each of the actuals is guaranteed to be an identifier and each of these is in
;; the previous frame with the address of the value in my-stack.  Thus, I need only copy
;; these addresses into the current frame.
```

DEFINITION:

```
map-call-formals (formals, actuals, bindings)
=   if formals ≃ nil then nil
    else cons (cons (car (car (formals)),
                     cdr (assoc (car (actuals), bindings))),
               map-call-formals (cdr (formals), cdr (actuals), bindings)) endif
```

THEOREM: length-map-call-formals
length (map-call-formals (*formals*, *actuals*, *bindings*)) = length (*formals*)

THEOREM: map-call-formals-plistp
plistp (map-call-formals (x, y, z))

THEOREM: listcars-map-call-formals
listcars (map-call-formals (*formals*, *actuals*, *bindings*)) = listcars (*formals*)

```
;; Each of the local values is placed onto the temp-stk, then the indexes into
;; the stack are placed there as well.  The distance of the value from the index
;; depends on the size of the elements between.

;; The initial value of n is (length temp-stk)
```

DEFINITION:

```
map-call-locals (locals, n)
=   if locals ≃ nil then nil
    elseif simple-mg-type-refp (cadr (car (locals)))
    then cons (cons (car (car (locals)), tag ('nat, n)),
                    map-call-locals (cdr (locals), 1 + n))
    else cons (cons (car (car (locals)), tag ('nat, n)),
                    map-call-locals (cdr (locals),
                                     array-length (cadr (car (locals)))
                                     +   n)) endif
```

THEOREM: length-map-call-locals
length (map-call-locals (*locals*, *n*)) = length (*locals*)

THEOREM: map-call-locals-plistp
plistp (map-call-locals (*locals*, *n*))

THEOREM: map-call-locals-preserves-listcars
listcars (map-call-locals (*locals*, *m*)) = listcars (*locals*)

```
;; The topmost frame on the ctrl-stk at the beginning of the body of the proc-call
;; contains the p-formals which represent both the formals and locals of the mg
;; subroutine.  The formals have the values of the actuals in the previous frame
;; and these are guaranteed to be addresses into the temp-stk.  The locals have been
;; placed on the temp-stk as well and the address computed for this frame are those
;; locations.  Thus, upon entry the following invariant is established: every local
;; (in the frame) contains an index into the temp-stk which contains the corresponding
;; value.
```

DEFINITION:

```
make-frame-alist (def, stmt, ctrl-stk, temp-stk)
=   append (map-call-locals (def-locals (def), length (temp-stk)),
            map-call-formals (def-formals (def),
                              call-actuals (stmt),
                              bindings (top (ctrl-stk))))
```

DEFINITION:

```
mg-actuals-to-p-actuals (mg-actuals, bindings)
=   if mg-actuals ≃ nil then nil
    else cons (cdr (assoc (car (mg-actuals), bindings)),
               mg-actuals-to-p-actuals (cdr (mg-actuals), bindings)) endif
```

THEOREM: length-mg-actuals-to-p-actuals
length (mg-actuals-to-p-actuals (*mg-actuals*, *bindings*)) = length (*mg-actuals*)

THEOREM: mg-actuals-to-p-actuals-plistp
plistp (mg-actuals-to-p-actuals (*actuals*, *bindings*))

```
;; %mapping call parameters
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                           ;;
;;                             THE TRANSLATOR                                ;;
;;                                                                           ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

EVENT: Add the shell *make-cinfo*, with recognizer function symbol *cinfop* and 3 accessors: *code*, with type restriction (none-of) and default value zero; *label-alist*, with type restriction (none-of) and default value zero; *label-cnt*, with type restriction (one-of numberp) and default value zero.

DEFINITION:
nullify (*cinfo*) = make-cinfo (**nil**, label-alist (*cinfo*), label-cnt (*cinfo*))

DEFINITION:
add-code (*cinfo*, *code*)
= make-cinfo (append (code (*cinfo*), *code*),
label-alist (*cinfo*),
label-cnt (*cinfo*))

DEFINITION:
discard-label (*cinfo*)
= make-cinfo (code (*cinfo*), cdr (label-alist (*cinfo*)), label-cnt (*cinfo*))

DEFINITION:
set-label-alist (*cinfo*, *new-label-alist*)
= make-cinfo (code (*cinfo*), *new-label-alist*, label-cnt (*cinfo*))

```
;; Notice that I could simply use the VALUE function directly.
```

DEFINITION:
fetch-label (*condition*, *label-alist*) = cdr (assoc (*condition*, *label-alist*))

```
;; If this definition stays unchanged, I can eliminate it entirely in favor of the
;; simpler hyp on code.
```

DEFINITION: ok-cinfop (*cinfo*) = plistp (code (*cinfo*))

```
;; Given a list (x1 ... xn) and a label l, this generated the list
;; ((x1 . l) (x2 . l) ... (xn . l)).  Notice that this allows that use
;; of the VALUE function for accessing the label.
```

DEFINITION:

```
make-label-alist (name-list, label)
=   if name-list ≃ nil then nil
    else cons (cons (car (name-list), label),
               make-label-alist (cdr (name-list), label)) endif
```

DEFINITION:

```
push-local-array-values-code (array-value)
=   if array-value ≃ nil then nil
    else cons (list ('push-constant,
                     mg-to-p-simple-literal (car (array-value))),
               push-local-array-values-code (cdr (array-value))) endif
```

THEOREM: length-push-local-array-values-code
length (push-local-array-values-code (*array-value*)) = length (*array-value*)

THEOREM: length-push-local-array-values-code2
(ok-mg-local-data-decl (*local*) ∧ (¬ simple-mg-type-refp (cadr (*local*))))
→ (array-length (cadr (*local*)) = length (caddr (*local*)))

EVENT: Disable length-push-local-array-values-code2.

DEFINITION:

```
push-locals-values-code (locals)
=   if locals ≃ nil then nil
    elseif simple-mg-type-refp (cadr (car (locals)))
    then cons (list ('push-constant,
                     mg-to-p-simple-literal (caddr (car (locals)))),
               push-locals-values-code (cdr (locals)))
    else append (push-local-array-values-code (caddr (car (locals))),
                 push-locals-values-code (cdr (locals))) endif
```

THEOREM: length-push-locals-values-code
ok-mg-local-data-plistp (*locals*)
→ (length (push-locals-values-code (*locals*)) = data-length (*locals*))

THEOREM: length-mg-to-p-local-values
ok-mg-local-data-plistp (*locals*)
→ (length (mg-to-p-local-values (*locals*)) = data-length (*locals*))

THEOREM: no-labels-in-push-local-array-values-code
find-labelp (*n*, push-local-array-values-code (*value*)) = **f**

THEOREM: no-labels-in-push-locals-values-code
find-labelp (*n*, push-locals-values-code (*actuals*)) = **f**

DEFINITION:

```
push-locals-addresses-code (locals, n)
=   if locals ≃ nil then nil
    elseif simple-mg-type-refp (cadr (car (locals)))
    then cons (list ('push-temp-stk-index, n),
               push-locals-addresses-code (cdr (locals), n))
    else cons (list ('push-temp-stk-index, n),
              push-locals-addresses-code (cdr (locals),
                                          1 + (n − array-length (cadr (car (locals)))))) endif
```

THEOREM: length-push-locals-addresses-code
length (push-locals-addresses-code (*locals*, *n*)) = length (*locals*)

THEOREM: no-labels-in-push-locals-addresses-code
find-labelp (*n*, push-locals-addresses-code (*actuals*, *m*)) = **f**

DEFINITION:

```
push-actuals-code (actuals)
=   if actuals ≃ nil then nil
    else cons (list ('push-local, car (actuals)),
              push-actuals-code (cdr (actuals))) endif
```

THEOREM: no-labels-in-push-actuals-code
find-labelp (*n*, push-actuals-code (*actuals*)) = **f**

THEOREM: length-push-actuals-code
length (push-actuals-code (*actuals*)) = length (*actuals*)

DEFINITION:

```
push-parameters-code (locals, actuals)
=   append (push-locals-values-code (locals),
            append (push-locals-addresses-code (locals,
                                                data-length (locals) − 1),
                    push-actuals-code (actuals)))
```

THEOREM: length-push-parameters-code

```
ok-mg-local-data-plistp (locals)
→   (length (push-parameters-code (locals, actuals))
      =   (data-length (locals) + length (locals) + length (actuals)))
```

```
;; COMPILING THE CONDITION MAPPING
;;
;; Generate the list '(lc lc+1 lc+2 ... lc+n-1).  These are the labels
;; necessary for the condition computation jumps.
```

DEFINITION:

```
cond-case-jump-label-list (lc, n)
=   if n ≃ 0 then nil
    else cons (lc, cond-case-jump-label-list (1 + lc, n − 1)) endif
```

THEOREM: length-cond-case-jump-label-list

```
 length (cond-case-jump-label-list (lc, n)) = fix (n)
```

DEFINITION:

```
index-cond-case-induction-hint (i, j, k)
=   if k ≃ 0 then t
    else index-cond-case-induction-hint (i − 1, 1 + j, k − 1) endif
```

THEOREM: get-cond-case-jump-label-list

```
 ((i < k) ∧ (j ∈ N))
 →   (get (i, cond-case-jump-label-list (j, k)) = (i + j))
```

EVENT: Disable get-cond-case-jump-label-list.

DEFINITION:

```
cond-conversion (actual-conds, lc, cond-list, label-alist)
=   if actual-conds ≃ nil then nil
    else cons (list ('dl,
                     lc,
                     nil,
                     list ('push-constant,
                           mg-cond-to-p-nat (car (actual-conds),
                                             cond-list))),
                cons ('(pop-global c-c),
                      cons (list ('jump,
                                  fetch-label (car (actual-conds),
                                               label-alist)),
                            cond-conversion (cdr (actual-conds),
                                             1 + lc,
                                             cond-list,
                                             label-alist)))) endif
```

THEOREM: length-cond-conversion

```
 length (cond-conversion (call-conds, lc, cond-list, label-alist))
 =   (3 ∗ length (call-conds))
```

Definition:

```
label-cnt-list (lc, n)
=   if n ≃ 0 then nil
    else cons (lc, label-cnt-list (lc, n − 1)) endif
```

Theorem: length-label-cnt-list

length (label-cnt-list (*lc*, *n*)) = fix (*n*)

```
;; I must make sure that the condition index is in-range.  I can do this by using the def-c
;; the list to index rather than the make-cond-list.

;; This was changed slightly to add two additional condition onto the front of the list.  Th
;; because the condition index for 'normal is not zero any longer, but is now two.  Conseque
;; I'm going to use the condition index as an index into the cond-case-jump-label-list, I mu
;; decrement it twice or kludge the list structure.  I simply add the label for 'routineerr
;; at the beginning.
```

Definition:

```
condition-map-code (actual-conds, lc, cond-list, label-alist, proc-locals-lngth)
=   append (list (list ('push-global, 'c-c),
                  append (cons ('jump-case,
                                cons (lc,
                                      cons (lc,
                                            cond-case-jump-label-list (1 + lc,
                                                                        1 + length (actual-conds))))),
                          label-cnt-list (lc, proc-locals-lngth)),
                 list ('dl, lc, nil, '(push-constant (nat 1))),
                 '(pop-global c-c),
                 list ('jump, fetch-label ('routineerror, label-alist))),
            append (cond-conversion (actual-conds,
                                     1 + (1 + lc),
                                     cond-list,
                                     label-alist),
                    list (list ('dl, 1 + lc, nil, '(no-op)))))
```

Definition:

```
proc-call-code (cinfo, stmt, cond-list, locals, cond-locals-lngth)
=   append (push-parameters-code (locals, call-actuals (stmt)),
            cons (list ('call, call-name (stmt)),
                  condition-map-code (call-conds (stmt),
                                      label-cnt (cinfo),
                                      cond-list,
                                      label-alist (cinfo),
                                      cond-locals-lngth)))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                    ;;
;;                   %Compiling the Predefineds                       ;;
;;                                                                    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The following functions define the sequence of statements laid
;; down for a call to a predefined procedure.
```

Definition:

```
mg-simple-variable-assignment-call-sequence (stmt)
=   list (list ('push-local, car (call-actuals (stmt))),
          list ('push-local, cadr (call-actuals (stmt))),
          '(call mg-simple-variable-assignment))
```

Definition:

```
mg-simple-constant-assignment-call-sequence (stmt)
=   list (list ('push-local, car (call-actuals (stmt))),
          list ('push-constant,
                mg-to-p-simple-literal (cadr (call-actuals (stmt)))),
          '(call mg-simple-constant-assignment))
```

Definition:

```
mg-simple-variable-eq-call-sequence (stmt)
=   list (list ('push-local, car (call-actuals (stmt))),
          list ('push-local, cadr (call-actuals (stmt))),
          list ('push-local, caddr (call-actuals (stmt))),
          '(call mg-simple-variable-eq))
```

Definition:

```
mg-simple-constant-eq-call-sequence (stmt)
=   list (list ('push-local, car (call-actuals (stmt))),
          list ('push-local, cadr (call-actuals (stmt))),
          list ('push-constant,
                mg-to-p-simple-literal (caddr (call-actuals (stmt)))),
          '(call mg-simple-constant-eq))
```

Definition:

```
mg-integer-le-call-sequence (stmt)
=   list (list ('push-local, car (call-actuals (stmt))),
          list ('push-local, cadr (call-actuals (stmt))),
          list ('push-local, caddr (call-actuals (stmt))),
          '(call mg-integer-le))
```

DEFINITION:

```
mg-integer-unary-minus-call-sequence (stmt, label-alist)
=    list (list ('push-local, car (call-actuals (stmt))),
           list ('push-local, cadr (call-actuals (stmt))),
           '(call mg-integer-unary-minus),
           '(push-global c-c),
           '(sub1-nat),
           list ('test-nat-and-jump,
                 'zero,
                 fetch-label ('routineerror, label-alist)))
```

DEFINITION:

```
mg-integer-add-call-sequence (stmt, label-alist)
=    list (list ('push-local, car (call-actuals (stmt))),
           list ('push-local, cadr (call-actuals (stmt))),
           list ('push-local, caddr (call-actuals (stmt))),
           '(call mg-integer-add),
           '(push-global c-c),
           '(sub1-nat),
           list ('test-nat-and-jump,
                 'zero,
                 fetch-label ('routineerror, label-alist)))
```

DEFINITION:

```
mg-integer-subtract-call-sequence (stmt, label-alist)
=    list (list ('push-local, car (call-actuals (stmt))),
           list ('push-local, cadr (call-actuals (stmt))),
           list ('push-local, caddr (call-actuals (stmt))),
           '(call mg-integer-subtract),
           '(push-global c-c),
           '(sub1-nat),
           list ('test-nat-and-jump,
                 'zero,
                 fetch-label ('routineerror, label-alist)))
```

DEFINITION:

```
mg-boolean-or-call-sequence (stmt)
=    list (list ('push-local, car (call-actuals (stmt))),
           list ('push-local, cadr (call-actuals (stmt))),
           list ('push-local, caddr (call-actuals (stmt))),
           '(call mg-boolean-or))
```

DEFINITION:

```
mg-boolean-and-call-sequence (stmt)
=    list (list ('push-local, car (call-actuals (stmt))),
          list ('push-local, cadr (call-actuals (stmt))),
          list ('push-local, caddr (call-actuals (stmt))),
          '(call mg-boolean-and))
```

Definition:

```
mg-boolean-not-call-sequence (stmt)
=    list (list ('push-local, car (call-actuals (stmt))),
          list ('push-local, cadr (call-actuals (stmt))),
          '(call mg-boolean-not))
```

```
;; The 4th argument is a numberp supplied by the pre-processor which is
;; the size of the array.  This is necessary for bounds checking.
;; >> Do I need to guarantee that it is a small-integerp?
```

Definition:

```
mg-index-array-call-sequence (stmt, label-alist)
=    list (list ('push-local, car (call-actuals (stmt))),
          list ('push-local, cadr (call-actuals (stmt))),
          list ('push-local, caddr (call-actuals (stmt))),
          list ('push-constant, tag ('int, cadddr (call-actuals (stmt)))),
          '(call mg-index-array),
          '(push-global c-c),
          '(sub1-nat),
          list ('test-nat-and-jump,
               'zero,
               fetch-label ('routineerror, label-alist)))
```

Definition:

```
mg-array-element-assignment-call-sequence (stmt, label-alist)
=    list (list ('push-local, car (call-actuals (stmt))),
          list ('push-local, cadr (call-actuals (stmt))),
          list ('push-local, caddr (call-actuals (stmt))),
          list ('push-constant, tag ('int, cadddr (call-actuals (stmt)))),
          '(call mg-array-element-assignment),
          '(push-global c-c),
          '(sub1-nat),
          list ('test-nat-and-jump,
               'zero,
               fetch-label ('routineerror, label-alist)))
```

Definition:

```
predefined-proc-call-sequence (stmt, label-alist)
=    case on call-name (stmt):
     case = mg-simple-variable-assignment
```

```
then mg-simple-variable-assignment-call-sequence (stmt)
case = mg-simple-constant-assignment
 then mg-simple-constant-assignment-call-sequence (stmt)
case = mg-simple-variable-eq
 then mg-simple-variable-eq-call-sequence (stmt)
case = mg-simple-constant-eq
 then mg-simple-constant-eq-call-sequence (stmt)
case = mg-integer-le
 then mg-integer-le-call-sequence (stmt)
case = mg-integer-unary-minus
 then mg-integer-unary-minus-call-sequence (stmt, label-alist)
case = mg-integer-add
 then mg-integer-add-call-sequence (stmt, label-alist)
case = mg-integer-subtract
 then mg-integer-subtract-call-sequence (stmt, label-alist)
case = mg-boolean-or
 then mg-boolean-or-call-sequence (stmt)
case = mg-boolean-and
 then mg-boolean-and-call-sequence (stmt)
case = mg-boolean-not
 then mg-boolean-not-call-sequence (stmt)
case = mg-index-array
 then mg-index-array-call-sequence (stmt, label-alist)
case = mg-array-element-assignment
 then mg-array-element-assignment-call-sequence (stmt, label-alist)
otherwise nil endcase
```

Event: Disable predefined-proc-call-sequence.

```
;; We now consider the bodies of the predefined routines.
```

Definition:
mg-simple-variable-assignment-translation

```
=  '(mg-simple-variable-assignment
      (dest source)
      nil
      (push-local source)
      (fetch-temp-stk)
      (push-local dest)
      (deposit-temp-stk)
      (ret))
```

Definition:

MG-SIMPLE-CONSTANT-ASSIGNMENT-TRANSLATION

```
=  '(mg-simple-constant-assignment
      (dest source)
      nil
      (push-local source)
      (push-local dest)
      (deposit-temp-stk)
      (ret))
```

```
;; >>> Notice that deposit-temp-stk is different from my old deposit-temp
;;     in the order of args on the stack.  THESE WILL ALL HAVE TO CHANGE.
```

DEFINITION:
MG-SIMPLE-VARIABLE-EQ-TRANSLATION

```
=  '(mg-simple-variable-eq
      (ans x y)
      nil
      (push-local x)
      (fetch-temp-stk)
      (push-local y)
      (fetch-temp-stk)
      (eq)
      (push-local ans)
      (deposit-temp-stk)
      (ret))
```

DEFINITION:
MG-SIMPLE-CONSTANT-EQ-TRANSLATION

```
=  '(mg-simple-constant-eq
      (ans x y)
      nil
      (push-local x)
      (fetch-temp-stk)
      (push-local y)
      (eq)
      (push-local ans)
      (deposit-temp-stk)
      (ret))
```

DEFINITION:
MG-INTEGER-LE-TRANSLATION

```
=  '(mg-integer-le
      (ans x y)
      nil
      (push-local y)
      (fetch-temp-stk)
      (push-local x)
      (fetch-temp-stk)
      (lt-int)
      (not-bool)
      (push-local ans)
      (deposit-temp-stk)
      (ret))

;; Since the representable positives and negatives are not
;; exactly complementary, I must check that the integer in question
;; is not that exact negative which would cause a problem.
```

DEFINITION:
MG-INTEGER-UNARY-MINUS-TRANSLATION

```
=  '(mg-integer-unary-minus
      (ans x)
      ((min-int (int -2147483648)) (temp-x (int 0)))
      (push-local x)
      (fetch-temp-stk)
      (set-local temp-x)
      (push-local min-int)
      (eq)
      (test-bool-and-jump f 0)
      (push-constant (nat 1))
      (pop-global c-c)
      (jump 1)
      (dl 0 nil (push-local temp-x))
      (neg-int)
      (push-local ans)
      (deposit-temp-stk)
      (dl 1 nil (ret)))
```

DEFINITION:
MG-INTEGER-ADD-TRANSLATION

```
=  '(mg-integer-add
      (ans y z)
      ((t1 (int 0)))
      (push-constant (bool f))
      (push-local y)
      (fetch-temp-stk)
      (push-local z)
      (fetch-temp-stk)
      (add-int-with-carry)
      (pop-local t1)
      (test-bool-and-jump t 0)
      (push-local t1)
      (push-local ans)
      (deposit-temp-stk)
      (jump 1)
      (dl 0 nil (push-constant (nat 1)))
      (pop-global c-c)
      (dl 1 nil (ret)))
```

DEFINITION:
MG-INTEGER-SUBTRACT-TRANSLATION

```
=  '(mg-integer-subtract
      (ans y z)
      ((t1 (int 0)))
      (push-constant (bool f))
      (push-local y)
      (fetch-temp-stk)
      (push-local z)
      (fetch-temp-stk)
      (sub-int-with-carry)
      (pop-local t1)
      (test-bool-and-jump t 0)
      (push-local t1)
      (push-local ans)
      (deposit-temp-stk)
      (jump 1)
      (dl 0 nil (push-constant (nat 1)))
      (pop-global c-c)
      (dl 1 nil (ret)))
```

DEFINITION:
MG-BOOLEAN-OR-TRANSLATION

```
=  '(mg-boolean-or
      (ans b1 b2)
      nil
      (push-local b1)
      (fetch-temp-stk)
      (push-local b2)
      (fetch-temp-stk)
      (or-bool)
      (push-local ans)
      (deposit-temp-stk)
      (ret))
```

Definition:
mg-boolean-and-translation

```
=  '(mg-boolean-and
     (ans b1 b2)
     nil
     (push-local b1)
     (fetch-temp-stk)
     (push-local b2)
     (fetch-temp-stk)
     (and-bool)
     (push-local ans)
     (deposit-temp-stk)
     (ret))
```

Definition:
mg-boolean-not-translation

```
=  '(mg-boolean-not
     (ans b1)
     nil
     (push-local b1)
     (fetch-temp-stk)
     (not-bool)
     (push-local ans)
     (deposit-temp-stk)
     (ret))
```

```
;; ans := A[i] of size
;; How do I know that the sub-nat to compute the index doesn't give an error?
```

Definition:
mg-index-array-translation

```
=  '(mg-index-array
     (ans a i array-size)
     ((temp-i (nat 0)))
     (push-local i)
     (fetch-temp-stk)
     (set-local temp-i)
     (test-int-and-jump neg 0)
     (push-local array-size)
     (push-local temp-i)
     (sub-int)
     (test-int-and-jump not-pos 0)
     (push-local a)
     (push-local temp-i)
```

```
      (int-to-nat)
      (add-nat)
      (fetch-temp-stk)
      (push-local ans)
      (deposit-temp-stk)
      (jump 1)
      (dl 0 nil (push-constant (nat 1)))
      (pop-global c-c)
      (dl 1 nil (ret)))

;; (mg-array-element-assignment A i value size)
```

DEFINITION:
MG-ARRAY-ELEMENT-ASSIGNMENT-TRANSLATION

```
=  '(mg-array-element-assignment
      (a i value array-size)
      ((temp-i (nat 0)))
      (push-local i)
      (fetch-temp-stk)
      (set-local temp-i)
      (test-int-and-jump neg 0)
      (push-local array-size)
      (push-local temp-i)
      (sub-int)
      (test-int-and-jump not-pos 0)
      (push-local value)
      (fetch-temp-stk)
      (push-local a)
      (push-local temp-i)
      (int-to-nat)
      (add-nat)
      (deposit-temp-stk)
      (jump 1)
      (dl 0 nil (push-constant (nat 1)))
      (pop-global c-c)
      (dl 1 nil (ret)))

;; The list of translations of the predefined routines is appended
;; to the list of translations of the user-defined routines and
;; becomes the program segment of the Piton program.
```

DEFINITION:
PREDEFINED-PROCEDURE-TRANSLATIONS-LIST

= list (MG-SIMPLE-VARIABLE-ASSIGNMENT-TRANSLATION,
MG-SIMPLE-CONSTANT-ASSIGNMENT-TRANSLATION,
MG-SIMPLE-VARIABLE-EQ-TRANSLATION,
MG-SIMPLE-CONSTANT-EQ-TRANSLATION,
MG-INTEGER-LE-TRANSLATION,
MG-INTEGER-UNARY-MINUS-TRANSLATION,
MG-INTEGER-ADD-TRANSLATION,
MG-INTEGER-SUBTRACT-TRANSLATION,
MG-BOOLEAN-OR-TRANSLATION,
MG-BOOLEAN-AND-TRANSLATION,
MG-BOOLEAN-NOT-TRANSLATION,
MG-INDEX-ARRAY-TRANSLATION,
MG-ARRAY-ELEMENT-ASSIGNMENT-TRANSLATION)

EVENT: Disable predefined-procedure-translations-list.

```
;; Insist that the condition on an IF statement is a variable.  This means that
;; it cannot be a boolean literal.  Hence the code for computing it is always.
;;   (push-local b)
;;   (fetch-temp-stk)
;; Otherwise, the number of statements would vary and I don't want to deal with that
;; now.  This is consistent with the convention for proc-calls.

;; Condition on an IF statement is either a boolean literal or the address of a
;; boolean in the my-stack array.

;; SIGNAL
;;
;;        (push-constant (nat n))            n is the index of condition in cond-list
;;        (pop-global c-c)
;;        (jump label)                       label is associated label of condition in labe
;;
;; PROG2
;;
;;        "code for left branch"
;;        "code for right branch"
;;
;;
;; LOOP
;;
;;         (dl l0 nil (no-op))
;;         "code for loop-body"
;;         (jump L0)
;;         (dl l1 nil (push-constant (nat 2)))
;;          (pop-global c-c)
;;
;; IF
;;
;;          (push-local b)
;;          (fetch-temp-stk)
;;          (test-bool-and-jump false L0)
;;          "code for true branch"
;;          (jump L1)
;;          (dl l0 nil (no-op))
;;          "code for false branch"
;;          (dl l1 nil (no-op))
;;
;; BEGIN-WHEN
;;
;;         "code for begin-body"
;;         (jump L1)
;;         (dl l0 nil (push-constant (nat 2)))
;;         (pop-global c-c)
;;         "code for when-arm-body"
;;         (dl l1 nil (no-op))
;;
;; PROC-CALL
;; For the statement
;;  (PROC-CALL-MG name (act1 act2 ... actj) (cond1 cond2 ... condn))
;; we make the following code.
;;    push-locals-values-code
;;    push-locals-addresses-code
;;    push actuals-code
;;    (call name)
;;    (push-global c-c)
;;    (case-jump (L0 L1 L2 ... Ln))
;;    (push-constant (nat 1))
;;    (pop-global c-c)
;;    (jump "label-for-routineerror")
;;    (dl l1 nil (push-constant "condition-number for cond1"))
;;    (pop-global c-c)
;;    (jump "label for cond1")
;;    (dl l2 nil (push-constant "condition-number for cond2"))
;;    (pop-global c-c)
;;    (jump "label for cond2")
;;        ...
;;    (dl ln nil (push-constant "condition-number for condn"))
;;    (pop-global c-c)
;;    (jump "label for condn")
;;    (dl 10 nil (no-op))
;;
;; PREDEFINED-PROC-CALL
```

DEFINITION:

```
translate (cinfo, cond-list, stmt, proc-list)
=    case on car (stmt):
     case = no-op-mg
     then cinfo
     case = signal-mg
      then make-cinfo (append (code (cinfo),
                                    list (list ('push-constant,
                                              mg-cond-to-p-nat (signalled-condition (stmt),
                                                                    cond-list)),
                                         list ('pop-global, 'c-c),
                                         list ('jump,
                                              fetch-label (signalled-condition (stmt),
                                                           label-alist (cinfo))))),
                           label-alist (cinfo),
                           label-cnt (cinfo))
     case = prog2-mg
      then translate (translate (cinfo,
                                    cond-list,
                                    prog2-left-branch (stmt),
                                    proc-list),
                         cond-list,
                         prog2-right-branch (stmt),
                         proc-list)
     case = loop-mg
      then discard-label (add-code (translate (make-cinfo (append (code (cinfo),
                                                                           list (list ('dl,
                                                                                     label-cnt (cinfo),
                                                                                     nil,
                                                                                     '(no-op)))),
                                                                 cons (cons ('leave,
                                                                              1 + label-cnt (cinfo)),
                                                                       label-alist (cinfo)),
                                                                 1 + (1 + label-cnt (cinfo))),
                                                    cond-list,
                                                    loop-body (stmt),
                                                    proc-list),
                                         list (list ('jump, label-cnt (cinfo)),
                                    list ('dl,
                                        1 + label-cnt (cinfo),
                                        nil,
                                        '(push-constant
                                           (nat 2))),
                                    '(pop-global c-c))))
case = if-mg
 then add-code (translate (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),
                                                            1 + (1 + label-cnt (cinfo))),
                                                cond-list,
                                                if-true-branch (stmt),
                                                proc-list),
                                      list (list ('jump,
                                              1 + label-cnt (cinfo)),
                                          list ('dl,
                                              label-cnt (cinfo),
                                              nil,
                                              '(no-op)))),
                            cond-list,
                            if-false-branch (stmt),
                            proc-list),
                 list (list ('dl, 1 + label-cnt (cinfo), nil, '(no-op))))
case = begin-mg
 then add-code (translate (add-code (set-label-alist (translate (make-cinfo (code (cinfo),
                                                                          append (make-label-alist (when-
                                                                                                    label-
                                                                                   label-alist (cinfo)),
                                                                          1 + (1 + label-cnt (cinfo))),
                                                               cond-list,
                                                               begin-body (stmt),
                                                               proc-list),
                                                    label-alist (cinfo)),
                                      list (list ('jump,
                                              1 + label-cnt (cinfo)),
                                          list ('dl,
                                              label-cnt (cinfo),
                                              nil,
                                                  '(push-constant
                                                     (nat 2))),
                                              '(pop-global c-c))),
                               cond-list,
                               when-handler (stmt),
                               proc-list),
                     list (list ('dl, 1 + label-cnt (cinfo), nil, '(no-op))))
    case = proc-call-mg
     then make-cinfo (append (code (cinfo),
                              proc-call-code (cinfo,
                                              stmt,
                                              cond-list,
                                              def-locals (fetch-called-def (stmt,
                                                                            proc-list)),
                                              length (def-cond-locals (fetch-called-def (stmt,
                                                                                         proc-list))))),
                      label-alist (cinfo),
                      label-cnt (cinfo)
                      +    (1 + (1 + length (call-conds (stmt)))))
    case = predefined-proc-call-mg
     then add-code (cinfo,
                    predefined-proc-call-sequence (stmt, label-alist (cinfo)))
    otherwise cinfo endcase
```

THEOREM: signal-translation

```
(car (stmt) = 'signal-mg)
→    (translate (cinfo, cond-list, stmt, proc-list)
        =    make-cinfo (append (code (cinfo),
                                 list (list ('push-constant,
                                             mg-cond-to-p-nat (signalled-condition (stmt),
                                                               cond-list)),
                                       list ('pop-global, 'c-c),
                                       list ('jump,
                                             fetch-label (signalled-condition (stmt),
                                                          label-alist (cinfo))))),
                         label-alist (cinfo),
                         label-cnt (cinfo)))
```

THEOREM: prog2-translation

```
(car (stmt) = 'prog2-mg)
→    (translate (cinfo, cond-list, stmt, proc-list)
        =    translate (translate (cinfo,
                                   cond-list,
                                   prog2-left-branch (stmt),
                               proc-list),
                     cond-list,
                     prog2-right-branch (stmt),
                     proc-list))

THEOREM: loop-translation
 (car (stmt) = 'loop-mg)
 →   (translate (cinfo, cond-list, stmt, proc-list)
       =   discard-label (add-code (translate (make-cinfo (append (code (cinfo),
                                                                  list (list ('dl,
                                                                          label-cnt (cinfo),
                                                                          nil,
                                                                          '(no-op)))),
                                                         cons (cons ('leave,
                                                                    1 + label-cnt (cinfo)),
                                                               label-alist (cinfo)),
                                                         1 + (1 + label-cnt (cinfo))),
                                             cond-list,
                                             loop-body (stmt),
                                             proc-list),
                                   list (list ('jump, label-cnt (cinfo)),
                                       list ('dl,
                                            1 + label-cnt (cinfo),
                                            nil,
                                            '(push-constant
                                              (nat 2))),
                                        '(pop-global c-c)))))

THEOREM: if-translation
 (car (stmt) = 'if-mg)
 →   (translate (cinfo, cond-list, stmt, proc-list)
       =   add-code (translate (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),
                                                               1 + (1 + label-cnt (cinfo))),
                                                   cond-list,
                                                   if-true-branch (stmt),
                                                   proc-list),
                                         list (list ('jump,
                                                  1 + label-cnt (cinfo)),
                                              list ('dl,
                                                  label-cnt (cinfo),
                                                  nil,
                                                  '(no-op)))),
                               cond-list,
                               if-false-branch (stmt),
                               proc-list),
                     list (list ('dl, 1 + label-cnt (cinfo), nil, '(no-op)))))

THEOREM: begin-translation
 (car (stmt) = 'begin-mg)
 →    (translate (cinfo, cond-list, stmt, proc-list)
        =    add-code (translate (add-code (set-label-alist (translate (make-cinfo (code (cinfo),
                                                                              append (make-label-alist (when-
                                                                                                          label-c
                                                                                       label-alist (cinfo)),
                                                                              1 + (1 + label-cnt (cinfo))),
                                                                  cond-list,
                                                                  begin-body (stmt),
                                                                  proc-list),
                                                        label-alist (cinfo)),
                                         list (list ('jump,
                                                  1 + label-cnt (cinfo)),
                                              list ('dl,
                                                  label-cnt (cinfo),
                                                  nil,
                                                  '(push-constant
                                                     (nat 2))),
                                              '(pop-global c-c))),
                               cond-list,
                               when-handler (stmt),
                               proc-list),
                     list (list ('dl, 1 + label-cnt (cinfo), nil, '(no-op)))))

THEOREM: call-translation
 (car (stmt) = 'proc-call-mg)
 →    (translate (cinfo, cond-list, stmt, proc-list)
        =    make-cinfo (append (code (cinfo),
                               proc-call-code (cinfo,
                                               stmt,
                                               cond-list,
                                               def-locals (fetch-called-def (stmt,
                                                                             proc-list)),
                                    length (def-cond-locals (fetch-called-def (stmt,
                                                                                proc-list))))),
                  label-alist (cinfo),
                  label-cnt (cinfo)
                  +   (1 + (1 + length (call-conds (stmt))))))
```

Theorem: predefined-call-translation

```
 (car (stmt) = 'predefined-proc-call-mg)
 →   (translate (cinfo, cond-list, stmt, proc-list)
        =   add-code (cinfo,
                       predefined-proc-call-sequence (stmt,
                                                       label-alist (cinfo))))
```

Event: Disable translate.

Theorem: predefined-proc-call-code-plistp

```
 plistp (predefined-proc-call-sequence (stmt, label-alist))
```

Theorem: not-find-labelp-predefined-proc-call-code

```
 find-labelp (n, predefined-proc-call-sequence (stmt, label-alist)) = f
```

```
;; COMPILATION OF A PROCEDURE
;;
;; Given a procedure def of the form
;; (procedure-defn-mg name (param1 ... paramn) (cond1 ... condi) (local1 ... localj)
;;                          (local-cond1 ... local-condk) body)
;; I make the code for the body in the context of the cinfo
;;   code: nil
;;   label-alist: ((cond1 . 0) (cond2 . 0) .... (local-cond1 . 0) ...)
;;   label-cnt: 1

;; The new scheme of transforming each of the MG locals into a formal of the Piton
;; subroutine eliminates the need to convert them within the code.  I hope it also
;; eliminates the need to store the stack pointer anywhere in the data-segment.
```

Definition:

```
translate-def-body (proc-def, proc-list)
=   add-code (translate (make-cinfo (nil,
                                     cons (cons ('routineerror, 0),
                                            make-label-alist (make-cond-list (proc-def),
                                                              0)),
                                     1),
                          make-cond-list (proc-def),
                          def-body (proc-def),
                          proc-list),
                 list ('(dl 0 nil (no-op)),
                      list ('pop*, data-length (def-locals (proc-def))),
                      '(ret)))
```

Event: Disable translate-def-body.

```
;; Both the MG formals and locals become formals in the Piton world.  This is a better
;; approach because it allows for structured locals just as for structured formals.
```

Definition:

```
translate-def (def, proc-list)
=   append (list (def-name (def),
                  append (listcars (def-locals (def)),
                          listcars (def-formals (def))),
                  nil),
            code (translate-def-body (def, proc-list)))
```

Definition:

```
translate-proc-list1 (proc-list1, proc-list2)
=   if proc-list1 ≃ nil then nil
    else cons (translate-def (car (proc-list1), proc-list2),
               translate-proc-list1 (cdr (proc-list1), proc-list2)) endif
```

Definition:

```
translate-proc-list (proc-list)
=   append (PREDEFINED-PROCEDURE-TRANSLATIONS-LIST,
            translate-proc-list1 (proc-list, proc-list))
```

Event: Disable translate-proc-list.

Theorem: translate-preserves-fields

```
 label-alist (translate (cinfo, cond-list, stmt, proc-list))
 =   label-alist (cinfo)
```

Theorem: code-always-plistp

```
 plistp (code (cinfo))
 →   plistp (code (translate (cinfo, cond-list, stmt, proc-list)))
```

Theorem: translate-preserves-ok-cinfop

```
 ok-cinfop (cinfo) → ok-cinfop (translate (cinfo, cond-list, stmt, proc-list))
```

Event: Disable translate-preserves-ok-cinfop.

Definition:
nearly-equal-cinfos (*x*, *y*)
= ((label-alist (*x*) = label-alist (*y*))
∧ (label-cnt (*x*) = label-cnt (*y*)))

Theorem: nearly-equal-cinfos-translate
(cinfop (*cinfo1*) ∧ cinfop (*cinfo2*) ∧ nearly-equal-cinfos (*cinfo1*, *cinfo2*))
→ nearly-equal-cinfos (translate (*cinfo1*, *cond-list*, *stmt*, *proc-list*),
translate (*cinfo2*, *cond-list*, *stmt*, *proc-list*))

Event: Disable nearly-equal-cinfos-translate.

Theorem: nullify-translate-leaves-nearly-equal
cinfop (*cinfo*)
→ nearly-equal-cinfos (translate (*cinfo*, *cond-list*, *stmt*, *proc-list*),
translate (nullify (*cinfo*), *cond-list*, *stmt*, *proc-list*))

Event: Disable nullify-translate-leaves-nearly-equal.

Theorem: nullify-translate-idempotence
cinfop (*cinfo*)
→ (nullify (translate (nullify (*cinfo*), *cond-list*, *stmt*, *proc-list*))
= nullify (translate (*cinfo*, *cond-list*, *stmt*, *proc-list*)))

Event: Disable nullify-translate-idempotence.

Theorem: nullify-translate-idempotence2
cinfop (*cinfo*)
→ (nullify (translate (*cinfo*, *cond-list*, *stmt*, *proc-list*))
= nullify (translate (nullify (*cinfo*), *cond-list*, *stmt*, *proc-list*)))

Event: Disable nullify-translate-idempotence2.

Theorem: code-doesnt-affect-other-fields
cinfop (*cinfo*)
→ ((label-alist (translate (*cinfo*, *cond-list*, *stmt*, *proc-list*))
= label-alist (translate (nullify (*cinfo*),
cond-list,
stmt,
proc-list)))
∧ (label-cnt (translate (*cinfo*, *cond-list*, *stmt*, *proc-list*))
= label-cnt (translate (nullify (*cinfo*),
cond-list,

```
                                        stmt,
                                        proc-list))))
```

Event: Disable code-doesnt-affect-other-fields.

Theorem: add-code-doesnt-affect-other-fields
(label-alist (add-code (*cinfo*, *code*)) = label-alist (*cinfo*))
∧ (label-cnt (add-code (*cinfo*, *code*)) = label-cnt (*cinfo*))

Theorem: set-label-alist-doesnt-affect-other-fields
(code (set-label-alist (*cinfo*, *label-alist*)) = code (*cinfo*))
∧ (label-cnt (set-label-alist (*cinfo*, *label-alist*)) = label-cnt (*cinfo*))

Theorem: discard-label-doesnt-affect-other-fields
(code (discard-label (*cinfo*)) = code (*cinfo*))
∧ (label-cnt (discard-label (*cinfo*)) = label-cnt (*cinfo*))

Theorem: nullify-cancels-add-code
nullify (add-code (*cinfo*, *code*)) = nullify (*cinfo*)

Theorem: code-add-code-commute
code (add-code (*cinfo*, *cd*)) = append (code (*cinfo*), *cd*)

Theorem: label-alist-set-label-alist
label-alist (set-label-alist (*state*, *label-alist*)) = *label-alist*

Theorem: nullify-doesnt-affect-proc-call-code
proc-call-code (nullify (*cinfo*), *stmt*, *cond-list*, *locals*, *k*)
= proc-call-code (*cinfo*, *stmt*, *cond-list*, *locals*, *k*)

Theorem: nullify-code-nil
code (nullify (*cinfo*)) = **nil**

Definition:
nullify-induction-hint (*cinfo*, *cond-list*, *stmt*, *proc-list*)

```
=   case on car (stmt):
    case = no-op-mg
    then t
    case = signal-mg
     then t
    case = prog2-mg
     then nullify-induction-hint (cinfo,
                                  cond-list,
                                  prog2-left-branch (stmt),
                                  proc-list)
```

```
       ∧   nullify-induction-hint (translate (cinfo,
                                              cond-list,
                                              prog2-left-branch (stmt),
                                              proc-list),
                                   cond-list,
                                   prog2-right-branch (stmt),
                                   proc-list)
       ∧   nullify-induction-hint (translate (nullify (cinfo),
                                              cond-list,
                                              prog2-left-branch (stmt),
                                              proc-list),
                                   cond-list,
                                   prog2-right-branch (stmt),
                                   proc-list)
case = loop-mg
 then nullify-induction-hint (make-cinfo (append (code (cinfo),
                                                  list (list ('dl,
                                                              label-cnt (cinfo),
                                                              nil,
                                                              '(no-op)))),
                                          cons (cons ('leave,
                                                      1 + label-cnt (cinfo)),
                                                label-alist (cinfo)),
                                          1 + (1 + label-cnt (cinfo))),
                              cond-list,
                              loop-body (stmt),
                              proc-list)
       ∧   nullify-induction-hint (make-cinfo (list (list ('dl,
                                                           label-cnt (cinfo),
                                                           nil,
                                                           '(no-op))),
                                               cons (cons ('leave,
                                                           1 + label-cnt (cinfo)),
                                                     label-alist (cinfo)),
                                               1 + (1 + label-cnt (cinfo))),
                                   cond-list,
                                   loop-body (stmt),
                                   proc-list)
case = if-mg
 then nullify-induction-hint (make-cinfo (list (list ('push-local,
                                                      if-condition (stmt)),
                                                '(fetch-temp-stk),
                                                list ('test-bool-and-jump,
                                                      'false,
                                                    label-cnt (cinfo))),
                                         label-alist (cinfo),
                                         1 + (1 + label-cnt (cinfo))),
                           cond-list,
                           if-true-branch (stmt),
                           proc-list)
∧    nullify-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),
                                              1 + (1 + label-cnt (cinfo))),
                                cond-list,
                                if-true-branch (stmt),
                                proc-list)
∧    nullify-induction-hint (add-code (translate (make-cinfo (append (code (cinfo),
                                                                                 list (list ('push-local,
                                                                                           if-condition (stmt)),
                                                                                      '(fetch-temp-stk),
                                                                                      list ('test-bool-and-jum
                                                                                           'false,
                                                                                           label-cnt (cinfo)))),
                                                                       label-alist (cinfo),
                                                                       1 + (1 + label-cnt (cinfo))),
                                                         cond-list,
                                                         if-true-branch (stmt),
                                                         proc-list),
                                            list (list ('jump,
                                                       1 + label-cnt (cinfo)),
                                                  list ('dl,
                                                       label-cnt (cinfo),
                                                       nil,
                                                       '(no-op)))),
                                cond-list,
                                if-false-branch (stmt),
                                proc-list)
∧    nullify-induction-hint (add-code (translate (make-cinfo (list (list ('push-local,
                                                                                 if-condition (stmt)),
                                                                            '(fetch-temp-stk),
                                                                            list ('test-bool-and-jump,
                                                                                 'false,
                                                                           label-cnt (cinfo))),
                                                                  label-alist (cinfo),
                                                                  1 + (1 + label-cnt (cinfo))),
                                                      cond-list,
                                                      if-true-branch (stmt),
                                                      proc-list),
                                            list (list ('jump,
                                                       1 + label-cnt (cinfo)),
                                                  list ('dl,
                                                       label-cnt (cinfo),
                                                       nil,
                                                       '(no-op)))),
                                  cond-list,
                                  if-false-branch (stmt),
                                  proc-list)
case = begin-mg
 then nullify-induction-hint (add-code (set-label-alist (translate (make-cinfo (code (cinfo),
                                                                             append (make-label-alist (whe
                                                                                                       lab
                                                                                     label-alist (cinfo)),
                                                                             1 + (1 + label-cnt (cinfo))),
                                                                 cond-list,
                                                                 begin-body (stmt),
                                                                 proc-list),
                                                       label-alist (cinfo)),
                                        list (list ('jump,
                                                    1 + label-cnt (cinfo)),
                                             list ('dl,
                                                    label-cnt (cinfo),
                                                    nil,
                                                    '(push-constant
                                                       (nat 2))),
                                             '(pop-global c-c))),
                              cond-list,
                              when-handler (stmt),
                              proc-list)
       ∧    nullify-induction-hint (make-cinfo (code (cinfo),
                                              append (make-label-alist (when-labels (stmt),
                                                                         label-cnt (cinfo)),
                                                      label-alist (cinfo)),
                                              1 + (1 + label-cnt (cinfo))),
                                  cond-list,
                                  begin-body (stmt),
                                  proc-list)
```

```
             ∧   nullify-induction-hint (add-code (set-label-alist (translate (nullify (make-cinfo (code (cinfo),
                                                                                                    append (make-lab

                                                                                                           label-alis
                                                                                                    1 + (1 + label-cn
                                                                            cond-list,
                                                                            begin-body (stmt),
                                                                            proc-list),
                                                                 label-alist (cinfo)),
                                                    list (list ('jump,
                                                                1 + label-cnt (cinfo)),
                                                          list ('dl,
                                                                label-cnt (cinfo),
                                                                nil,
                                                                '(push-constant
                                                                  (nat 2))),
                                                          '(pop-global
                                                            c-c))),
                                             cond-list,
                                             when-handler (stmt),
                                             proc-list)
        case = proc-call-mg
         then t
        case = predefined-proc-call-mg
         then t
        otherwise f endcase
```

THEOREM: new-code-prog2-case-induction-hyps

```
 ((car (stmt) = 'prog2-mg) ∧ ok-cinfop (cinfo))
 →   (ok-cinfop (translate (nullify (cinfo),
                            cond-list,
                            prog2-left-branch (stmt),
                            proc-list))
       ∧   ok-cinfop (translate (cinfo,
                                 cond-list,
                                 prog2-left-branch (stmt),
                                 proc-list)))
```

EVENT: Disable nullify.

THEOREM: new-code-prog2-case

```
 ((car (stmt) = 'prog2-mg)
  ∧   ok-cinfop (cinfo)
  ∧   (ok-cinfop (translate (nullify (cinfo),
```

$$
\begin{array}{l}
\hspace{18em}\mathit{cond\text{-}list},\\
\hspace{18em}\text{prog2-left-branch}\,(\mathit{stmt}),\\
\hspace{18em}\mathit{proc\text{-}list}))\\
\hspace{4em}\rightarrow\quad(\text{append}\,(\text{code}\,(\text{translate}\,(\text{nullify}\,(\,\mathit{cinfo}),\\
\hspace{25em}\mathit{cond\text{-}list},\\
\hspace{25em}\text{prog2-left-branch}\,(\mathit{stmt}),\\
\hspace{25em}\mathit{proc\text{-}list})),\\
\hspace{14em}\text{code}\,(\text{translate}\,(\text{nullify}\,(\text{translate}\,(\text{nullify}\,(\,\mathit{cinfo}),\\
\hspace{37em}\mathit{cond\text{-}list},\\
\hspace{37em}\text{prog2-left-branch}\,(\mathit{stmt}),\\
\hspace{37em}\mathit{proc\text{-}list})),\\
\hspace{25em}\mathit{cond\text{-}list},\\
\hspace{25em}\text{prog2-right-branch}\,(\mathit{stmt}),\\
\hspace{25em}\mathit{proc\text{-}list})))\\
\hspace{8em}=\quad\text{code}\,(\text{translate}\,(\text{translate}\,(\text{nullify}\,(\,\mathit{cinfo}),\\
\hspace{29em}\mathit{cond\text{-}list},\\
\hspace{29em}\text{prog2-left-branch}\,(\mathit{stmt}),\\
\hspace{29em}\mathit{proc\text{-}list}),\\
\hspace{22em}\mathit{cond\text{-}list},\\
\hspace{22em}\text{prog2-right-branch}\,(\mathit{stmt}),\\
\hspace{22em}\mathit{proc\text{-}list}))))\\
\wedge\quad(\text{ok-cinfop}\,(\text{translate}\,(\,\mathit{cinfo},\\
\hspace{18em}\mathit{cond\text{-}list},\\
\hspace{18em}\text{prog2-left-branch}\,(\mathit{stmt}),\\
\hspace{18em}\mathit{proc\text{-}list}))\\
\hspace{4em}\rightarrow\quad(\text{append}\,(\text{code}\,(\text{translate}\,(\,\mathit{cinfo},\\
\hspace{25em}\mathit{cond\text{-}list},\\
\hspace{25em}\text{prog2-left-branch}\,(\mathit{stmt}),\\
\hspace{25em}\mathit{proc\text{-}list})),\\
\hspace{14em}\text{code}\,(\text{translate}\,(\text{nullify}\,(\text{translate}\,(\,\mathit{cinfo},\\
\hspace{37em}\mathit{cond\text{-}list},\\
\hspace{37em}\text{prog2-left-branch}\,(\mathit{stmt}),\\
\hspace{37em}\mathit{proc\text{-}list})),\\
\hspace{25em}\mathit{cond\text{-}list},\\
\hspace{25em}\text{prog2-right-branch}\,(\mathit{stmt}),\\
\hspace{25em}\mathit{proc\text{-}list})))\\
\hspace{8em}=\quad\text{code}\,(\text{translate}\,(\text{translate}\,(\,\mathit{cinfo},\\
\hspace{29em}\mathit{cond\text{-}list},\\
\hspace{29em}\text{prog2-left-branch}\,(\mathit{stmt}),\\
\hspace{29em}\mathit{proc\text{-}list}),\\
\hspace{22em}\mathit{cond\text{-}list},\\
\hspace{22em}\text{prog2-right-branch}\,(\mathit{stmt}),\\
\hspace{22em}\mathit{proc\text{-}list}))))\\
\wedge\quad(\text{ok-cinfop}\,(\,\mathit{cinfo})
\end{array}
$$

```
         →   (append (code (cinfo),
                          code (translate (nullify (cinfo),
                                           cond-list,
                                           prog2-left-branch (stmt),
                                           proc-list)))
                 =   code (translate (cinfo,
                                      cond-list,
                                      prog2-left-branch (stmt),
                                      proc-list)))))
 →   (append (code (cinfo),
                  code (translate (nullify (cinfo), cond-list, stmt, proc-list)))
         =   code (translate (cinfo, cond-list, stmt, proc-list)))
```

THEOREM: new-code-loop-case-induction-hyps

```
 ok-cinfop (cinfo)
 →   (ok-cinfop (make-cinfo (list (cons ('dl,
                                        cons (label-cnt (cinfo),
                                              '(nil (no-op))))),
                              cons (cons ('leave, 1 + label-cnt (cinfo)),
                                    label-alist (cinfo)),
                              1 + (1 + label-cnt (cinfo))))
         ∧   ok-cinfop (make-cinfo (append (code (cinfo),
                                            list (cons ('dl,
                                                        cons (label-cnt (cinfo),
                                                              '(nil
                                                                (no-op)))))),
                                    cons (cons ('leave, 1 + label-cnt (cinfo)),
                                          label-alist (cinfo)),
                                    1 + (1 + label-cnt (cinfo)))))
```

THEOREM: new-code-loop-case

```
 ((car (stmt) = 'loop-mg)
  ∧   ok-cinfop (cinfo)
  ∧   (ok-cinfop (make-cinfo (list (cons ('dl,
                                         cons (label-cnt (cinfo),
                                               '(nil (no-op))))),
                               cons (cons ('leave, 1 + label-cnt (cinfo)),
                                     label-alist (cinfo)),
                               1 + (1 + label-cnt (cinfo))))
         →   (append (code (make-cinfo (list (cons ('dl,
                                                   cons (label-cnt (cinfo),
                                                         '(nil (no-op))))),
                                         cons (cons ('leave,
                                                     1 + label-cnt (cinfo)),
                                     label-alist (cinfo)),
                                  1 + (1 + label-cnt (cinfo)))),
                         code (translate (nullify (make-cinfo (list (cons ('dl,
                                                                       cons (label-cnt (cinfo),
                                                                             '(nil
                                                                               (no-op))))),
                                                                  cons (cons ('leave,
                                                                              1 + label-cnt (cinfo)),
                                                                        label-alist (cinfo)),
                                                                  1 + (1 + label-cnt (cinfo)))),
                                                   cond-list,
                                                   loop-body (stmt),
                                                   proc-list)))
     =    code (translate (make-cinfo (list (cons ('dl,
                                                   cons (label-cnt (cinfo),
                                                         '(nil
                                                           (no-op))))),
                                             cons (cons ('leave,
                                                         1 + label-cnt (cinfo)),
                                                   label-alist (cinfo)),
                                             1 + (1 + label-cnt (cinfo))),
                                  cond-list,
                                  loop-body (stmt),
                                  proc-list))))
∧    (ok-cinfop (make-cinfo (append (code (cinfo),
                                     list (cons ('dl,
                                                 cons (label-cnt (cinfo),
                                                       '(nil (no-op)))))),
                             cons (cons ('leave, 1 + label-cnt (cinfo)),
                                   label-alist (cinfo)),
                             1 + (1 + label-cnt (cinfo))))
  →    (append (code (make-cinfo (append (code (cinfo),
                                          list (cons ('dl,
                                                      cons (label-cnt (cinfo),
                                                            '(nil
                                                              (no-op)))))),
                                   cons (cons ('leave,
                                               1 + label-cnt (cinfo)),
                                         label-alist (cinfo)),
                                   1 + (1 + label-cnt (cinfo)))),
               code (translate (nullify (make-cinfo (append (code (cinfo),
                                                             list (cons ('dl,
                                                                         cons (label-cnt (cinfo),
                                                                               '(nil
                                                                                           (no-op))))),
                                                             cons (cons ('leave,
                                                                          1 + label-cnt (cinfo)),
                                                                    label-alist (cinfo)),
                                                             1 + (1 + label-cnt (cinfo)))),
                                        cond-list,
                                        loop-body (stmt),
                                        proc-list)))
             =    code (translate (make-cinfo (append (code (cinfo),
                                                          list (cons ('dl,
                                                                     cons (label-cnt (cinfo),
                                                                           '(nil
                                                                              (no-op))))),
                                                 cons (cons ('leave,
                                                             1 + label-cnt (cinfo)),
                                                       label-alist (cinfo)),
                                                 1 + (1 + label-cnt (cinfo))),
                                    cond-list,
                                    loop-body (stmt),
                                    proc-list)))))
 →    (append (code (cinfo),
                code (translate (nullify (cinfo), cond-list, stmt, proc-list)))
       =    code (translate (cinfo, cond-list, stmt, proc-list)))
```

Theorem: new-code-if-case-induction-hyps

```
 ok-cinfop (cinfo)
 →    (ok-cinfop (add-code (translate (make-cinfo (list (list ('push-local,
                                                               if-condition (stmt)),
                                                          '(fetch-temp-stk),
                                                          list ('test-bool-and-jump,
                                                               'false,
                                                               label-cnt (cinfo))),
                                                      label-alist (cinfo),
                                                      1 + (1 + label-cnt (cinfo))),
                                         cond-list,
                                         if-true-branch (stmt),
                                         proc-list),
                              list (list ('jump, 1 + label-cnt (cinfo)),
                                  cons ('dl,
                                         cons (label-cnt (cinfo),
                                               '(nil (no-op))))))
       ∧    ok-cinfop (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),
                                                     1 + (1 + label-cnt (cinfo))),
                                         cond-list,
                                         if-true-branch (stmt),
                                         proc-list),
                               list (list ('jump, 1 + label-cnt (cinfo)),
                                    cons ('dl,
                                         cons (label-cnt (cinfo),
                                               '(nil (no-op))))))
       ∧    ok-cinfop (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),
                                 1 + (1 + label-cnt (cinfo))))
       ∧    ok-cinfop (make-cinfo (list (list ('push-local,
                                          if-condition (stmt)),
                                      '(fetch-temp-stk),
                                      list ('test-bool-and-jump,
                                          'false,
                                          label-cnt (cinfo))),
                                 label-alist (cinfo),
                                 1 + (1 + label-cnt (cinfo)))))
```

THEOREM: new-code-if-case

```
 ((car (stmt) = 'if-mg)
  ∧    ok-cinfop (cinfo)
  ∧    (ok-cinfop (add-code (translate (make-cinfo (list (list ('push-local,
                                                          if-condition (stmt)),
                                                      '(fetch-temp-stk),
                                                      list ('test-bool-and-jump,
                                                          'false,
                                                          label-cnt (cinfo))),
                                                  label-alist (cinfo),
                                                  1 + (1 + label-cnt (cinfo))),
                                      cond-list,
                                      if-true-branch (stmt),
                                   proc-list),
                        list (list ('jump, 1 + label-cnt (cinfo)),
                             cons ('dl,
                                   cons (label-cnt (cinfo),
                                         '(nil (no-op))))))
→    (append (code (add-code (translate (make-cinfo (list (list ('push-local,
                                                                      if-condition (stmt)),
                                                                 '(fetch-temp-stk),
                                                                 list ('test-bool-and-jump,
                                                                      'false,
                                                                      label-cnt (cinfo))),
                                                            label-alist (cinfo),
                                                            1 + (1 + label-cnt (cinfo))),
                                              cond-list,
                                              if-true-branch (stmt),
                                              proc-list),
                                   list (list ('jump, 1 + label-cnt (cinfo)),
                                        cons ('dl,
                                              cons (label-cnt (cinfo),
                                                    '(nil (no-op))))))),
                code (translate (nullify (add-code (translate (make-cinfo (list (list ('push-local,
                                                                                          if-condition (stmt)),
                                                                                     '(fetch-temp-stk),
                                                                                     list ('test-bool-and-jump,
                                                                                          'false,
                                                                                          label-cnt (cinfo))),
                                                                                label-alist (cinfo),
                                                                                1 + (1 + label-cnt (cinfo))),
                                                                  cond-list,
                                                                  if-true-branch (stmt),
                                                                  proc-list),
                                                       list (list ('jump,
                                                                 1 + label-cnt (cinfo)),
                                                            cons ('dl,
                                                                  cons (label-cnt (cinfo),
                                                                        '(nil
                                                                           (no-op))))))),
                                   cond-list,
                                   if-false-branch (stmt),
                                   proc-list)))
       =    code (translate (add-code (translate (make-cinfo (list (list ('push-local,
                                                                            if-condition (stmt)),
                                                                        '(fetch-temp-stk),
                                                                        list ('test-bool-and-jump,
                                                                              'false,
                                                                              label-cnt (cinfo))),
                                                                     label-alist (cinfo),
                                                                     1 + (1 + label-cnt (cinfo))),
                                                        cond-list,
                                                        if-true-branch (stmt),
                                                        proc-list),
                                             list (list ('jump,
                                                      1 + label-cnt (cinfo)),
                                                 cons ('dl,
                                                        cons (label-cnt (cinfo),
                                                              '(nil
                                                                 (no-op)))))),
                                  cond-list,
                                  if-false-branch (stmt),
                                  proc-list))))
∧   (ok-cinfop (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),
                                                    1 + (1 + label-cnt (cinfo))),
                                       cond-list,
                                       if-true-branch (stmt),
                                       proc-list),
                            list (list ('jump, 1 + label-cnt (cinfo)),
                                 cons ('dl,
                                       cons (label-cnt (cinfo),
                                             '(nil (no-op)))))))
     →     (append (code (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),
                                                              1 + (1 + label-cnt (cinfo))),
                                                 cond-list,
                                                 if-true-branch (stmt),
                                                 proc-list),
                              list (list ('jump, 1 + label-cnt (cinfo)),
                                   cons ('dl,
                                         cons (label-cnt (cinfo),
                                               '(nil (no-op))))))),
          code (translate (nullify (add-code (translate (make-cinfo (append (code (cinfo),
                                                                                    list (list ('push-local,
                                                                                               if-condition (stm
                                                                                          '(fetch-temp-stk)
                                                                                          list ('test-bool-an
                                                                                               'false,
                                                                                               label-cnt (cinfo)
                                                                          label-alist (cinfo),
                                                                          1 + (1 + label-cnt (cinfo))),
                                                              cond-list,
                                                              if-true-branch (stmt),
                                                              proc-list),
                                                   list (list ('jump,
                                                              1 + label-cnt (cinfo)),
                                                        cons ('dl,
                                                              cons (label-cnt (cinfo),
                                                                    '(nil
                                                                       (no-op))))))),
                              cond-list,
                              if-false-branch (stmt),
                              proc-list)))
=    code (translate (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),
                                                              1 + (1 + label-cnt (cinfo))),
                                                 cond-list,
                                                 if-true-branch (stmt),
                                                 proc-list),
                                    list (list ('jump,
                                               1 + label-cnt (cinfo)),
                                         cons ('dl,
                                               cons (label-cnt (cinfo),
                                                     '(nil
                                                        (no-op)))))),
                        cond-list,
                                if-false-branch (stmt),
                                proc-list))))
∧   (ok-cinfop (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),
                            1 + (1 + label-cnt (cinfo))))
  →  (append (code (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),
                                1 + (1 + label-cnt (cinfo)))),
              code (translate (nullify (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),
                                                    1 + (1 + label-cnt (cinfo)))),
                               cond-list,
                               if-true-branch (stmt),
                               proc-list)))
   =   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),
                                    1 + (1 + label-cnt (cinfo))),
                        cond-list,
                        if-true-branch (stmt),
                        proc-list))))
```

```
∧   (ok-cinfop (make-cinfo (list (list ('push-local, if-condition (stmt)),
                                 '(fetch-temp-stk),
                                 list ('test-bool-and-jump,
                                       'false,
                                       label-cnt (cinfo))),
                           label-alist (cinfo),
                           1 + (1 + label-cnt (cinfo))))
     →   (append (code (make-cinfo (list (list ('push-local,
                                                if-condition (stmt)),
                                          '(fetch-temp-stk),
                                          list ('test-bool-and-jump,
                                                'false,
                                                label-cnt (cinfo))),
                                    label-alist (cinfo),
                                    1 + (1 + label-cnt (cinfo)))),
                 code (translate (nullify (make-cinfo (list (list ('push-local,
                                                                   if-condition (stmt)),
                                                             '(fetch-temp-stk),
                                                             list ('test-bool-and-jump,
                                                                   'false,
                                                                   label-cnt (cinfo))),
                                                       label-alist (cinfo),
                                                       1 + (1 + label-cnt (cinfo)))),
                                  cond-list,
                                  if-true-branch (stmt),
                                  proc-list)))
          =   code (translate (make-cinfo (list (list ('push-local,
                                                       if-condition (stmt)),
                                                 '(fetch-temp-stk),
                                                 list ('test-bool-and-jump,
                                                       'false,
                                                       label-cnt (cinfo))),
                                           label-alist (cinfo),
                                           1 + (1 + label-cnt (cinfo))),
                              cond-list,
                              if-true-branch (stmt),
                              proc-list)))))
→   (append (code (cinfo),
            code (translate (nullify (cinfo), cond-list, stmt, proc-list)))
     =   code (translate (cinfo, cond-list, stmt, proc-list)))
```

THEOREM: new-code-begin-case-induction-hyps

```
ok-cinfop (cinfo)
→   (ok-cinfop (add-code (set-label-alist (translate (nullify (make-cinfo (code (cinfo),
                                                                    append (make-label-alist (when-labels (
                                                                                              label-cnt (cin
                                                                                  label-alist (cinfo)),
                                                                    1 + (1 + label-cnt (cinfo)))),
                                                       cond-list,
                                                       begin-body (stmt),
                                                       proc-list),
                                                label-alist (cinfo)),
                                      cons (list ('jump, 1 + label-cnt (cinfo)),
                                            cons (cons ('dl,
                                                        cons (label-cnt (cinfo),
                                                              '(nil
                                                                (push-constant
                                                                 (nat 2))))),
                                                  '((pop-global c-c))))))
      ∧   ok-cinfop (make-cinfo (code (cinfo),
                                 append (make-label-alist (when-labels (stmt),
                                                           label-cnt (cinfo)),
                                         label-alist (cinfo)),
                                 1 + (1 + label-cnt (cinfo))))
      ∧   ok-cinfop (add-code (set-label-alist (translate (make-cinfo (code (cinfo),
                                                                       append (make-label-alist (when-labels (stm
                                                                                                 label-cnt (cinfo))
                                                                               label-alist (cinfo)),
                                                                       1 + (1 + label-cnt (cinfo))),
                                                           cond-list,
                                                           begin-body (stmt),
                                                           proc-list),
                                                 label-alist (cinfo)),
                                       cons (list ('jump, 1 + label-cnt (cinfo)),
                                             cons (cons ('dl,
                                                         cons (label-cnt (cinfo),
                                                               '(nil
                                                                 (push-constant
                                                                  (nat 2))))),
                                                   '((pop-global c-c)))))))
```

```
(prove-lemma new-code-begin-case (rewrite)
      (IMPLIES
(AND
  (equal (car STMT) 'BEGIN-MG)
  (OK-CINFOP CINFO)
```

```
 (IMPLIES
  (OK-CINFOP
   (ADD-CODE
(SET-LABEL-ALIST
 (TRANSLATE
  (NULLIFY (MAKE-CINFO (CODE CINFO)
 (APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT) (LABEL-CNT CINFO))
 (LABEL-ALIST CINFO))
 (ADD1 (ADD1 (LABEL-CNT CINFO)))))
  COND-LIST (BEGIN-BODY STMT) PROC-LIST)
 (LABEL-ALIST CINFO))
(CONS (LIST 'JUMP (ADD1 (LABEL-CNT CINFO)))
   (CONS (CONS 'DL (CONS (LABEL-CNT CINFO) '(NIL (PUSH-CONSTANT (NAT 2)))))
  '((POP-GLOBAL C-C))))))
  (EQUAL
   (APPEND
(CODE
 (ADD-CODE
  (SET-LABEL-ALIST
   (TRANSLATE
(NULLIFY (MAKE-CINFO (CODE CINFO)
   (APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT)
    (LABEL-CNT CINFO))
   (LABEL-ALIST CINFO))
   (ADD1 (ADD1 (LABEL-CNT CINFO)))))
COND-LIST
(BEGIN-BODY STMT)
PROC-LIST)
   (LABEL-ALIST CINFO))
  (CONS (LIST 'JUMP (ADD1 (LABEL-CNT CINFO)))
 (CONS (CONS 'DL
   (CONS (LABEL-CNT CINFO)
  '(NIL (PUSH-CONSTANT (NAT 2)))))
'((POP-GLOBAL C-C))))))
(CODE
 (TRANSLATE
  (NULLIFY
   (ADD-CODE
(SET-LABEL-ALIST
 (TRANSLATE
  (NULLIFY (MAKE-CINFO (CODE CINFO)
 (APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT)
  (LABEL-CNT CINFO))
 (LABEL-ALIST CINFO))
 (ADD1 (ADD1 (LABEL-CNT CINFO)))))
    COND-LIST
    (BEGIN-BODY STMT)
    PROC-LIST)
  (LABEL-ALIST CINFO))
(CONS (LIST 'JUMP (ADD1 (LABEL-CNT CINFO)))
      (CONS (CONS 'DL
  (CONS (LABEL-CNT CINFO)
'(NIL (PUSH-CONSTANT (NAT 2)))))
    '((POP-GLOBAL C-C))))))
    COND-LIST
    (WHEN-HANDLER STMT)
    PROC-LIST)))
      (CODE
(TRANSLATE
  (ADD-CODE
    (SET-LABEL-ALIST
      (TRANSLATE
(NULLIFY (MAKE-CINFO (CODE CINFO)
     (APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT)
       (LABEL-CNT CINFO))
     (LABEL-ALIST CINFO))
     (ADD1 (ADD1 (LABEL-CNT CINFO)))))
COND-LIST
(BEGIN-BODY STMT)
PROC-LIST)
      (LABEL-ALIST CINFO))
    (CONS (LIST 'JUMP (ADD1 (LABEL-CNT CINFO)))
  (CONS (CONS 'DL
      (CONS (LABEL-CNT CINFO)
    '(NIL (PUSH-CONSTANT (NAT 2)))))
'((POP-GLOBAL C-C)))))
  COND-LIST
  (WHEN-HANDLER STMT)
  PROC-LIST))))
  (IMPLIES
    (OK-CINFOP (MAKE-CINFO (CODE CINFO)
   (APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT) (LABEL-CNT CINFO))
   (LABEL-ALIST CINFO))
   (ADD1 (ADD1 (LABEL-CNT CINFO)))))
    (EQUAL
      (APPEND
(CODE (MAKE-CINFO (CODE CINFO)
  (APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT) (LABEL-CNT CINFO))
  (LABEL-ALIST CINFO))
  (ADD1 (ADD1 (LABEL-CNT CINFO)))))
(CODE
  (TRANSLATE
    (NULLIFY (MAKE-CINFO (CODE CINFO)
 (APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT) (LABEL-CNT CINFO))
 (LABEL-ALIST CINFO))
 (ADD1 (ADD1 (LABEL-CNT CINFO)))))
    COND-LIST (BEGIN-BODY STMT) PROC-LIST)))
      (CODE (TRANSLATE (MAKE-CINFO (CODE CINFO)
   (APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT) (LABEL-CNT CINFO))
   (LABEL-ALIST CINFO))
   (ADD1 (ADD1 (LABEL-CNT CINFO))))
       COND-LIST (BEGIN-BODY STMT) PROC-LIST))))
  (IMPLIES
    (OK-CINFOP
      (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))))
     COND-LIST (BEGIN-BODY STMT) PROC-LIST)
  (LABEL-ALIST CINFO))
(CONS (LIST 'JUMP (ADD1 (LABEL-CNT CINFO)))
      (CONS (CONS 'DL (CONS (LABEL-CNT CINFO) '(NIL (PUSH-CONSTANT (NAT 2)))))
    '((POP-GLOBAL C-C))))))
    (EQUAL
      (APPEND
(CODE
  (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))))
 COND-LIST
 (BEGIN-BODY STMT)
 PROC-LIST)
      (LABEL-ALIST CINFO))
    (CONS (LIST 'JUMP (ADD1 (LABEL-CNT CINFO)))
  (CONS (CONS 'DL
      (CONS (LABEL-CNT CINFO)
    ’(NIL (PUSH-CONSTANT (NAT 2)))))
’((POP-GLOBAL C-C))))))
(CODE
  (TRANSLATE
    (NULLIFY
      (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))))
     COND-LIST
     (BEGIN-BODY STMT)
     PROC-LIST)
  (LABEL-ALIST CINFO))
(CONS (LIST ’JUMP (ADD1 (LABEL-CNT CINFO)))
      (CONS (CONS ’DL
  (CONS (LABEL-CNT CINFO)
’(NIL (PUSH-CONSTANT (NAT 2)))))
    ’((POP-GLOBAL C-C))))))
    COND-LIST
    (WHEN-HANDLER STMT)
    PROC-LIST)))
      (CODE
(TRANSLATE
  (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))))
 COND-LIST
 (BEGIN-BODY STMT)
 PROC-LIST)
      (LABEL-ALIST CINFO))
    (CONS (LIST ’JUMP (ADD1 (LABEL-CNT CINFO)))
  (CONS (CONS ’DL
      (CONS (LABEL-CNT CINFO)
    ’(NIL (PUSH-CONSTANT (NAT 2)))))
’((POP-GLOBAL C-C)))))
  COND-LIST
  (WHEN-HANDLER STMT)
  PROC-LIST)))))
(EQUAL (APPEND (CODE CINFO)
       (CODE (TRANSLATE (NULLIFY CINFO)
COND-LIST STMT PROC-LIST)))
       (CODE (TRANSLATE CINFO COND-LIST STMT PROC-LIST))))
    ((INSTRUCTIONS (disable add-code set-label-alist)
       PROMOTE (DEMOTE 3) (DIVE 1 1) (REWRITE NEW-CODE-BEGIN-CASE-INDUCTION-HYPS) UP S TOP P
       (DEMOTE 3) (DIVE 1 1) (REWRITE NEW-CODE-BEGIN-CASE-INDUCTION-HYPS) UP S TOP PROMOTE
       (DEMOTE 3) (DIVE 1 1) (REWRITE NEW-CODE-BEGIN-CASE-INDUCTION-HYPS) UP S TOP PROMOTE
       (DIVE 2 1) (REWRITE BEGIN-TRANSLATION) UP (REWRITE CODE-ADD-CODE-COMMUTE) (DIVE 1) =
       UP (REWRITE ASSOCIATIVITY-OF-APPEND) (DIVE 1) (REWRITE CODE-ADD-CODE-COMMUTE) (DIVE 1
       (REWRITE SET-LABEL-ALIST-DOESNT-AFFECT-OTHER-FIELDS) = (DROP 4) UP
       (REWRITE ASSOCIATIVITY-OF-APPEND) UP (REWRITE ASSOCIATIVITY-OF-APPEND) TOP (REWRITE A
       (DIVE 1 1) (REWRITE BEGIN-TRANSLATION) UP (REWRITE CODE-ADD-CODE-COMMUTE) (DIVE 1) (S
       S TOP (DEMOTE 3) (DIVE 1 2) (S-PROP NULLIFY) S TOP PROMOTE (DIVE 1 1) = (DROP 3) TOP
       (BASH (ENABLE NULLIFY ADD-CODE TRANSLATE-PRESERVES-FIELDS APPEND-REWRITE2 SET-LABEL-A
       PROMOTE (DIVE 2 1 1) (DIVE 1 3) (REWRITE CODE-DOESNT-AFFECT-OTHER-FIELDS)
       TOP (PROVE (ENABLE NULLIFY)))))
```

THEOREM: new-code-appended-to-old
ok-cinfop (*cinfo*)
→ (append (code (*cinfo*),
code (translate (nullify (*cinfo*), *cond-list*, *stmt*, *proc-list*)))
= code (translate (*cinfo*, *cond-list*, *stmt*, *proc-list*)))

EVENT: Disable new-code-appended-to-old.

THEOREM: new-code-appended-to-old1
ok-cinfop (*cinfo*)
→ (code (translate (*cinfo*, *cond-list*, *stmt*, *proc-list*))
= append (code (*cinfo*),
code (translate (nullify (*cinfo*), *cond-list*, *stmt*, *proc-list*))))

EVENT: Disable new-code-appended-to-old1.

DEFINITION:
collect-labels (*codelist*)
= **if** *codelist* ≃ **nil then nil**
elseif caar (*codelist*) = `'dl`
then cons (cadar (*codelist*), collect-labels (cdr (*codelist*)))
else collect-labels (cdr (*codelist*)) **endif**

THEOREM: collect-labels-plistp
plistp (collect-labels (*lst*))

Theorem: collect-labels-distributes
collect-labels (append (*code1*, *code2*))
= append (collect-labels (*code1*), collect-labels (*code2*))

Definition:
all-labels-unique (*codelist*) = no-duplicates (collect-labels (*codelist*))

Event: Disable all-labels-unique.

Theorem: all-labels-unique-append
all-labels-unique (append (x, y))
→ (all-labels-unique (x) ∧ all-labels-unique (y))

Theorem: all-labels-unique-reduction
(¬ all-labels-unique (y)) → (¬ all-labels-unique (append (x, y)))

Event: Disable all-labels-unique-reduction.

Theorem: all-labels-unique-reduction2
(¬ all-labels-unique (y)) → (¬ all-labels-unique (cons (x, y)))

Event: Disable all-labels-unique-reduction2.

Theorem: find-labelp-rewrites-to-member
find-labelp (*lab*, *code*) = (*lab* ∈ collect-labels (*code*))

Event: Disable find-labelp-rewrites-to-member.

Theorem: all-labels-unique-reduction3
(find-labelp (*lab*, *code1*) ∧ find-labelp (*lab*, *code2*))
→ (¬ all-labels-unique (append (*code1*, *code2*)))

Event: Disable all-labels-unique-reduction3.

Theorem: no-duplicates-append-list
no-duplicates (append (*lst*, cons (x, cons (y, *lst2*))))
→ no-duplicates (append (*lst*, list (y)))

Theorem: no-duplicates-append-list2
no-duplicates (append (*lst*, cons (y, *lst2*)))
→ no-duplicates (append (*lst*, list (y)))

THEOREM: labels-unique-append2
all-labels-unique (append (*lst1*, cons (*x*, cons (*y*, *lst2*))))
→ all-labels-unique (append (*lst1*, list (*y*)))

THEOREM: find-labelp-member-collect-labels
find-labelp (*x*, *code*) → (*x* ∈ collect-labels (*code*))

DEFINITION:
label-hole-big-enough (*cinfo*, *cond-list*, *stmt*, *proc-list*, *y*)
= all-labels-unique (append (code (translate (*cinfo*,
cond-list,
stmt,
proc-list)),
y))

THEOREM: labels-unique-not-find-labelp
(all-labels-unique (append (*code1*, *code2*)) ∧ find-labelp (*label*, *code2*))
→ (¬ find-labelp (*label*, *code1*))

THEOREM: labels-unique-not-find-labelp1
all-labels-unique (append (*lst*, list (list (`'dl`, *label*, **nil**, *w*))))
→ (find-labelp (*label*, *lst*) = **f**)

DEFINITION:
ok-cond-list (*lst*)
= **if** *lst* ≃ **nil** **then** *lst* = **nil**
else (ok-mg-namep (car (*lst*))
∨ (car (*lst*) ∈ `'(leave routineerror)`))
∧ ok-cond-list (cdr (*lst*)) **endif**

THEOREM: identifier-plistp-make-cond-list-ok
identifier-plistp (*lst*) → ok-cond-list (*lst*)

THEOREM: make-cond-list-ok
((car (*stmt*) = `'proc-call-mg`)
∧ ok-mg-statement (*stmt*, *cond-list*, *name-alist*, *proc-list*)
∧ ok-mg-def-plistp (*proc-list*))
→ ok-cond-list (make-cond-list (fetch-called-def (*stmt*, *proc-list*)))

THEOREM: cond-subsetp-preserves-ok-mg-statep
(cond-subsetp (*r-cond-list*, *t-cond-list*)
∧ (cc (*mg-state*) ≠ `'leave`)
∧ ok-mg-statep (*mg-state*, *r-cond-list*))
→ ok-mg-statep (*mg-state*, *t-cond-list*)

Definition:
ok-translation-parameters$\,(cinfo,\ cond\text{-}list,\ stmt,\ proc\text{-}list,\ y)$
$=\quad$ (ok-cinfop$\,(cinfo)$
$\quad\wedge\quad$ ok-cond-list$\,(cond\text{-}list)$
$\quad\wedge\quad$ label-hole-big-enough$\,(cinfo,\ cond\text{-}list,\ stmt,\ proc\text{-}list,\ y)$)

Theorem: label-cnt-monotonic
label-cnt (translate$\,(cinfo,\ cond\text{-}list,\ stmt,\ proc\text{-}list)$) $\not<$ label-cnt$\,(cinfo)$

Theorem: label-cnt-monotonic2
($n <$ label-cnt$\,(cinfo)$)
$\rightarrow\quad$ ($n <$ label-cnt (translate$\,(cinfo,\ cond\text{-}list,\ stmt,\ proc\text{-}list)$))

Theorem: label-cnt-monotonic3
($n <$ label-cnt$\,(cinfo)$)
$\rightarrow\quad$ (($n <$ label-cnt (translate$\,(cinfo,\ cond\text{-}list,\ stmt,\ proc\text{-}list)$)) $=$ **t**)

Theorem: label-cnt-add1-add1-monotonic
(($n < lc$) $\wedge$ (label-cnt$\,(cinfo) = (1 + (1 + lc))$))
$\rightarrow\quad$ (($n <$ label-cnt (translate$\,(cinfo,\ cond\text{-}list,\ stmt,\ proc\text{-}list)$)) $=$ **t**)

Theorem: label-cnt-monotonic-cond-conversion
($n < lc$)
$\rightarrow\quad$ (find-labelp (n, cond-conversion$\,(actual\text{-}conds,\ lc,\ cond\text{-}list,\ label\text{-}alist)$)
$\qquad = \quad$ **f**)

Theorem: not-find-labelp-push-parameters-code
find-labelp (n, push-parameters-code$\,(locals,\ actuals)$) $=$ **f**

Event: Disable not-find-labelp-push-parameters-code.

Theorem: find-labelp-monotonic-lessp
(($n <$ label-cnt$\,(cinfo)$) $\wedge$ ($\neg$ find-labelp (n, code$\,(cinfo)$)))
$\rightarrow\quad$ (find-labelp (n, code (translate$\,(cinfo,\ cond\text{-}list,\ stmt,\ proc\text{-}list)$)) $=$ **f**)

```
;; The following definition is used only in the proof of procedure-calls.
```

Definition:
label-cnt-big-enough$\,(lc,\ code)$
$=\quad$ **if** $code \simeq$ **nil then t**
elseif caar$\,(code)$ $=$ `'dl`
then (cadar$\,(code) < lc$) $\wedge$ label-cnt-big-enough (lc, cdr$\,(code)$)
else label-cnt-big-enough (lc, cdr$\,(code)$) **endif**

Definition:
cond-conversion-induction-hint (*lst*, *n*)
= **if** *lst* $\simeq$ **nil then t**
else cond-conversion-induction-hint (cdr (*lst*), 1 + *n*) **endif**

Theorem: label-count-big-enough-not-find-labelp
label-cnt-big-enough (*lc*, *code*) → (find-labelp (*lc*, *code*) = **f**)

Theorem: greater-label-count-big-enough
(label-cnt-big-enough (*n*, *code*) $\land$ ($n \leq m$))
→ label-cnt-big-enough (*m*, *code*)

Theorem: label-cnt-big-enough-distributes
label-cnt-big-enough (*lc*, append (*lst1*, *lst2*))
= (label-cnt-big-enough (*lc*, *lst1*) $\land$ label-cnt-big-enough (*lc*, *lst2*))

Theorem: label-cnt-lessp1
(*n* < label-cnt (*cinfo*))
→ ((*n* < label-cnt (translate (*cinfo*, *cond-list*, *stmt*, *proc-list*))) = **t**)

Theorem: label-cnt-big-enough-distributes2
(label-cnt-big-enough (*n*, *lst1*) $\land$ label-cnt-big-enough (*n*, *lst2*))
→ label-cnt-big-enough (*n*, append (*lst1*, *lst2*))

Theorem: label-cnt-big-enough-for-push-actuals-code
label-cnt-big-enough (*n*, push-actuals-code (*actuals*))

Theorem: label-cnt-big-enough-for-push-local-array-values-code
label-cnt-big-enough (*n*, push-local-array-values-code (*array-value*))

Theorem: label-cnt-big-enough-for-push-locals-values-code
label-cnt-big-enough (*n*, push-locals-values-code (*actuals*))

Theorem: label-cnt-big-enough-for-push-locals-addresses-code
label-cnt-big-enough (*n*, push-locals-addresses-code (*actuals*, *m*))

Theorem: label-cnt-big-enough-for-cond-conversion
label-cnt-big-enough (*lc* + (1 + (1 + length (*lst*))),
cond-conversion (*lst*, 1 + (1 + *lc*), *cond-list*, *label-alist*))

Theorem: label-cnt-big-enough-for-proc-call-code
label-cnt-big-enough (label-cnt (*cinfo*), code (*cinfo*))
→ label-cnt-big-enough (label-cnt (*cinfo*)
+ (1 + (1 + length (call-conds (*stmt*)))),
proc-call-code (*cinfo*, *stmt*, *cond-list*, *locals*, *k*))

THEOREM: label-cnt-big-enough-for-predefined-proc-call-code
label-cnt-big-enough (*n*, predefined-proc-call-sequence (*stmt*, *label-alist*))

THEOREM: label-cnt-stays-big-enough
label-cnt-big-enough (label-cnt (*cinfo*), code (*cinfo*))
→ label-cnt-big-enough (label-cnt (translate (*cinfo*,
cond-list,
stmt,
proc-list)),
code (translate (*cinfo*, *cond-list*, *stmt*, *proc-list*)))

THEOREM: label-cnt-big-enough-add1
label-cnt-big-enough (*x*, *y*) → label-cnt-big-enough ($1 + x$, *y*)

THEOREM: lesser-label-doesnt-disturb-no-duplicates
(no-duplicates (*lst*) ∧ ($x \notin$ *lst*)) → no-duplicates (append (*lst*, list (*x*)))

THEOREM: find-labelp-reduces-to-member
find-labelp (*x*, *lst*) = ($x \in$ collect-labels (*lst*))

THEOREM: member-labels-unique-not-find-labelp
(all-labels-unique (append (*code*, *code2*)) ∧ find-labelp (*label*, *code2*))
→ (¬ find-labelp (*label*, *code*))

THEOREM: no-duplicates-right-cons-reduction
no-duplicates (collect-labels (*lst*))
→ (no-duplicates (append (collect-labels (*lst*), list (*x*)))
= (¬ find-labelp (*x*, *lst*)))

THEOREM: label-cnt-big-enough-not-find-labelp
label-cnt-big-enough (*lc*, *code*) → (find-labelp (*lc*, *code*) = **f**)

THEOREM: not-member-cond-conversion
($n <$ *lc*)
→ (($n \in$ collect-labels (cond-conversion (*conds*, *lc*, *cond-list*, *label-alist*)))
= **f**)

THEOREM: no-duplicates-cond-conversion
no-duplicates (collect-labels (cond-conversion (*conds*, *lc*, *cond-list*, *label-alist*)))

THEOREM: no-duplicates-cond-conversion-base-case
no-duplicates (append (collect-labels (cond-conversion (*conds*,
$1 + (1 + lc)$,
cond-list,
label-alist)),
list ($1 + lc$)))

Theorem: no-duplicates-proc-call
(no-duplicates (collect-labels (*code*)) ∧ label-cnt-big-enough (*lc*, *code*))
→ no-duplicates (append (collect-labels (*code*),
cons (*lc*,
append (collect-labels (cond-conversion (*conds*,
1 + (1 + *lc*),
cond-list,
label-alist)),
list (1 + *lc*)))))

Theorem: collect-labels-push-actuals-code-nil
collect-labels (push-actuals-code (*actuals*)) = **nil**

Theorem: collect-labels-push-local-array-values-code-nil
collect-labels (push-local-array-values-code (*array-value*)) = **nil**

Theorem: collect-labels-push-locals-values-code-nil
collect-labels (push-locals-values-code (*actuals*)) = **nil**

Theorem: collect-labels-push-locals-addresses-code-nil
collect-labels (push-locals-addresses-code (*actuals*, *m*)) = **nil**

Theorem: collect-labels-predefined-proc-call-code-nil
collect-labels (predefined-proc-call-sequence (*stmt*, *label-alist*)) = **nil**

Theorem: collect-labels-strip-label
collect-labels (cons (cons (`'dl`, cons (*label*, *x*)), *y*))
= cons (*label*, collect-labels (*y*))

Theorem: labels-unique-loop-case
((car (*stmt*) = `'loop-mg`)
∧ no-duplicates (collect-labels (code (*cinfo*)))
∧ label-cnt-big-enough (label-cnt (*cinfo*), code (*cinfo*))
∧ ((no-duplicates (collect-labels (code (make-cinfo (append (code (*cinfo*),
list (cons (`'dl`,
cons (label-cnt (*cinfo*),
`'(nil`
`(no-op)`))))),
cons (cons (`'leave`,
1 + label-cnt (*cinfo*)),
label-alist (*cinfo*)),
1 + (1 + label-cnt (*cinfo*))))))
∧ label-cnt-big-enough (label-cnt (make-cinfo (append (code (*cinfo*),
list (cons (`'dl`,
cons (label-cnt (*cinfo*),

```
                                                                         '(nil
                                                                            (no-op))))),
                                                   cons (cons ('leave,
                                                               1 + label-cnt (cinfo)),
                                                          label-alist (cinfo)),
                                                   1 + (1 + label-cnt (cinfo)))),
                               code (make-cinfo (append (code (cinfo),
                                                       list (cons ('dl,
                                                                   cons (label-cnt (cinfo),
                                                                        '(nil
                                                                           (no-op))))),
                                               cons (cons ('leave,
                                                           1 + label-cnt (cinfo)),
                                                    label-alist (cinfo)),
                                               1 + (1 + label-cnt (cinfo))))))
      →   no-duplicates (collect-labels (code (translate (make-cinfo (append (code (cinfo),
                                                                       list (cons ('dl,
                                                                                   cons (label-cnt (cinfo),
                                                                                        '(nil
                                                                                           (no-op))))),
                                                               cons (cons ('leave,
                                                                           1 + label-cnt (cinfo)),
                                                                    label-alist (cinfo)),
                                                               1 + (1 + label-cnt (cinfo))),
                                                    cond-list,
                                                    loop-body (stmt),
                                                    proc-list))))))
 →   no-duplicates (collect-labels (code (translate (cinfo,
                                              cond-list,
                                              stmt,
                                              proc-list))))
```

Event: Disable labels-unique-loop-case.

Theorem: labels-unique-if-case-hyps1

```
 ((car (stmt) = 'if-mg)
  ∧   no-duplicates (collect-labels (code (cinfo)))
  ∧   label-cnt-big-enough (label-cnt (cinfo), code (cinfo)))
 →   (no-duplicates (collect-labels (code (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),
                                                     1 + (1 + label-cnt (cinfo))))))
       ∧   label-cnt-big-enough (label-cnt (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),
                                                      1 + (1 + label-cnt (cinfo)))),
                                code (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),
                                                  1 + (1 + label-cnt (cinfo))))))
```

EVENT: Disable labels-unique-if-case-hyps1.

THEOREM: label-cnt-big-enough-not-member
label-cnt-big-enough (*lc*, *code*) → (*lc* ∉ collect-labels (*code*))

THEOREM: labels-unique-if-case-hyps2

```
 ((car (stmt) = 'if-mg)
  ∧   no-duplicates (collect-labels (code (cinfo)))
  ∧   label-cnt-big-enough (label-cnt (cinfo), code (cinfo))
  ∧   no-duplicates (collect-labels (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),
                                                              1 + (1 + label-cnt (cinfo))),
                                                  cond-list,
                                                  if-true-branch (stmt),
                                                  proc-list)))))
 →    (no-duplicates (collect-labels (code (add-code (translate (make-cinfo (append (code (cinfo),
                                                                                          list (list ('push-local,
                                                                                                     if-condition (stmt))
                                                                                                '(fetch-temp-stk),
                                                                                                list ('test-bool-and-j
                                                                                                     'false,
                                                                                                     label-cnt (cinfo)))),
                                                                                label-alist (cinfo),
                                                                                1 + (1 + label-cnt (cinfo))),
                                                                  cond-list,
                                                                  if-true-branch (stmt),
                                                                  proc-list),
                                                      list (list ('jump,
                                                                1 + label-cnt (cinfo)),
                                                           cons ('dl,
                                                                 cons (label-cnt (cinfo),
                                                                        '(nil
                                                                           (no-op))))))))
∧   label-cnt-big-enough (label-cnt (add-code (translate (make-cinfo (append (code (cinfo),
                                                                                            list (list ('push-local,
                                                                                                      if-condition (stmt)
                                                                                                 '(fetch-temp-stk),
                                                                                                 list ('test-bool-and-
                                                                                                      'false,
                                                                                                      label-cnt (cinfo)))
                                                                                  label-alist (cinfo),
                                                                                  1 + (1 + label-cnt (cinfo))),
                                                                   cond-list,
                                                                   if-true-branch (stmt),
                                                                   proc-list),
                                                       list (list ('jump,
                                                                 1 + label-cnt (cinfo)),
                                                            cons ('dl,
                                                                   cons (label-cnt (cinfo),
                                                                          '(nil
                                                                             (no-op))))))),
                               code (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),
                                                                             1 + (1 + label-cnt (cinfo))),
                                                                cond-list,
                                                                if-true-branch (stmt),
                                                                proc-list),
                                                     list (list ('jump,
                                                              1 + label-cnt (cinfo)),
                                                          cons ('dl,
                                                               cons (label-cnt (cinfo),
                                                                      '(nil
                                                                        (no-op))))))))
```

EVENT: Disable labels-unique-if-case-hyps2.

EVENT: Disable label-cnt-big-enough-not-member.

THEOREM: labels-unique-if-case

```
 ((car (stmt) = 'if-mg)
  ∧   no-duplicates (collect-labels (code (cinfo)))
  ∧   label-cnt-big-enough (label-cnt (cinfo), code (cinfo))
  ∧   ((no-duplicates (collect-labels (code (add-code (translate (make-cinfo (append (code (cinfo),
                                                                                          list (list ('push-local,
                                                                                                    if-condition (stmt)
                                                                                               '(fetch-temp-stk),
                                                                                               list ('test-bool-and-
                                                                                                    'false,
                                                                                                    label-cnt (cinfo))))
                                                                                 label-alist (cinfo),
                                                                                 1 + (1 + label-cnt (cinfo))),
                                                                    cond-list,
                                                                    if-true-branch (stmt),
                                                                    proc-list),
                                                         list (list ('jump,
                                                                  1 + label-cnt (cinfo)),
                                                              cons ('dl,
                                                                    cons (label-cnt (cinfo),
                                                                          '(nil
                                                                             (no-op))))))))
        ∧   label-cnt-big-enough (label-cnt (add-code (translate (make-cinfo (append (code (cinfo),
                                                                                            list (list ('push-local,
                                                                                                     if-condition (stmt
                                                                                               '(fetch-temp-stk),
                                                                                                list ('test-bool-and
                                                                                                     'false,
                                                                                                     label-cnt (cinfo))
```

```
                                                                     label-alist (cinfo),
                                                                     1 + (1 + label-cnt (cinfo))),
                                                         cond-list,
                                                         if-true-branch (stmt),
                                                         proc-list),
                                               list (list ('jump,
                                                        1 + label-cnt (cinfo)),
                                                   cons ('dl,
                                                         cons (label-cnt (cinfo),
                                                               '(nil
                                                                 (no-op))))))),
                           code (add-code (translate (make-cinfo (append (code (cinfo),
                                                                         list (list ('push-local,
                                                                                     if-condition (stmt)),
                                                                               '(fetch-temp-stk),
                                                                               list ('test-bool-and-jum
                                                                                     'false,
                                                                                     label-cnt (cinfo)))),
                                                                 label-alist (cinfo),
                                                                 1 + (1 + label-cnt (cinfo))),
                                                     cond-list,
                                                     if-true-branch (stmt),
                                                     proc-list),
                                           list (list ('jump,
                                                   1 + label-cnt (cinfo)),
                                               cons ('dl,
                                                     cons (label-cnt (cinfo),
                                                           '(nil
                                                             (no-op)))))))))
→    no-duplicates (collect-labels (code (translate (add-code (translate (make-cinfo (append (code (cinfo),
                                                                                            list (list ('pus
                                                                                                     if-co
                                                                                                '(fetch
                                                                                                list ('tes
                                                                                                     'fal
                                                                                                     labe
                                                                                  label-alist (cinfo),
                                                                                  1 + (1 + label-cnt (ci
                                                                      cond-list,
                                                                      if-true-branch (stmt),
                                                                      proc-list),
                                                            list (list ('jump,
                                                                    1 + label-cnt (cinfo)),
                                                                cons ('dl,
                                                                           cons (label-cnt (cinfo),
                                                                                '(nil
                                                                                   (no-op)))))),
                                                      cond-list,
                                                      if-false-branch (stmt),
                                                      proc-list)))))
∧   ((no-duplicates (collect-labels (code (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),
                                                    1 + (1 + label-cnt (cinfo))))))
      ∧   label-cnt-big-enough (label-cnt (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),
                                                      1 + (1 + label-cnt (cinfo)))),
                                code (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),
                                                  1 + (1 + label-cnt (cinfo))))))
     →   no-duplicates (collect-labels (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),
                                                                  1 + (1 + label-cnt (cinfo))),
                                                      cond-list,
                                                      if-true-branch (stmt),
                                                  proc-list))))))
 →   no-duplicates (collect-labels (code (translate (cinfo,
                                                  cond-list,
                                                  stmt,
                                                  proc-list))))
```

EVENT: Disable labels-unique-if-case.

THEOREM: labels-unique-begin-case-hyps

```
 ((car (stmt) = 'begin-mg)
  ∧   no-duplicates (collect-labels (code (cinfo)))
  ∧   label-cnt-big-enough (label-cnt (cinfo), code (cinfo))
  ∧   no-duplicates (collect-labels (code (translate (make-cinfo (code (cinfo),
                                                             append (make-label-alist (when-labels (stmt),
                                                                                       label-cnt (cinfo)),
                                                                     label-alist (cinfo)),
                                                             1 + (1 + label-cnt (cinfo))),
                                                  cond-list,
                                                  begin-body (stmt),
                                                  proc-list)))))
 →   (no-duplicates (collect-labels (code (add-code (set-label-alist (translate (make-cinfo (code (cinfo),
                                                                                          append (make-label-al

                                                                                                  label-alist (ci
                                                                                          1 + (1 + label-cnt (ci
                                                                               cond-list,
                                                                               begin-body (stmt),
                                                                               proc-list),
                                                                   label-alist (cinfo)),
                                                   cons (list ('jump,
                                                               1 + label-cnt (cinfo)),
                                                         cons (cons ('dl,
                                                                     cons (label-cnt (cinfo),
                                                                           '(nil
                                                                             (push-constant
                                                                              (nat
                                                                               2))))),
                                                               '((pop-global
                                                                  c-c))))))))
     ∧   label-cnt-big-enough (label-cnt (add-code (set-label-alist (translate (make-cinfo (code (cinfo),
                                                                                          append (make-label-a

                                                                                                  label-alist (c
                                                                                          1 + (1 + label-cnt (c
                                                                             cond-list,
                                                                             begin-body (stmt),
                                                                             proc-list),
                                                                   label-alist (cinfo)),
                                                    cons (list ('jump,
                                                              1 + label-cnt (cinfo)),
                                                        cons (cons ('dl,
                                                                     cons (label-cnt (cinfo),
                                                                           '(nil
                                                                             (push-constant
                                                                               (nat
                                                                                2))))),
                                                                '((pop-global
                                                                   c-c)))))),
                                code (add-code (set-label-alist (translate (make-cinfo (code (cinfo),
                                                                                     append (make-label-alist

                                                                                              label-alist (cinfo
                                                                                     1 + (1 + label-cnt (cinfo
                                                                         cond-list,
                                                                         begin-body (stmt),
                                                                         proc-list),
                                                               label-alist (cinfo)),
                                                cons (list ('jump,
                                                          1 + label-cnt (cinfo)),
                                                      cons (cons ('dl,
                                                                 cons (label-cnt (cinfo),
                                                                       '(nil
                                                                         (push-constant
                                                                           (nat
                                                                            2))))),
                                                            '((pop-global
                                                               c-c)))))))
```

EVENT: Disable labels-unique-begin-case-hyps.

```
(prove-lemma labels-unique-begin-case (rewrite)
       (IMPLIES
 (AND (equal (car STMT) 'BEGIN-MG)
      (NO-DUPLICATES (COLLECT-LABELS (CODE CINFO)))
      (LABEL-CNT-BIG-ENOUGH (LABEL-CNT CINFO) (CODE CINFO))
      (IMPLIES
(AND
                  (NO-DUPLICATES
    (COLLECT-LABELS
      (CODE
(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))))
       COND-LIST
       (BEGIN-BODY STMT)
       PROC-LIST)
    (LABEL-ALIST CINFO))
  (CONS (LIST 'JUMP (ADD1 (LABEL-CNT CINFO)))
(CONS (CONS 'DL
    (CONS (LABEL-CNT CINFO)
  '(NIL (PUSH-CONSTANT (NAT 2)))))
      '((POP-GLOBAL C-C))))))))
  (LABEL-CNT-BIG-ENOUGH
    (LABEL-CNT
      (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))))
     COND-LIST
     (BEGIN-BODY STMT)
     PROC-LIST)
  (LABEL-ALIST CINFO))
(CONS (LIST 'JUMP (ADD1 (LABEL-CNT CINFO)))
      (CONS (CONS 'DL
  (CONS (LABEL-CNT CINFO)
'(NIL (PUSH-CONSTANT (NAT 2)))))
    '((POP-GLOBAL C-C))))))
    (CODE
      (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))))
     COND-LIST
     (BEGIN-BODY STMT)
     PROC-LIST)
  (LABEL-ALIST CINFO))
(CONS (LIST 'JUMP (ADD1 (LABEL-CNT CINFO)))
      (CONS (CONS 'DL
  (CONS (LABEL-CNT CINFO)
'(NIL (PUSH-CONSTANT (NAT 2)))))
    '((POP-GLOBAL C-C))))))))
(NO-DUPLICATES
  (COLLECT-LABELS
    (CODE
      (TRANSLATE
(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))))
       COND-LIST
       (BEGIN-BODY STMT)
       PROC-LIST)
    (LABEL-ALIST CINFO))
  (CONS (LIST 'JUMP (ADD1 (LABEL-CNT CINFO)))
(CONS (CONS 'DL
    (CONS (LABEL-CNT CINFO)
  '(NIL (PUSH-CONSTANT (NAT 2)))))
      '((POP-GLOBAL C-C)))))
COND-LIST
(WHEN-HANDLER STMT)
PROC-LIST)))))
      (IMPLIES
(AND
  (NO-DUPLICATES
    (COLLECT-LABELS
      (CODE (MAKE-CINFO (CODE CINFO)
(APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT)
  (LABEL-CNT CINFO))
(LABEL-ALIST CINFO))
(ADD1 (ADD1 (LABEL-CNT CINFO)))))))
  (LABEL-CNT-BIG-ENOUGH
    (LABEL-CNT (MAKE-CINFO (CODE CINFO)
   (APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT)
     (LABEL-CNT CINFO))
   (LABEL-ALIST CINFO))
   (ADD1 (ADD1 (LABEL-CNT CINFO)))))
    (CODE (MAKE-CINFO (CODE CINFO)
      (APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT)
(LABEL-CNT CINFO))
      (LABEL-ALIST CINFO))
      (ADD1 (ADD1 (LABEL-CNT CINFO)))))))
(NO-DUPLICATES
  (COLLECT-LABELS
    (CODE (TRANSLATE (MAKE-CINFO (CODE CINFO)
 (APPEND (MAKE-LABEL-ALIST (WHEN-LABELS STMT)
   (LABEL-CNT CINFO))
 (LABEL-ALIST CINFO))
 (ADD1 (ADD1 (LABEL-CNT CINFO))))
     COND-LIST
     (BEGIN-BODY STMT)
     PROC-LIST))))))
 (NO-DUPLICATES (COLLECT-LABELS (CODE (TRANSLATE CINFO COND-LIST STMT PROC-LIST)))))
 ((INSTRUCTIONS PROMOTE
                (DEMOTE 5)
                (DIVE 1 1)
                S
                (REWRITE LABEL-CNT-BIG-ENOUGH-ADD1)
                UP S TOP PROMOTE
                (DEMOTE 4)
                (DIVE 1 1 1)
                (REWRITE LABELS-UNIQUE-BEGIN-CASE-HYPS)
                NX
                (REWRITE LABELS-UNIQUE-BEGIN-CASE-HYPS)
                UP UP S TOP PROMOTE
                (DIVE 1 1 1)
                (REWRITE BEGIN-TRANSLATION)
                UP
                (REWRITE CODE-ADD-CODE-COMMUTE)
                UP
                (REWRITE COLLECT-LABELS-DISTRIBUTES)
                (DIVE 2)
                (= *
                   (LIST (ADD1 (LABEL-CNT CINFO)))
                   ((ENABLE COLLECT-LABELS)))
                UP UP
```

```
(REWRITE NO-DUPLICATES-RIGHT-CONS-REDUCTION)
(DIVE 1)
(REWRITE FIND-LABELP-MONOTONIC-LESSP)
UP S
(DIVE 2)
(REWRITE ADD-CODE-DOESNT-AFFECT-OTHER-FIELDS)
(REWRITE SET-LABEL-ALIST-DOESNT-AFFECT-OTHER-FIELDS)
UP
(REWRITE LABEL-CNT-LESSP1)
PROVE
(DIVE 1 2)
(REWRITE CODE-ADD-CODE-COMMUTE)
(DIVE 1)
(REWRITE SET-LABEL-ALIST-DOESNT-AFFECT-OTHER-FIELDS)
UP UP
(REWRITE FIND-LABELP-APPEND2)
(DIVE 3)
(= F)
TOP S
(DIVE 1)
(REWRITE FIND-LABELP-MONOTONIC-LESSP)
TOP S PROVE S
(DIVE 1)
(REWRITE LABEL-CNT-BIG-ENOUGH-NOT-FIND-LABELP)
TOP S
(REWRITE LABEL-CNT-BIG-ENOUGH-ADD1)
(DEMOTE 5)
S
(REWRITE LABEL-CNT-BIG-ENOUGH-ADD1))))
```

EVENT: Disable labels-unique-begin-case.

EVENT: Disable find-labelp-rewrites-to-member.

THEOREM: translate-leaves-labels-unique

$$\begin{array}{ll}
\multicolumn{2}{l}{(\text{no-duplicates}\,(\text{collect-labels}\,(\text{code}\,(\mathit{cinfo})))} \\
\wedge & \text{label-cnt-big-enough}\,(\text{label-cnt}\,(\mathit{cinfo}),\ \text{code}\,(\mathit{cinfo}))) \\
\rightarrow & \text{no-duplicates}\,(\text{collect-labels}\,(\text{code}\,(\text{translate}\,(\mathit{cinfo},\ \mathit{cond\text{-}list},\ \mathit{stmt},\ \mathit{proc\text{-}list}))))
\end{array}$$

```
;; Note: many of the following lemmas may never be used, particularly the ones
;; involving assoc in the hyps.
```

THEOREM: translate-proc-list-assoc1
(definedp (*subr*, *proc-list*) ∧ ok-mg-def-plistp1 (*proc-list*, *proc-list2*))
→ (translate-def (assoc (*subr*, *proc-list*), *proc-list2*)
= assoc (*subr*, translate-proc-list1 (*proc-list*, *proc-list2*)))

EVENT: Disable translate-proc-list-assoc1.

THEOREM: translate-proc-list-assoc
(user-defined-procp (*subr*, *proc-list*) ∧ ok-mg-def-plistp (*proc-list*))
→ (translate-def (assoc (*subr*, *proc-list*), *proc-list*)
= assoc (*subr*, translate-proc-list (*proc-list*)))

EVENT: Disable translate-proc-list-assoc.

THEOREM: translate-proc-list-assoc2
(user-defined-procp (*subr*, *proc-list*) ∧ ok-mg-def-plistp (*proc-list*))
→ (assoc (*subr*, translate-proc-list (*proc-list*))
= translate-def (assoc (*subr*, *proc-list*), *proc-list*))

EVENT: Disable translate-proc-list-assoc2.

THEOREM: translate-definedp1
(ok-mg-def-plistp1 (*lst1*, *lst2*) ∧ definedp (*x*, *lst1*))
→ definedp (*x*, translate-proc-list1 (*lst1*, *lst2*))

EVENT: Disable translate-definedp1.

THEOREM: assoc-mg-simple-variable-assignment-translate-proc-list
assoc (`'mg-simple-variable-assignment`, translate-proc-list (*proc-list*))
= MG-SIMPLE-VARIABLE-ASSIGNMENT-TRANSLATION

THEOREM: assoc-mg-simple-constant-assignment-translate-proc-list
assoc (`'mg-simple-constant-assignment`, translate-proc-list (*proc-list*))
= MG-SIMPLE-CONSTANT-ASSIGNMENT-TRANSLATION

THEOREM: assoc-mg-simple-variable-eq-translate-proc-list
assoc (`'mg-simple-variable-eq`, translate-proc-list (*proc-list*))
= MG-SIMPLE-VARIABLE-EQ-TRANSLATION

THEOREM: assoc-mg-simple-constant-eq-translate-proc-list
assoc (`'mg-simple-constant-eq`, translate-proc-list (*proc-list*))
= MG-SIMPLE-CONSTANT-EQ-TRANSLATION

THEOREM: assoc-mg-integer-le-translate-proc-list
assoc (`'mg-integer-le`, translate-proc-list (*proc-list*))
= MG-INTEGER-LE-TRANSLATION

THEOREM: assoc-mg-integer-unary-minus-translate-proc-list
assoc (`'mg-integer-unary-minus`, translate-proc-list (*proc-list*))
= MG-INTEGER-UNARY-MINUS-TRANSLATION

THEOREM: assoc-mg-integer-add-translate-proc-list
assoc (`'mg-integer-add`, translate-proc-list (*proc-list*))
= MG-INTEGER-ADD-TRANSLATION

THEOREM: assoc-mg-integer-subtract-translate-proc-list
assoc (`'mg-integer-subtract`, translate-proc-list (*proc-list*))
= MG-INTEGER-SUBTRACT-TRANSLATION

THEOREM: assoc-mg-boolean-or-translate-proc-list
assoc (`'mg-boolean-or`, translate-proc-list (*proc-list*))
= MG-BOOLEAN-OR-TRANSLATION

THEOREM: assoc-mg-boolean-and-translate-proc-list
assoc (`'mg-boolean-and`, translate-proc-list (*proc-list*))
= MG-BOOLEAN-AND-TRANSLATION

THEOREM: assoc-mg-boolean-not-translate-proc-list
assoc (`'mg-boolean-not`, translate-proc-list (*proc-list*))
= MG-BOOLEAN-NOT-TRANSLATION

THEOREM: assoc-mg-index-array-translate-proc-list
assoc (`'mg-index-array`, translate-proc-list (*proc-list*))
= MG-INDEX-ARRAY-TRANSLATION

THEOREM: assoc-mg-array-element-assignment-translate-proc-list
assoc (`'mg-array-element-assignment`, translate-proc-list (*proc-list*))
= MG-ARRAY-ELEMENT-ASSIGNMENT-TRANSLATION

THEOREM: assoc-user-defined-proc2
(¬ predefined-procp (*subr*))
→ (assoc (*subr*, translate-proc-list (*proc-list*))
    = assoc (*subr*, translate-proc-list1 (*proc-list*, *proc-list*)))

THEOREM: translate-def-body-rewrite
(ok-mg-def-plistp (*proc-list*)
 ∧ user-defined-procp (*subr*, *proc-list*)
 ∧ (code (translate-def-body (assoc (*subr*, *proc-list*), *proc-list*))
    = append (code (translate (*cinfo*, *t-cond-list*, *stmt*, *proc-list*)),

code2)))
$\rightarrow$ (cdddr (assoc (*subr*, translate-proc-list (*proc-list*)))
= append (code (translate (*cinfo*, *t-cond-list*, *stmt*, *proc-list*)),
code2))

Event: Disable translate-def-body-rewrite.

Theorem: car-definedp-defined-procp1
(user-defined-procp (*subr*, *proc-list*)
$\land$ ok-mg-def-plistp1 (*proc-list*, *proc-list2*))
$\rightarrow$ definedp (*subr*, translate-proc-list1 (*proc-list*, *proc-list2*))

Event: Disable car-definedp-defined-procp1.

Theorem: car-definedp-defined-procp
(user-defined-procp (*subr*, *proc-list*) $\land$ ok-mg-def-plistp (*proc-list*))
$\rightarrow$ definedp (*subr*, translate-proc-list (*proc-list*))

Event: Disable car-definedp-defined-procp.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                          ;;
;;                                  CLOCK                                   ;;
;;                                                                          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The time required for a call to a predefined procedure is the sum of
;; the time for the call sequence and that spent in the body.  The call
;; sequence is fixed but the body may have various paths.
```

Definition:
clock-predefined-proc-call-sequence (*name*)
= **case on** *name*:
case = *mg-simple-variable-assignment*
then 3
case = *mg-simple-constant-assignment*
then 3
case = *mg-simple-variable-eq*
then 4
case = *mg-simple-constant-eq*
then 4
case = *mg-integer-le*

```
     then 4
   case = mg-integer-unary-minus
     then 6
   case = mg-integer-add
     then 7
   case = mg-integer-subtract
     then 7
   case = mg-boolean-or
     then 4
   case = mg-boolean-and
     then 4
   case = mg-boolean-not
     then 3
   case = mg-index-array
     then 8
   case = mg-array-element-assignment
     then 8
   otherwise 0 endcase
```

EVENT: Disable clock-predefined-proc-call-sequence.

DEFINITION:
clock-predefined-proc-call-body-translation (*stmt*, *mg-state*)

```
=  case on call-name (stmt):
   case = mg-simple-variable-assignment
   then 5
   case = mg-simple-constant-assignment
     then 4
   case = mg-simple-variable-eq
     then 8
   case = mg-simple-constant-eq
     then 7
   case = mg-integer-le
     then 9
   case = mg-integer-unary-minus
     then if small-integerp (inegate (untag (caddr (assoc (cadr (call-actuals (stmt)),
                                                             mg-alist (mg-state))))),
                             MG-WORD-SIZE)  then 11
          else 10 endif
   case = mg-integer-add
     then if small-integerp (iplus (untag (caddr (assoc (cadr (call-actuals (stmt)),
                                                           mg-alist (mg-state)))),
                                    untag (caddr (assoc (caddr (call-actuals (stmt)),
                                                           mg-alist (mg-state))))),
                          MG-WORD-SIZE)  then 13
          else 11 endif
    case = mg-integer-subtract
     then if small-integerp (idifference (untag (caddr (assoc (cadr (call-actuals (stmt)),
                                                               mg-alist (mg-state)))),
                                          untag (caddr (assoc (caddr (call-actuals (stmt)),
                                                               mg-alist (mg-state))))),
                          MG-WORD-SIZE)  then 13
          else 11 endif
    case = mg-boolean-or
     then 8
    case = mg-boolean-and
     then 8
    case = mg-boolean-not
     then 6
    case = mg-index-array
     then if negativep (cadaddr (assoc (caddr (call-actuals (stmt)),
                                        mg-alist (mg-state))))  then 7
          elseif (idifference (cadddr (call-actuals (stmt)),
                               cadaddr (assoc (caddr (call-actuals (stmt)),
                                               mg-alist (mg-state))))
                  =   0)
                 ∨   negativep (idifference (cadddr (call-actuals (stmt)),
                                             cadaddr (assoc (caddr (call-actuals (stmt)),
                                                             mg-alist (mg-state)))))
          then 11
          else 17 endif
    case = mg-array-element-assignment
     then if negativep (cadaddr (assoc (cadr (call-actuals (stmt)),
                                        mg-alist (mg-state))))  then 7
          elseif (idifference (cadddr (call-actuals (stmt)),
                               cadaddr (assoc (cadr (call-actuals (stmt)),
                                               mg-alist (mg-state))))
                  =   0)
                 ∨   negativep (idifference (cadddr (call-actuals (stmt)),
                                             cadaddr (assoc (cadr (call-actuals (stmt)),
                                                             mg-alist (mg-state)))))
          then 11
          else 17 endif
    otherwise 0 endcase
```

EVENT: Disable clock-predefined-proc-call-body-translation.

DEFINITION:

```
predefined-proc-call-clock (stmt, mg-state)
=   (clock-predefined-proc-call-sequence (call-name (stmt))
      +   clock-predefined-proc-call-body-translation (stmt, mg-state))
```

EVENT: Disable predefined-proc-call-clock.

```
;; Removed the definition of clock-r
```

DEFINITION:

```
clock (stmt, proc-list, mg-state, n)
=   if $(n \simeq 0) \lor (\neg\ \text{normal}(mg\text{-}state))$ then 0
    else case on car (stmt):
           case = no-op-mg
           then 0
           case = signal-mg
            then 3
           case = prog2-mg
            then clock (prog2-left-branch (stmt),
                        proc-list,
                        mg-state,
                        n − 1)
                  +   clock (prog2-right-branch (stmt),
                             proc-list,
                             mg-meaning (prog2-left-branch (stmt),
                                         proc-list,
                                         mg-state,
                                         n − 1),
                             n − 1)
           case = loop-mg
            then if ¬ normal (mg-meaning (loop-body (stmt),
                                          proc-list,
                                          mg-state,
                                          n − 1))
                  then if cc (mg-meaning (loop-body (stmt),
                                          proc-list,
                                          mg-state,
                                          n − 1))
                          =   'leave
                       then 3 + clock (loop-body (stmt),
                                       proc-list,
                                       mg-state,
                                       n − 1)
                       else 1 + clock (loop-body (stmt),
                                  proc-list,
                                  mg-state,
                                  $n - 1$) endif
                     else $1 + ((1 +$ clock (loop-body (stmt),
                                  proc-list,
                                  mg-state,
                                  $n - 1$))
                          $+$  clock (stmt,
                                 proc-list,
                                 mg-meaning (loop-body (stmt),
                                        proc-list,
                                        mg-state,
                                        $n - 1$),
                                 $n - 1$)) endif
                 case = if-mg
                  then if mg-expression-falsep (if-condition (stmt), mg-state)
                     then if normal (mg-meaning (if-false-branch (stmt),
                                      proc-list,
                                      mg-state,
                                      $n - 1$))
                        then 5 + clock (if-false-branch (stmt),
                                  proc-list,
                                  mg-state,
                                  $n - 1$)
                        else 4 + clock (if-false-branch (stmt),
                                  proc-list,
                                  mg-state,
                                  $n - 1$) endif
                     elseif normal (mg-meaning (if-true-branch (stmt),
                                     proc-list,
                                     mg-state,
                                     $n - 1$))
                     then 5 + clock (if-true-branch (stmt),
                               proc-list,
                               mg-state,
                               $n - 1$)
                     else 3 + clock (if-true-branch (stmt),
                               proc-list,
                               mg-state,
                               $n - 1$) endif
                 case = begin-mg
                  then if cc (mg-meaning (begin-body (stmt),
                               proc-list,
                               mg-state,
                   n − 1))
  ∈  when-labels (stmt)
then if normal (mg-meaning (when-handler (stmt),
                              proc-list,
                              set-condition (mg-meaning (begin-body (stmt),
                                                          proc-list,
                                                          mg-state,
                                                          n − 1),
                                            'normal),
                              n − 1))
      then clock (begin-body (stmt),
                  proc-list,
                  mg-state,
                  n − 1)
            +  (3 + clock (when-handler (stmt),
                            proc-list,
                            set-condition (mg-meaning (begin-body (stmt),
                                                        proc-list,
                                                        mg-state,
                                                        n − 1),
                                          'normal),
                            n − 1))
      else clock (begin-body (stmt),
                 proc-list,
                 mg-state,
                 n − 1)
           +  (2 + clock (when-handler (stmt),
                           proc-list,
                           set-condition (mg-meaning (begin-body (stmt),
                                                       proc-list,
                                                       mg-state,
                                                       n − 1),
                                         'normal),
                           n − 1)) endif
elseif normal (mg-meaning (begin-body (stmt),
                            proc-list,
                            mg-state,
                            n − 1))
then 2 + clock (begin-body (stmt),
                 proc-list,
                 mg-state,
                 n − 1)
else clock (begin-body (stmt),
            proc-list,
                        mg-state,
                        n − 1) endif
           case = proc-call-mg
            then data-length (def-locals (fetch-called-def (stmt,
                                                            proc-list)))
                 +  length (def-locals (fetch-called-def (stmt,
                                                          proc-list)))
                 +  length (call-actuals (stmt))
                 +  1
                 +  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)
                 +  5
                 +  if normal (mg-meaning (def-body (fetch-called-def (stmt,
                                                                       proc-list)),
                                           proc-list,
                                           make-call-environment (mg-state,
                                                                  stmt,
                                                                  fetch-called-def (stmt,
                                                                                    proc-list)),
                                           n − 1)) then 1
                    else 3 endif
           case = predefined-proc-call-mg
            then predefined-proc-call-clock (stmt, mg-state)
           otherwise 0 endcase endif
```

Theorem: clock-prog2

```
 (car (stmt) = 'prog2-mg)
 →   (clock (stmt, proc-list, mg-state, n)
       =  if (n ≄ 0) ∧ normal (mg-state)
          then clock (prog2-left-branch (stmt), proc-list, mg-state, n − 1)
               +   clock (prog2-right-branch (stmt),
                          proc-list,
                          mg-meaning (prog2-left-branch (stmt),
                                      proc-list,
                                      mg-state,
                                      n − 1),
                          n − 1)
          else 0 endif)
```

THEOREM: clock-loop

```
 (car (stmt) = 'loop-mg)
 →   (clock (stmt, proc-list, mg-state, n)
        =   if (n ≄ 0) ∧ normal (mg-state)
            then if ¬ normal (mg-meaning (loop-body (stmt),
                                          proc-list,
                                          mg-state,
                                          n − 1))
                 then if cc (mg-meaning (loop-body (stmt),
                                         proc-list,
                                         mg-state,
                                         n − 1))
                         =  'leave
                      then 3 + clock (loop-body (stmt),
                                      proc-list,
                                      mg-state,
                                      n − 1)
                      else 1 + clock (loop-body (stmt),
                                      proc-list,
                                      mg-state,
                                      n − 1) endif
                 else 1 + ((1 + clock (loop-body (stmt),
                                       proc-list,
                                       mg-state,
                                       n − 1))
                           +   clock (stmt,
                                      proc-list,
                                      mg-meaning (loop-body (stmt),
                                                  proc-list,
                                                  mg-state,
                                                  n − 1),
                                      n − 1)) endif
            else 0 endif)
```

EVENT: Disable clock-loop.

THEOREM: clock-if

```
 (car (stmt) = 'if-mg)
 →   (clock (stmt, proc-list, mg-state, n)
        =   if (n ≄ 0) ∧ normal (mg-state)
            then if mg-expression-falsep (if-condition (stmt), mg-state)
                 then if normal (mg-meaning (if-false-branch (stmt),
                                             proc-list,
                                             mg-state,
                                        n − 1))
                    then 5 + clock (if-false-branch (stmt),
                                    proc-list,
                                    mg-state,
                                    n − 1)
                    else 4 + clock (if-false-branch (stmt),
                                   proc-list,
                                   mg-state,
                                   n − 1) endif
              elseif normal (mg-meaning (if-true-branch (stmt),
                                         proc-list,
                                         mg-state,
                                         n − 1))
              then 5 + clock (if-true-branch (stmt),
                              proc-list,
                              mg-state,
                              n − 1)
              else 3 + clock (if-true-branch (stmt),
                             proc-list,
                             mg-state,
                             n − 1) endif
         else 0 endif)
```

Theorem: clock-begin

```
 (car (stmt) = 'begin-mg)
→    (clock (stmt, proc-list, mg-state, n)
      =  if (n ≄ 0) ∧ normal (mg-state)
         then if cc (mg-meaning (begin-body (stmt),
                                 proc-list,
                                 mg-state,
                                 n − 1))
                 ∈   when-labels (stmt)
              then if normal (mg-meaning (when-handler (stmt),
                                          proc-list,
                                          set-condition (mg-meaning (begin-body (stmt),
                                                                     proc-list,
                                                                     mg-state,
                                                                     n − 1),
                                                         'normal),
                                          n − 1))
                   then clock (begin-body (stmt),
                               proc-list,
                               mg-state,
                               n − 1)
```

```
                         +    (3 + clock (when-handler (stmt),
                                          proc-list,
                                          set-condition (mg-meaning (begin-body (stmt),
                                                                     proc-list,
                                                                     mg-state,
                                                                     n − 1),
                                                         'normal),
                                          n − 1))
                      else clock (begin-body (stmt),
                                  proc-list,
                                  mg-state,
                                  n − 1)
                        +    (2 + clock (when-handler (stmt),
                                         proc-list,
                                         set-condition (mg-meaning (begin-body (stmt),
                                                                    proc-list,
                                                                    mg-state,
                                                                    n − 1),
                                                        'normal),
                                         n − 1)) endif
                 elseif normal (mg-meaning (begin-body (stmt),
                                            proc-list,
                                            mg-state,
                                            n − 1))
                 then 2 + clock (begin-body (stmt),
                                 proc-list,
                                 mg-state,
                                 n − 1)
                 else clock (begin-body (stmt),
                             proc-list,
                             mg-state,
                             n − 1) endif
            else 0 endif)
```

Theorem: clock-proc-call

```
 (car (stmt) = 'proc-call-mg)
 →    (clock (stmt, proc-list, mg-state, n)
        =    if (n ≄ 0) ∧ normal (mg-state)
             then data-length (def-locals (fetch-called-def (stmt, proc-list)))
                  +    length (def-locals (fetch-called-def (stmt, proc-list)))
                  +    length (call-actuals (stmt))
                  +    1
                  +    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)
          +   5
          +   if normal (mg-meaning (def-body (fetch-called-def (stmt,
                                                                  proc-list)),
                                     proc-list,
                                     make-call-environment (mg-state,
                                                            stmt,
                                                            fetch-called-def (stmt,
                                                                              proc-list)),
                                     n − 1))  then 1
              else 3 endif
       else 0 endif)
```

THEOREM: clock-predefined-proc-call

```
 (car (stmt) = 'predefined-proc-call-mg)
 →   (clock (stmt, proc-list, mg-state, n)
       =   if (n ≄ 0) ∧ normal (mg-state)
           then predefined-proc-call-clock (stmt, mg-state)
           else 0 endif)
```

DEFINITION:

```
map-down (mg-state, proc-list, ctrl-stk, temp-stk, addr, cond-list)
=   p-state (addr,
             ctrl-stk,
             map-down-values (mg-alist (mg-state),
                              bindings (top (ctrl-stk)),
                              temp-stk),
             translate-proc-list (proc-list),
             list (list ('c-c, mg-cond-to-p-nat (cc (mg-state), cond-list))),
             MG-MAX-CTRL-STK-SIZE,
             MG-MAX-TEMP-STK-SIZE,
             MG-WORD-SIZE,
             'run)
```

```
;; I need the hyp that cc is not 'leave for this theorem because cond-subsetp does not
;; preserves ok-cc unless cond is not 'leave, but I can prove that meaning never
;; returns leave anyway.
```

THEOREM: map-up-vars-inverts-map-down

```
 (all-cars-unique (mg-vars)
```

∧ mg-alistp (*mg-vars*)
∧ no-p-aliasing (*bindings*, *mg-vars*)
∧ mg-vars-list-ok-in-p-state (*mg-vars*, *bindings*, *temp-stk*))
→ (map-up-vars-list (*bindings*,
map-down-values (*mg-vars*, *bindings*, *temp-stk*),
signature (*mg-vars*))
= *mg-vars*)

THEOREM: cond-subset-preserves-ok-cc
((*cc* ≠ `'leave`)
∧ cond-subsetp (*r-cond-list*, *t-cond-list*)
∧ ok-cc (*cc*, *r-cond-list*))
→ ok-cc (*cc*, *t-cond-list*)

THEOREM: map-up-inverts-map-down
(all-cars-unique (mg-alist (*mg-state*))
∧ ok-mg-statep (*mg-state*, *r-cond-list*)
∧ cond-subsetp (*r-cond-list*, *t-cond-list*)
∧ 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*))
∧ (cc (*mg-state*) ≠ `'leave`)
∧ (¬ resource-errorp (*mg-state*)))
→ (map-up (map-down (*mg-state*,
proc-list,
ctrl-stk,
temp-stk,
addr,
t-cond-list),
signature (mg-alist (*mg-state*)),
t-cond-list)
= *mg-state*)

```
;; These are used in the proofs which follow!
```

THEOREM: call-exact-time-hyps1
((car (*stmt*) = `'proc-call-mg`)
∧ ok-mg-statement (*stmt*, *r-cond-list*, *name-alist*, *proc-list*)
∧ 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)

THEOREM: resources-adequate-temp-stk-not-max

$$\begin{array}{ll}
(\neg\ \text{resources-inadequatep}\,(\mathit{stmt}, & \\
\qquad\qquad\qquad\qquad\qquad \mathit{proc\text{-}list}, & \\
\qquad\qquad\qquad\qquad\qquad \text{list}\,(\text{length}\,(\mathit{temp\text{-}stk}),\ \text{p-ctrl-stk-size}\,(\mathit{ctrl\text{-}stk})))) & \\
\rightarrow \quad ((\text{length}\,(\mathit{temp\text{-}stk}) < \textsc{mg-max-temp-stk-size}) = \mathbf{t}) &
\end{array}$$

THEOREM: plus-difference-cancellation

$$((x - y) \not\simeq \mathtt{0}) \rightarrow (((x - y) + y) = \text{fix}\,(x))$$

THEOREM: lessp-difference-lemma1

$$((n < (r + l)) \wedge (r < (m - l))) \rightarrow ((n < m) = \mathbf{t})$$

THEOREM: resources-adequate-temp-stk-not-max2

$$\begin{array}{ll}
((\neg\ \text{resources-inadequatep}\,(\mathit{stmt}, & \\
\qquad\qquad\qquad\qquad\qquad \mathit{proc\text{-}list}, & \\
\qquad\qquad\qquad\qquad\qquad \text{list}\,(\text{length}\,(\mathit{temp\text{-}stk}),\ \text{p-ctrl-stk-size}\,(\mathit{ctrl\text{-}stk})))) & \\
\ \wedge \quad (\text{car}\,(\mathit{stmt}) = \texttt{'predefined-proc-call-mg}) & \\
\ \wedge \quad (n < (\text{predefined-proc-call-temp-stk-requirement}\,(\text{call-name}\,(\mathit{stmt})) & \\
\qquad\qquad + \quad \text{length}\,(\mathit{temp\text{-}stk})))) & \\
\rightarrow \quad ((n < \textsc{mg-max-temp-stk-size}) = \mathbf{t}) &
\end{array}$$

THEOREM: lessp-difference-lemma3

$$((n \leq p) \wedge (p < (m - c))) \rightarrow ((m < (n + c)) = \mathbf{f})$$

EVENT: Disable lessp-difference-lemma3.

THEOREM: resources-adequate-ctrl-stk-not-max

$$\begin{array}{ll}
((\neg\ \text{resources-inadequatep}\,(\mathit{stmt}, & \\
\qquad\qquad\qquad\qquad\qquad \mathit{proc\text{-}list}, & \\
\qquad\qquad\qquad\qquad\qquad \text{list}\,(\text{length}\,(\mathit{temp\text{-}stk}),\ \text{p-ctrl-stk-size}\,(\mathit{ctrl\text{-}stk})))) & \\
\ \wedge \quad (\text{car}\,(\mathit{stmt}) = \texttt{'predefined-proc-call-mg}) & \\
\ \wedge \quad (n \leq \text{predefined-proc-call-p-frame-size}\,(\text{call-name}\,(\mathit{stmt})))) & \\
\rightarrow \quad ((\textsc{mg-max-ctrl-stk-size} < (n + \text{p-ctrl-stk-size}\,(\mathit{ctrl\text{-}stk}))) = \mathbf{f}) &
\end{array}$$

THEOREM: lessp-transitive3

$$((y < n) \wedge (n < (m - x))) \rightarrow (((x + y) < m) = \mathbf{t})$$

THEOREM: lessp-difference

$$(y < (m - x)) \rightarrow (((x + y) < m) = \mathbf{t})$$

THEOREM: resources-proc-call-temp-stk-ok

$$\begin{array}{ll}
((\text{car}\,(\mathit{stmt}) = \texttt{'proc-call-mg}) & \\
\ \wedge \quad (\neg\ \text{resources-inadequatep}\,(\mathit{stmt}, & \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{proc\text{-}list}, & \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{list}\,(\text{length}\,(\mathit{temp\text{-}stk}), &
\end{array}$$

```
                                        p-ctrl-stk-size (ctrl-stk)))))
 →   (((length (temp-stk)
         +   data-length (def-locals (fetch-called-def (stmt, proc-list)))
         +   length (def-locals (fetch-called-def (stmt, proc-list)))
         +   length (call-actuals (stmt)))
        <   MG-MAX-TEMP-STK-SIZE)
       =   t)
```

Event: Disable resources-proc-call-temp-stk-ok.

Theorem: user-defined-def-locals-nil

```
 (ok-mg-def-plistp (proc-list)
  ∧   (car (stmt) = 'proc-call-mg)
  ∧   ok-mg-statement (stmt, r-cond-list, name-alist, proc-list))
 →   (length (caddr (assoc (call-name (stmt), translate-proc-list (proc-list))))
       =   0)
```

Event: Disable user-defined-def-locals-nil.

Theorem: user-defined-def-formals-rewrite

```
 (ok-mg-def-plistp (proc-list)
  ∧   (car (stmt) = 'proc-call-mg)
  ∧   ok-mg-statement (stmt, r-cond-list, name-alist, proc-list))
 →   (length (cadr (assoc (call-name (stmt), translate-proc-list (proc-list))))
       =   (length (def-locals (assoc (call-name (stmt), proc-list)))
             +   length (def-formals (assoc (call-name (stmt), proc-list)))))
```

Event: Disable user-defined-def-formals-rewrite.

Theorem: difference-preserves-lessp2

$(n < m) \rightarrow (((n - k) < m) = \mathbf{t})$

Theorem: plus-lessp

$((n + m + x) < (m + n)) = \mathbf{f}$

Theorem: resources-proc-call-ctrl-stk-ok

```
 ((car (stmt) = 'proc-call-mg)
  ∧   (¬ resources-inadequatep (stmt,
                                proc-list,
                                list (length (temp-stk),
                                      p-ctrl-stk-size (ctrl-stk))))
  ∧   ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)
  ∧   ok-mg-def-plistp (proc-list)
```

$$
\begin{array}{ll}
\land & \text{user-defined-procp}\,(\mathit{subr},\ \mathit{proc\text{-}list}))\\
\rightarrow & ((\textsc{mg-max-ctrl-stk-size}\\
 & \quad < \quad (\mathtt{2}\\
 & \qquad\quad + \quad \text{length}\,(\text{cadr}\,(\text{assoc}\,(\text{call-name}\,(\mathit{stmt}),\\
 & \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{translate-proc-list}\,(\mathit{proc\text{-}list}))))\\
 & \qquad\quad + \quad \text{length}\,(\text{caddr}\,(\text{assoc}\,(\text{call-name}\,(\mathit{stmt}),\\
 & \qquad\qquad\qquad\qquad\qquad\qquad\quad\;\; \text{translate-proc-list}\,(\mathit{proc\text{-}list}))))\\
 & \qquad\quad + \quad \text{p-ctrl-stk-size}\,(\mathit{ctrl\text{-}stk})))\\
 & \quad = \quad \mathbf{f})
\end{array}
$$

Event: Disable resources-proc-call-ctrl-stk-ok.

Event: Make the library `"c5"`.

Index