Event: Start with the library "c4".

 Definition:

\[
\text{mg-to-p-local-values} (\text{locals}) =
\begin{align*}
\text{if } & \text{locals} \simeq \text{nil} \text{ then nil} \\
\text{elseif } & \text{simple-mg-type-refp (cadr (car (locals)))} \\
\text{then } & \text{cons (mg-to-p-simple-literal (caddr (car (locals))), mg-to-p-local-values (cdr (locals)))} \\
\text{else } & \text{append (mg-to-p-simple-literal-list (caddr (car (locals))), mg-to-p-local-values (cdr (locals))) endif}
\end{align*}
\]

Theorem: mg-to-p-local-values-plistp

\[\text{plistp (mg-to-p-local-values (lst))}\]

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

Definition:

\[
\text{map-call-formals} (\text{formals}, \text{actuals}, \text{bindings}) =
\begin{align*}
\text{if } & \text{formals} \simeq \text{nil} \text{ then nil} \\
\text{else } & \text{cons (cons (car (car (formals))), cdr (assoc (car (actuals), bindings))), map-call-formals (cdr (formals), cdr (actuals), bindings)) endif}
\end{align*}
\]

Theorem: length-map-call-formals

\[\text{length (map-call-formals (formals, actuals, bindings)) = length (formals)}\]

Theorem: map-call-formals-plistp

\[\text{plistp (map-call-formals (x, y, z))}\]

Theorem: listcars-map-call-formals

\[\text{listcars (map-call-formals (formals, actuals, bindings)) = listcars (formals)}\]

\[\text{;}; \text{Each of the local values is placed onto the temp-stk, then the indexes into the stack are placed there as well. The distance of the value from the index}\]

1
Definition:
map-call-locals \( (\text{locals}, n) \)
\[
= \begin{cases} 
\text{nil} & \text{if } \text{locals} \simeq \text{nil} \\
\text{simple-mg-type-refp} (\text{cadr (car (locals)})) & \text{elseif}
\text{cons} (\text{cons (car (locals)), tag ('nat, n)}), \\
\text{map-call-locals} (\text{cdr (locals)}, 1 + n) & \text{else} \\
\text{cons} (\text{car (cadr (car (locals))), tag ('nat, n)}), \\
\text{map-call-locals} (\text{cdr (locals)}, \\
\text{array-length} (\text{cadr (car (locals)})) \\
+ n) & \text{endif}
\end{cases}
\]

Theorem: length-map-call-locals
\[ \text{length} (\text{map-call-locals} (\text{locals}, \text{n})) = \text{length} (\text{locals}) \]

Theorem: map-call-locals-plistp
\[ \text{plistp} (\text{map-call-locals} (\text{locals}, \text{n})) \]

Theorem: map-call-locals-preserves-listcars
\[ \text{listcars} (\text{map-call-locals} (\text{locals}, \text{m})) = \text{listcars} (\text{locals}) \]

Definition:
make-frame-alist \( (\text{def, stmt, ctrl-stk, temp-stk}) \)
\[
= \text{append} (\text{map-call-locals} (\text{def-locals} (\text{def}), \text{length} (\text{temp-stk})), \\
\text{map-call-formals} (\text{def-formals} (\text{def}), \\
\text{call-actuals} (\text{stmt}), \\
\text{bindings} (\text{top (ctrl-stk)})))
\]

Definition:
mg-actuals-to-p-actuals \( (\text{mg-actuals, bindings}) \)
\[
= \begin{cases} 
\text{nil} & \text{if } \text{mg-actuals} \simeq \text{nil} \\
\text{cons} (\text{cdr (assoc (car (mg-actuals), bindings)}), \\
\text{mg-actuals-to-p-actuals} (\text{cdr (mg-actuals), bindings}) & \text{else} \\
\text{endif}
\end{cases}
\]

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

;;;; depends on the size of the elements between.
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**: \( \text{ok-cinfop}(cinfo) = \text{plistp}(\text{code}(\text{cinfo})) \)

;; Given a list \( (x_1 \ldots x_n) \) and a label \( l \), this generated the list
;; \( ((x_1 . l) (x_2 . l) \ldots (x_n . l)) \). Notice that this allows that use
;; of the VALUE function for accessing the label.

**Definition**:
\[
\text{make-label-alist}(\text{name-list}, \text{label}) =
\begin{cases}
\text{nil} & \text{if } \text{name-list} \simeq \text{nil} \\
\text{cons}(\text{cons}(\text{car}(\text{name-list}), \text{label}), \\
\qquad \text{make-label-alist}(\text{cdr}(\text{name-list}), \text{label})) & \text{else}
\end{cases}
\]

**Definition**:
\[
\text{push-local-array-values-code}(\text{array-value}) =
\begin{cases}
\text{nil} & \text{if } \text{array-value} \simeq \text{nil} \\
\text{cons}(\text{list}(\text{'push-constant}, \\
\qquad \text{mg-to-p-simple-literal}(\text{car}(\text{array-value}))), \\
\qquad \text{push-local-array-values-code}(\text{cdr}(\text{array-value}))) & \text{else}
\end{cases}
\]

**Theorem**: length-push-local-array-values-code
\[
\text{length}(\text{push-local-array-values-code}(\text{array-value})) = \text{length}(\text{array-value})
\]

**Theorem**: length-push-local-array-values-code2
\[
(\text{ok-mg-local-data-decl}(\text{local}) \land \neg \text{simple-mg-type-refp}(\text{cadr}(\text{local}))) \rightarrow (\text{array-length}(\text{cadr}(\text{local})) = \text{length}(\text{caddr}(\text{local})))
\]

**Event**: Disable length-push-local-array-values-code2.

**Definition**:
\[
\text{push-locals-values-code}(\text{locals}) =
\begin{cases}
\text{nil} & \text{if } \text{locals} \simeq \text{nil} \\
\text{elseif } \text{simple-mg-type-refp}(\text{cadr}(\text{car}(\text{locals}))) \\
\qquad \text{then} \text{cons}(\text{list}(\text{'push-constant}, \\
\qquad \text{mg-to-p-simple-literal}(\text{caddr}(\text{car}(\text{locals})))), \\
\qquad \text{push-locals-values-code}(\text{cdr}(\text{locals}))) & \text{then}
\end{cases}
\]

**Theorem**: length-push-locals-values-code
\[
\text{ok-mg-local-data-plistp}(\text{locals}) \rightarrow (\text{length}(\text{push-locals-values-code}(\text{locals})) = \text{data-length}(\text{locals}))
\]

**Theorem**: length-mg-to-p-local-values
\[
\text{ok-mg-local-data-plistp}(\text{locals}) \rightarrow (\text{length}(\text{mg-to-p-local-values}(\text{locals})) = \text{data-length}(\text{locals}))
\]
**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 \( locals \cong \text{nil} \) then \( \text{nil} \)
  elseif simple-mg-type-refp \( (\text{cadr} \ (\text{car} \ (locals))) \)
  then cons \( ('\text{push-temp-stk-index}, \ n) \),
         push-locals-addresses-code \( (\text{cdr} \ (locals), \ n) \)
  else cons \( ('\text{push-temp-stk-index}, \ n),
         \text{push-locals-addresses-code} \ (\text{cdr} \ (locals),
          1 + (n - \text{array-length} \ (\text{cadr} \ (\text{car} \ (locals)))))) \) endif

**Theorem:** length-push-locals-addresses-code
length \( \text{push-locals-addresses-code} \ (locals, n) \) = 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 \( actuals \cong \text{nil} \) then \( \text{nil} \)
  else cons \( ('\text{push-local}, \ \text{car} \ (actuals)),
          \text{push-actuals-code} \ (\text{cdr} \ (actuals)) \) endif

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

**Theorem:** length-push-actuals-code
length \( \text{push-actuals-code} \ (actuals) \) = length \( actuals \)

**Definition:**
push-parameters-code \( (locals, actuals) \)
= append \( \text{push-locals-values-code} \ (locals),
          \text{append} \ (\text{push-locals-addresses-code} \ (locals,
          \text{data-length} \ (locals) - 1),
          \text{push-actuals-code} \ (actuals)) \)

**Theorem:** length-push-parameters-code
ok-mg-local-data-plistp \( locals \)
→ (length \( \text{push-parameters-code} \ (locals, actuals) \)
  = (\text{data-length} \ (locals) + \text{length} \ (locals) + \text{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 \approx 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)) = \text{fix} (n)

**Definition:**
index-cond-case-induction-hint (i, j, k) =
  if k \approx 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) \land (j \in \mathbb{N})) \rightarrow (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 \approx 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 \ast \text{length}(call-conds))

6
\textbf{Definition:}
\begin{align*}
\text{label-cnt-list}\ (lc, n) = & \quad \text{if } n \approx 0 \text{ then nil} \\
& \quad \text{else } \text{cons}\ (lc, \text{label-cnt-list}\ (lc, n - 1)) \text{ endif}
\end{align*}

\textbf{Theorem:} length-label-cnt-list \\
\begin{align*}
\text{length}\ (\text{label-cnt-list}\ (lc, n)) = & \quad \text{fix}\ (n) \\
\end{align*}

\text{;; I must make sure that the condition index is in-range. I can do this by using the def-conds as the list to index rather than the make-cond-list.}

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

\textbf{Definition:}
\begin{align*}
\text{condition-map-code}\ (\text{actual-conds,} \ lc, \ \text{cond-list,} \ \text{label-alist,} \ \text{proc-locals-lngth}) = & \quad \text{append}\ (\text{list}\ ('\text{push-global}, \ 'c-c), \ \\
& \quad \text{append}\ (\text{cons}\ ('\text{jump-case}, \ \\
& \quad \quad \text{cons}\ (lc, \ \\
& \quad \quad \quad \text{cond-case-jump-label-list}\ (1 + lc, \ \\
& \quad \quad \quad \quad 1 + \text{length}\ (\text{actual-conds}))))), \ \\
& \quad \text{label-cnt-list}\ (lc, \ \text{proc-locals-lngth}), \ \\
& \quad \text{list}\ ('\text{dl}, \ lc, \ \text{nil}, \ ('\text{push-constant} \ (\text{nat} \ 1))), \ \\
& \quad \quad ('\text{pop-global} \ c-c), \ \\
& \quad \quad \text{list}\ ('\text{jump}, \ \text{fetch-label} ('\text{routineerror}, \ \text{label-alist})), \ \\
& \quad \quad \text{append}\ (\text{cond-conversion}\ (\text{actual-conds}, \ \\
& \quad \quad \quad 1 + (1 + lc), \ \\
& \quad \quad \quad \text{cond-list}, \ \\
& \quad \quad \quad \text{label-alist}), \ \\
& \quad \quad \quad \text{list}\ (\text{list}\ ('\text{dl}, \ 1 + lc, \ \text{nil}, \ ('\text{no-op})))))
\end{align*}

\textbf{Definition:}
\begin{align*}
\text{proc-call-code}\ (\text{cinfo,} \ \text{stmt,} \ \text{cond-list,} \ \text{locals,} \ \text{cond-locals-lngth}) = & \quad \text{append}\ (\text{push-parameters-code}\ (\text{locals,} \ \text{call-actuals}\ (\text{stmt})), \ \\
& \quad \text{cons}\ (\text{list}\ ('\text{call}, \ \text{call-name}\ (\text{stmt})), \ \\
& \quad \quad \text{condition-map-code}\ (\text{call-conds}\ (\text{stmt}), \ \\
& \quad \quad \quad \text{label-cnt}\ (\text{cinfo}), \ \\
& \quad \quad \quad \text{cond-list}, \ \\
& \quad \quad \quad \text{label-alist}\ (\text{cinfo}), \ \\
& \quad \quad \quad \text{cond-locals-lngth})))
\end{align*}

7
The following functions define the sequence of statements laid down for a call to a predefined procedure.

**Definition:**

\[ mg\text{-}simple\text{-}variable\text{-}assignment\text{-}call\text{-}sequence \ (stmt) \]
\[ \begin{align*} &= \text{list} \ (\text{list} \ \left( \text{'push\text{-}local} , \text{car} \ (\text{call\text{-}actuals} \ (stmt)) \right), \\
& \quad \text{list} \ \left( \text{'push\text{-}local} , \text{cadr} \ (\text{call\text{-}actuals} \ (stmt)) \right), \ \\
& \quad \left( \text{call} \ \text{mg\text{-}simple\text{-}variable\text{-}assignment} \right) \end{align*} \]

**Definition:**

\[ mg\text{-}simple\text{-}constant\text{-}assignment\text{-}call\text{-}sequence \ (stmt) \]
\[ \begin{align*} &= \text{list} \ (\text{list} \ \left( \text{'push\text{-}local} , \text{car} \ (\text{call\text{-}actuals} \ (stmt)) \right), \\
& \quad \text{list} \ \left( \text{'push\text{-}constant} , \\
& \quad \quad \text{mg\text{-}to\text{-}p\text{-}simple\text{-}literal} \ (\text{cadr} \ (\text{call\text{-}actuals} \ (stmt)))) \right), \ \\
& \quad \left( \text{call} \ \text{mg\text{-}simple\text{-}constant\text{-}assignment} \right) \end{align*} \]

**Definition:**

\[ mg\text{-}simple\text{-}variable\text{-}eq\text{-}call\text{-}sequence \ (stmt) \]
\[ \begin{align*} &= \text{list} \ (\text{list} \ \left( \text{'push\text{-}local} , \text{car} \ (\text{call\text{-}actuals} \ (stmt)) \right), \\
& \quad \text{list} \ \left( \text{'push\text{-}local} , \text{cadr} \ (\text{call\text{-}actuals} \ (stmt)) \right), \\
& \quad \text{list} \ \left( \text{'push\text{-}local} , \text{caddr} \ (\text{call\text{-}actuals} \ (stmt)) \right), \ \\
& \quad \left( \text{call} \ \text{mg\text{-}simple\text{-}variable\text{-}eq} \right) \end{align*} \]

**Definition:**

\[ mg\text{-}simple\text{-}constant\text{-}eq\text{-}call\text{-}sequence \ (stmt) \]
\[ \begin{align*} &= \text{list} \ (\text{list} \ \left( \text{'push\text{-}local} , \text{car} \ (\text{call\text{-}actuals} \ (stmt)) \right), \\
& \quad \text{list} \ \left( \text{'push\text{-}local} , \text{cadr} \ (\text{call\text{-}actuals} \ (stmt)) \right), \\
& \quad \text{list} \ \left( \text{'push\text{-}constant} , \\
& \quad \quad \text{mg\text{-}to\text{-}p\text{-}simple\text{-}literal} \ (\text{caddr} \ (\text{call\text{-}actuals} \ (stmt)))) \right), \ \\
& \quad \left( \text{call} \ \text{mg\text{-}simple\text{-}constant\text{-}eq} \right) \end{align*} \]

**Definition:**

\[ mg\text{-}integer\text{-}le\text{-}call\text{-}sequence \ (stmt) \]
\[ \begin{align*} &= \text{list} \ (\text{list} \ \left( \text{'push\text{-}local} , \text{car} \ (\text{call\text{-}actuals} \ (stmt)) \right), \\
& \quad \text{list} \ \left( \text{'push\text{-}local} , \text{cadr} \ (\text{call\text{-}actuals} \ (stmt)) \right), \\
& \quad \text{list} \ \left( \text{'push\text{-}local} , \text{caddr} \ (\text{call\text{-}actuals} \ (stmt)) \right), \ \\
& \quad \left( \text{call} \ \text{mg\text{-}integer\text{-}le} \right) \end{align*} \]
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))
list (‘push-local, cadr (call-actuals (stmt))),
list (‘push-local, caddr (call-actuals (stmt)));
\(\text{(call mg-boolean-and)}\)

**Definition:**

\text{mg-boolean-not-call-sequence} (\text{stmt})
= 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)))))
\(\text{(call mg-boolean-not)}\);

;; 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:**

\text{mg-index-array-call-sequence} (\text{stmt}, \text{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)))))
\(\text{(call mg-index-array)},\)
\(\text{(push-global c-c)},\)
\(\text{(sub1-nat)},\)
list (‘test-nat-and-jump,
‘zero,
fetch-label (‘routineerror, label-alist)))

**Definition:**

\text{mg-array-element-assignment-call-sequence} (\text{stmt}, \text{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)))))
\(\text{(call mg-array-element-assignment)},\)
\(\text{(push-global c-c)},\)
\(\text{(sub1-nat)},\)
list (‘test-nat-and-jump,
‘zero,
fetch-label (‘routineerror, label-alist)))

**Definition:**

\text{predefined-proc-call-sequence} (\text{stmt}, \text{label-alist})
= case on call-name (\text{stmt}):
  case = mg-simple-variable-assignment
then \( mg\text{-simple-variable-assignment-call-sequence (stmt)} \)
case = \( mg\text{-simple-constant-assignment} \)
  then \( mg\text{-simple-constant-assignment-call-sequence (stmt)} \)
case = \( mg\text{-simple-variable-eq} \)
  then \( mg\text{-simple-variable-eq-call-sequence (stmt)} \)
case = \( mg\text{-simple-constant-eq} \)
  then \( mg\text{-simple-constant-eq-call-sequence (stmt)} \)
case = \( mg\text{-integer-le} \)
  then \( mg\text{-integer-le-call-sequence (stmt)} \)
case = \( mg\text{-integer-Unary-Minus} \)
  then \( mg\text{-integer-unary-minus-call-sequence (stmt, label-alist)} \)
case = \( mg\text{-integer-add} \)
  then \( mg\text{-integer-add-call-sequence (stmt, label-alist)} \)
case = \( mg\text{-integer-subtract} \)
  then \( mg\text{-integer-subtract-call-sequence (stmt, label-alist)} \)
case = \( mg\text{-boolean-or} \)
  then \( mg\text{-boolean-or-call-sequence (stmt)} \)
case = \( mg\text{-boolean-and} \)
  then \( mg\text{-boolean-and-call-sequence (stmt)} \)
case = \( mg\text{-boolean-not} \)
  then \( mg\text{-boolean-not-call-sequence (stmt)} \)
case = \( mg\text{-index-array} \)
  then \( mg\text{-index-array-call-sequence (stmt, label-alist)} \)
case = \( mg\text{-array-element-assignment} \)
  then \( mg\text{-array-element-assignment-call-sequence (stmt, label-alist)} \)
otherwise nil endcase

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

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

**Definition:**

\[ mg\text{-simple-variable-assignment-translation} = \ ' (mg\text{-simple-variable-assignment} (dest source) nil (push-local source) (fetch-temp-stk) (push-local dest) (deposit-temp-stk) (ret)) \]

**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 x)
   (fetch-temp-stk)
   (push-local y)
   (eq)
   (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**

\[
\text{mg-integer-unary-minus-translation} = \text{'}(mg-integer-unary-minus} \\
\begin{array}{l}
\text{(ans x)} \\
\text{((min-int (int \ -2147483648)) (temp-x (int \ 0)))} \\
\text{(push-local x)} \\
\text{(fetch-temp-stk)} \\
\text{(set-local temp-x)} \\
\text{(push-local min-int)} \\
\text{(eq)} \\
\text{(test-bool-and-jump f 0)} \\
\text{(push-constant (nat 1))} \\
\text{(pop-global c-c)} \\
\text{(jump 1)} \\
\text{(dl 0 nil (push-local temp-x))} \\
\text{(neg-int)} \\
\text{(push-local ans)} \\
\text{(deposit-temp-stk)} \\
\text{(dl 1 nil (ret)))}
\end{array}
\]

**Definition:**

**MG-INTEGER-ADD-TRANSLATION**

\[
\text{mg-integer-add-translation} = \text{'}(mg-integer-add} \\
\begin{array}{l}
\text{(ans y z)} \\
\text{((t1 (int \ 0)))} \\
\text{(push-constant (bool f))} \\
\text{(push-local y)} \\
\text{(fetch-temp-stk)} \\
\text{(push-local z)} \\
\text{(fetch-temp-stk)}
\end{array}
\]
(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)))
(push-global c-c)
(dl 1 nil (ret)))

Definition:

\texttt{mg-integer-subtract-translation} = ' (mg-integer-subtract
 \begin{verbatim}
 (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)))
\end{verbatim}

Definition:

\texttt{mg-boolean-or-translation} = ' (mg-boolean-or
 \begin{verbatim}
 (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))
\end{verbatim}
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:**

\[
\text{MG-ARRAY-ELEMENT-ASSIGNMENT-TRANSLATION} = \begin{array}{l}
\text{'(mg-array-element-assignment} \\
\text{ (a i value array-size) \\
\text{ ((temp-i (nat 0))) \\
\text{ (push-local i) \\
\text{ (fetch-temp-stk) \\
\text{ (set-local temp-i) \\
\text{ (test-int-and-jump neg 0) \\
\text{ (push-local array-size) \\
\text{ (push-local temp-i) \\
\text{ (sub-int) \\
\text{ (test-int-and-jump not-pos 0) \\
\text{ (push-local value) \\
\text{ (fetch-temp-stk) \\
\text{ (push-local a) \\
\text{ (push-local temp-i) \\
\text{ (int-to-nat) \\
\text{ (add-nat) \\
\text{ (deposit-temp-stk) \\
\text{ (jump 1) \\
\text{ (dl 0 nil (push-constant (nat 1))) \\
\text{ (pop-global c-c) \\
\text{ (dl 1 nil (ret)))} \\
\end{array}
\]

;; 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:**

\[
\text{PREDEFINED-PROCEDURE-TRANSLATIONS-LIST}
\]

16
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 label-alist
;;;
;;; PROG2
;;;
;;;   "code for left branch"
;;;   "code for right branch"

;;; LOOP
;;;
;;;   (dl 10 nil (no-op))
;;;   "code for loop-body"
;;;   (jump L0)
;;;   (dl 11 nil (push-constant (nat 2)))
IF

(push-local b)
(fetch-temp-stk)
(test-bool-and-jump false L0)
"code for true branch"
(jump L1)
(dl 10 nil (no-op))
"code for false branch"
(dl 11 nil (no-op))

BEGIN-WHEN

"code for begin-body"
(jump L1)
(dl 10 nil (push-constant (nat 2)))
(pop-global c-c)
"code for when-arm-body"
(dl 11 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 11 nil (push-constant "condition-number for cond1"))
(pop-global c-c)
(jump "label for cond1")
(dl 12 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)
Definition:

\[ \text{translate} (\text{cinfo}, \text{cond-list}, \text{stmt}, \text{proc-list}) = \begin{cases} \text{case on car (stmt)}: \\ \text{case} = \text{no-op-mg} & \text{then} \ \text{cinfo} \\ \text{case} = \text{signal-mg} & \text{then} \ \text{make-cinfo (append (code (cinfo),} \\
& \quad \text{list (list ('push-constant,} \\
& \quad \quad \text{mg-cond-to-p-nat (signalled-condition (stmt),} \\
& \quad \quad \quad \text{cond-list))}, \\
& \quad \text{list ('pop-global, 'c-c),} \\
& \quad \text{list ('jump,} \\
& \quad \quad \text{fetch-label (signalled-condition (stmt),} \\
& \quad \quad \quad \text{label-alist (cinfo)}))), \\
& \quad \text{label-alist (cinfo),} \\
& \quad \text{label-cnt (cinfo)} \\ \text{case} = \text{prog2-mg} & \text{then} \ \text{translate (translate (cinfo,} \\
& \quad \text{cond-list,} \\
& \quad \text{prog2-left-branch (stmt),} \\
& \quad \text{proc-list),} \\
& \quad \text{cond-list,} \\
& \quad \text{prog2-right-branch (stmt),} \\
& \quad \text{proc-list)} \\ \text{case} = \text{loop-mg} & \text{then} \ \text{discard-label (add-code (translate (make-cinfo (append (code (cinfo),} \\
& \quad \text{list (list ('dl,} \\
& \quad \quad \text{label-cnt (cinfo),} \\
& \quad \quad \quad \text{nil,} \\
& \quad \quad \quad \quad \text{'(no-op))))),} \\
& \quad \quad \text{cons (cons ('leave,} \\
& \quad \quad \quad \quad 1 + \text{label-cnt (cinfo)}),} \\
& \quad \quad \quad \text{label-alist (cinfo)},} \\
& \quad \quad \quad 1 + (1 + \text{label-cnt (cinfo)}),} \\
& \quad \quad \text{cond-list,} \\
& \quad \quad \text{loop-body (stmt),} \\
& \quad \quad \text{proc-list),} \\
& \quad \text{list (list ('jump, label-cnt (cinfo))},} \\
& \quad \text{19} \end{cases} \]
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))))
  else 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, begin-body (stmt), proc-list),
                                          list (list (jump, 1 + label-cnt (cinfo)),
                                                   list ('dl, label-cnt (cinfo), nil),
                                                   20)
'(push-constant
  (nat 2)),
'(pop-global c-c)),

cond-list,
when-handler (stmt),
proc-list),
list (list ('d1, 1 + label-cnt (cinfo), nil, '(no-op))))

\textbf{case} = \texttt{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)))))

\textbf{case} = \texttt{predefined-proc-call-mg}
then add-code (cinfo,
predefined-proc-call-sequence (stmt, label-alist (cinfo)))

\textbf{otherwise} cinfo endcase

\textbf{Theorem: signal-translation}
(car (stmt) = 'signal-mg)
→ (translate (cinfo, cond-list, stmt, proc-list)
  = make-cinfo (append (code (cinfo),
    list (list ('\texttt{push-constant},
      mg-cond-to-p-nat (signalled-condition (stmt),
        cond-list)),
    list ('\texttt{pop-global, c-c},
      list ('jump, fetch-label (signalled-condition (stmt),
        label-alist (cinfo)))))),
    label-alist (cinfo),
    label-cnt (cinfo))))

\textbf{Theorem: prog2-translation}
(car (stmt) = \texttt{prog2-mg})
→ (translate (cinfo, cond-list, stmt, proc-list)
  = translate (translate (cinfo,
    cond-list, prog2-left-branch (stmt),
    ...))
Theorem: loop-translation
\(\text{car}(\text{stmt}) = \text{'loop-mg}\)
\(\rightarrow (\text{translate}(\text{cinfo}, \text{cond-list}, \text{stmt}, \text{proc-list}) = \text{discard-label}(\text{add-code}(\text{translate}(\text{make-cinfo}(\text{append}(\text{code}(\text{cinfo}),
\quad \text{list}(\text{'dl},
\quad \text{label-cnt}(\text{cinfo}),
\quad \text{nil},
\quad \text{'(no-op)})),
\quad \text{cons}(\text{cons}(\text{'leave},
\quad 1 + \text{label-cnt}(\text{cinfo})),
\quad \text{label-alist}(\text{cinfo})),
\quad 1 + (1 + \text{label-cnt}(\text{cinfo}))),
\quad \text{cond-list},
\quad \text{loop-body}(\text{stmt}),
\quad \text{proc-list}),
\quad \text{list}(\text{list}(\text{'jump}, \text{label-cnt}(\text{cinfo})),
\quad \text{list}(\text{'dl},
\quad 1 + \text{label-cnt}(\text{cinfo}),
\quad \text{nil},
\quad \text{'(push-constant (nat 2))},
\quad \text{'(pop-global c-c))})))
\]
Theorem: begin-translation
\[(\text{car} (\text{stmt}) = \text{'begin-mg}) \Rightarrow (\text{translate} (\text{cinfo}, \text{cond-list}, \text{stmt}, \text{proc-list})) = \text{add-code} (\text{translate} (\text{add-code} (\text{set-label-alist} (\text{translate} (\text{make-cinfo} (\text{code} (\text{cinfo}), \text{append} (\text{make-label-alist} (\text{when-labels} (\text{stmt}), \text{label-cnt} (\text{cinfo}), \text{label-alist} (\text{cinfo}), 1 + (1 + \text{label-cnt} (\text{cinfo})))), \text{cond-list}, \text{begin-body} (\text{stmt}), \text{proc-list}), \text{label-alist} (\text{cinfo})), \text{list} (\text{list} (\text{'jump}, 1 + \text{label-cnt} (\text{cinfo})), \text{list} (\text{'dl}, \text{label-cnt} (\text{cinfo}), \text{nil}, \text{'(push-constant (\text{nat} 2))}, \text{'(pop-global c-c))), \text{cond-list}, \text{when-handler} (\text{stmt}), \text{proc-list}), \text{list} (\text{list} (\text{'dl}, 1 + \text{label-cnt} (\text{cinfo}), \text{nil}, \text{'(no-op))))))\]

Theorem: call-translation
\[(\text{car} (\text{stmt}) = \text{'proc-call-mg}) \Rightarrow (\text{translate} (\text{cinfo}, \text{cond-list}, \text{stmt}, \text{proc-list})) = \text{make-cinfo} (\text{append} (\text{code} (\text{cinfo}), \text{proc-call-code} (\text{cinfo}, \text{stmt}, \text{cond-list}, \text{def-locals} (\text{fetch-called-def} (\text{stmt}, \text{proc-list})))), \text{cond-list}, \text{when-handler} (\text{stmt}), \text{proc-list}), \text{list} (\text{list} (\text{'dl}, 1 + \text{label-cnt} (\text{cinfo}), \text{nil}, \text{'(no-op))))))\]
\[ \text{length (def-cond-locals (fetch-called-def (\text{stmt}, proc-list)))}, \]

\[ \text{+ (1 + (1 + length (call-conds (\text{stmt}))))} \]

**Theorem:** predefined-call-translation
\[
(\text{car (\text{stmt}) = predefined-proc-call-mg}) \rightarrow (\text{translate (\text{cinfo}, cond-list, stmt, proc-list)} = \text{add-code (\text{cinfo}, predefined-proc-call-sequence (\text{stmt}, label-alist (\text{cinfo}))})}
\]

**Event:** Disable translate.

**Theorem:** predefined-proc-call-code-plistp
\[
\text{plistp (predefined-proc-call-sequence (\text{stmt}, label-alist))}
\]

**Theorem:** not-find-labelp-predefined-proc-call-code
\[
\text{find-labelp (n, predefined-proc-call-sequence (\text{stmt}, label-alist)) = f}
\]

;; 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:**
\[
\text{translate-def-body (proc-def, proc-list)} = \text{add-code (translate (make-cinfo (\text{nil},\)}
\[
\text{cons (cons ('routineerror, 0),\)}
\[
\text{make-label-alist (make-cond-list (proc-def, 0)),\)}
\[
\text{1),\)}
\[
\text{make-cond-list (proc-def),\)}
\]

24
Let \( \text{def-body} (\text{proc-def}), \text{proc-list} ), \) 
\[
\text{list ('dl 0 nil (no-op)), } \\
\text{list ('pop*, data-length (def-locals (proc-def))), } \\
\text{'(ret))}
\]

**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:**

\[
\text{translate-def} (\text{def}, \text{proc-list}) = \text{append (list (def-name (def)),} \\
\quad \text{append (listcars (def-locals (def)),} \\
\quad \text{listcars (def-formals (def))),} \\
\quad \text{nil),} \\
\quad \text{code (translate-def-body (def, proc-list)))}
\]

**Definition:**

\[
\text{translate-proc-list1} (\text{proc-list1}, \text{proc-list2}) = \begin{cases} 
\text{nil} & \text{if } \text{proc-list1} \simeq \text{nil} \\
\text{cons (translate-def (car (proc-list1), \text{proc-list2}),} \\
\text{translate-proc-list1 (cdr (proc-list1), \text{proc-list2})}) & \text{endif}
\end{cases}
\]

**Definition:**

\[
\text{translate-proc-list} (\text{proc-list}) = \text{append (PREDEFINED-PROCEDURE-TRANSLATIONS-LIST,} \\
\text{translate-proc-list1 (proc-list, proc-list))}
\]

**Event:** Disable translate-proc-list.

**Theorem:** translate-preserves-fields

\[
\text{label-alist (translate (cinfo, cond-list, stmt, proc-list))} = \text{label-alist (cinfo)}
\]

**Theorem:** code-always-plistp

\[
\text{plistp (code (cinfo))} \rightarrow \text{plistp (code (translate (cinfo, cond-list, stmt, proc-list)))}
\]

**Theorem:** translate-preserves-ok-cinfop

\[
\text{ok-cinfop (cinfo) } \rightarrow \text{ok-cinfop (translate (cinfo, cond-list, stmt, proc-list))}
\]

**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)))

26
EVENT: Disable code-doesnt-affect-other-fields.

THEOREM: add-code-doesnt-affect-other-fields
\( (\text{label-alist} (\text{add-code} (\text{cinfo, code}))) = \text{label-alist} (\text{cinfo})) \)
\( \land \ (\text{label-cnt} (\text{add-code} (\text{cinfo, code}))) = \text{label-cnt} (\text{cinfo})) \)

THEOREM: set-label-alist-doesnt-affect-other-fields
\( (\text{code} (\text{set-label-alist} (\text{cinfo, label-alist}))) = \text{code} (\text{cinfo})) \)
\( \land \ (\text{label-cnt} (\text{set-label-alist} (\text{cinfo, label-alist}))) = \text{label-cnt} (\text{cinfo})) \)

THEOREM: discard-label-doesnt-affect-other-fields
\( (\text{code} (\text{discard-label} (\text{cinfo}))) = \text{code} (\text{cinfo})) \)
\( \land \ (\text{label-cnt} (\text{discard-label} (\text{cinfo}))) = \text{label-cnt} (\text{cinfo})) \)

THEOREM: nullify-cancels-add-code
\( \text{nullify} (\text{add-code} (\text{cinfo, code})) = \text{nullify} (\text{cinfo}) \)

THEOREM: code-add-code-commute
\( \text{code} (\text{add-code} (\text{cinfo, cd})) = \text{append} (\text{code} (\text{cinfo}), \text{cd}) \)

THEOREM: label-alist-set-label-alist
\( \text{label-alist} (\text{set-label-alist} (\text{state, label-alist}))) = \text{label-alist} \)

THEOREM: nullify-doesnt-affect-proc-call-code
\( \text{proc-call-code} (\text{nullify} (\text{cinfo}), \text{stmt, cond-list, locals, k}) = \text{proc-call-code} (\text{cinfo, stmt, cond-list, locals, k}) \)

THEOREM: nullify-code-nil
\( \text{code} (\text{nullify} (\text{cinfo})) = \text{nil} \)

DEFINITION:
\( \text{nullify-induction-hint} (\text{cinfo, cond-list, stmt, proc-list}) = \text{case on} \ \text{car} (\text{stmt}): \)
\( \text{case} = \text{no-op-mg} \)
\( \text{then} \ \text{t} \)
\( \text{case} = \text{signal-mg} \)
\( \text{then} \ \text{t} \)
\( \text{case} = \text{prog2-mg} \)
\( \text{then} \ \text{nullify-induction-hint} (\text{cinfo}, \text{cond-list}, \text{prog2-left-branch} (\text{stmt}), \text{proc-list}) \)
∀ 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)

\texttt{case = loop-mg}
\texttt{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)

\texttt{case = if-mg}
\texttt{then nullify-induction-hint (make-cinfo (list (list ('push-local, if-condition (stmt)),
    '(fetch-temp-stk),
    list ('test-bool-and-jump, 'false),
    nil, nil))}

28
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-jump,
                     'false,
                     label-cnt (cino))),
    label-alist (cinfo),
    1 + (1 + label-cnt (cinfo))),
   cond-list, 
   if-true-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-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 (cino))),
    label-alist (cinfo),
    1 + (1 + label-cnt (cinfo))),
   cond-list, 
   if-false-branch (stmt),
   proc-list)
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 ('d1,
label-cnt (cinfo),
nil,
'(no-op))),

cond-list,
if-false-branch (stmt),
proc-list)

\textbf{case} = \textit{begin-mg}
\textbf{then} nullify-induction-hint (add-code (set-label-alist (translate (make-cinfo (code (cinfo),
append (make-label-alist (when-labels (stmt),
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 ('d1,
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),
lab
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-label-alist (begin-body (stmt)
cond-list
label-alist 1 + (1 + label-cnt (cinfo)))
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),

31
\[
\begin{align*}
\text{cond-list}, \\
\text{prog2-left-branch} (\text{stmt}), \\
\text{proc-list})
\quad \rightarrow \quad \text{append} \ (\text{code} \ (\text{translate} \ (\text{nullify} \ (\text{cinfo}), \\
\quad \text{cond-list}, \\
\quad \text{prog2-left-branch} (\text{stmt}), \\
\quad \text{proc-list})), \\
\quad \text{code} \ (\text{translate} \ (\text{nullify} \ (\text{translate} \ (\text{nullify} \ (\text{cinfo}), \\
\quad \text{cond-list}, \\
\quad \text{prog2-left-branch} (\text{stmt}), \\
\quad \text{proc-list})), \\
\quad \text{cond-list}, \\
\quad \text{prog2-right-branch} (\text{stmt}), \\
\quad \text{proc-list})))) \\
= \quad \text{code} \ (\text{translate} \ (\text{translate} \ (\text{nullify} \ (\text{cinfo}), \\
\quad \text{cond-list}, \\
\quad \text{prog2-left-branch} (\text{stmt}), \\
\quad \text{proc-list})), \\
\quad \text{cond-list}, \\
\quad \text{prog2-right-branch} (\text{stmt}), \\
\quad \text{proc-list})))) \\
\wedge \quad \text{(ok-cinfop} \ (\text{translate} \ (\text{cinfo}, \\
\quad \text{cond-list}, \\
\quad \text{prog2-left-branch} (\text{stmt}), \\
\quad \text{proc-list})))) \\
\rightarrow \quad \text{append} \ (\text{code} \ (\text{translate} \ (\text{cinfo}, \\
\quad \text{cond-list}, \\
\quad \text{prog2-left-branch} (\text{stmt}), \\
\quad \text{proc-list})), \\
\quad \text{code} \ (\text{translate} \ (\text{nullify} \ (\text{translate} \ (\text{cinfo}, \\
\quad \text{cond-list}, \\
\quad \text{prog2-left-branch} (\text{stmt}), \\
\quad \text{proc-list})), \\
\quad \text{cond-list}, \\
\quad \text{prog2-right-branch} (\text{stmt}), \\
\quad \text{proc-list})))) \\
= \quad \text{code} \ (\text{translate} \ (\text{translate} \ (\text{cinfo}, \\
\quad \text{cond-list}, \\
\quad \text{prog2-left-branch} (\text{stmt}), \\
\quad \text{proc-list})), \\
\quad \text{cond-list}, \\
\quad \text{prog2-right-branch} (\text{stmt}), \\
\quad \text{proc-list})))) \\
\wedge \quad \text{(ok-cinfop} \ (\text{cinfo})
\end{align*}
\]
\[\begin{align*}
\text{Theorem: new-code-loop-case-induction-hyps} \\
& \text{ok-cinfop (cinfo)} \\
\rightarrow & \ (\text{ok-cinfop (make-cinfo (list (cons (}'dl, \\
& \text{cons (label-cnt (cinfo),)} \\
& \text{cons (nil (no-op))))),)} \\
& \text{cons (cons ('leave, 1 + label-cnt (cinfo)),)} \\
& \text{label-alist (cinfo)),)} \\
& 1 + (1 + label-cnt (cinfo))) \\
\wedge & \ (\text{ok-cinfop (make-cinfo (append (code (cinfo),)} \\
& \text{list (cons (}'dl, \\
& \text{cons (label-cnt (cinfo),)} \\
& \text{cons (nil (no-op))))),)} \\
& \text{cons (cons ('leave, 1 + label-cnt (cinfo)),)} \\
& \text{label-alist (cinfo)),)} \\
& 1 + (1 + label-cnt (cinfo)))\]

\[\begin{align*}
\text{Theorem: new-code-loop-case} \\
& ((\text{car (stmt)} = \text{'loop-mg}) \\n& \wedge \text{ok-cinfop (cinfo)} \\
& \wedge \ (\text{ok-cinfop (make-cinfo (list (cons (}'dl, \\
& \text{cons (label-cnt (cinfo),)} \\
& \text{cons (nil (no-op))))),)} \\
& \text{cons (cons ('leave, 1 + label-cnt (cinfo)),)} \\
& \text{label-alist (cinfo)),)} \\
& 1 + (1 + label-cnt (cinfo))) \\
\rightarrow & \ (\text{append (code (make-cinfo (list (cons (}'dl, \\
& \text{cons (label-cnt (cinfo),)} \\
& \text{cons (nil (no-op))))),)} \\
& \text{cons (cons ('leave, 1 + label-cnt (cinfo)),)} \\
& 1 + label-cnt (cinfo)),)}
\end{align*}\]
\begin{verbatim}
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)))))))))
\end{verbatim}
Theorem: new-code-if-case-induction-hyps

\[ \text{ok-cinfop} (\text{cinfo}) \rightarrow \text{ok-cinfop} (\text{add-code} (\text{translate} (\text{make-cinfo} (\text{list} (\text{list} (\text{'push-local},
\text{if-condition} (\text{stmt}),
\text{'\textit{(fetch-temp-stk)},
\text{list} (\text{test-bool-and-jump},
\text{'false},
\text{label-cnt} (\text{cinfo})),
\text{label-alist} (\text{cinfo}),
1 + (1 + \text{label-cnt} (\text{cinfo})))),
\text{cond-list},
\text{if-true-branch} (\text{stmt}),
\text{proc-list}),
\text{list} (\text{list} (\text{'jump}, 1 + \text{label-cnt} (\text{cinfo})),
\text{cons} (\text{'dl},
\text{cons} (\text{label-cnt} (\text{cinfo}),
\text{'\textit{(nil (no-op)}))))))
\∧ \text{ok-cinfop} (\text{add-code} (\text{translate} (\text{make-cinfo} (\text{append} (\text{code} (\text{cinfo}),
\text{list} (\text{list} (\text{'push-local},
\text{if-condition} (\text{stmt})))))))
\text{proc-list))))
\]

\[ \rightarrow (\text{append} (\text{code} (\text{cinfo}),
\text{code} (\text{translate} (\text{nullify} (\text{cinfo}, \text{cond-list}, \text{stmt}, \text{proc-list}))))
\]

\[ = \text{code} (\text{translate} (\text{cinfo}, \text{cond-list}, \text{stmt}, \text{proc-list}))
\]
\( (\text{fetch-temp-stk}), \)  
\( \text{list (}\text{test-bool-and-jump,} \) \)
\( \text{'false,} \)  
\( \text{label-cnt (}\text{cinfo}))))), \)
\( \text{label-alist (}\text{cinfo}), \)
\( 1 + (1 + \text{label-cnt (}\text{cinfo})))\), 
\( \text{cond-list,} \)  
\( \text{if-true-branch (}\text{stmt}), \)  
\( \text{proc-list)}, \)  
\( \text{list (}\text{jump,} 1 + \text{label-cnt (}\text{cinfo})), \)  
\( \text{cons (}\text{'dl,} \)  
\( \text{cons (}\text{label-cnt (}\text{cinfo}, \) \)
\( \text{'(nil (no-op)))))))) \)  
\( \land \)  
\( \text{ok-cinfop (}\text{make-cinfo (append (code (}\text{cinfo),} \) \)
\( \text{list (}\text{'}\text{push-local,} \) \)
\( \text{if-condition (}\text{stmt}), \) \)
\( \text{'(fetch-temp-stk),} \) \)
\( \text{list (}\text{'}\text{test-bool-and-jump,} \) \)
\( \text{'false,} \) \)
\( \text{label-cnt (}\text{cinfo)\)}, \) \)
\( \text{label-alist (}\text{cinfo}), \) \)
\( 1 + (1 + \text{label-cnt (}\text{cinfo)\}))\)) \)  
\( \land \)  
\( \text{ok-cinfop (}\text{make-cinfo (list (}\text{'}\text{push-local,} \) \)
\( \text{if-condition (}\text{stmt}), \) \)
\( \text{'(fetch-temp-stk),} \) \)
\( \text{list (}\text{'}\text{test-bool-and-jump,} \) \)
\( \text{'false,} \) \)
\( \text{label-cnt (}\text{cinfo)\)}, \) \)
\( \text{label-alist (}\text{cinfo}), \) \)
\( 1 + (1 + \text{label-cnt (}\text{cinfo)\}))\)) \)

**Theorem**: new-code-if-case  
\( ((\text{car (}\text{stmt} = '\text{if-mg}) \) \)  
\( \land \)  
\( \text{ok-cinfop (}\text{cinfo}), \) \)
\( \land \)  
\( \text{ok-cinfop (}\text{add-code (translate (}\text{make-cinfo (list (}\text{'}\text{push-local,} \) \)
\( \text{if-condition (}\text{stmt}), \) \)
\( \text{'(fetch-temp-stk),} \) \)
\( \text{list (}\text{'}\text{test-bool-and-jump,} \) \)
\( \text{'false,} \) \)
\( \text{label-cnt (}\text{cinfo)\)}, \) \)
\( \text{label-alist (}\text{cinfo}), \) \)
\( 1 + (1 + \text{label-cnt (}\text{cinfo)\}))\)) \)
\( \text{cond-list,} \) \)
\( \text{if-true-branch (}\text{stmt}, \) \)
\( 36 \)
\[\text{proc-list},\]
\[
\text{list (list ('jump, 1 + label-cnt (cinfo)),}
\]
\[
\text{cons ('dl,}
\]
\[
\text{cons (label-cnt (cinfo),}
\]
\[
\text{'(nil (no-op))))))}
\]
\[\rightarrow\]
\[
\text{(append (code (add-code (translate (make-cinfo (list (list ('push-local,
\text{if-condition (stmt)),
\text{'(fetch-temp-stk),
\text{list ('test-bool-and-jump,}
\text{'false,}
\text{label-cnt (cinfo))},
\text{label-alist (cinfo)},
\text{1 + (1 + label-cnt (cinfo))},
\text{cond-list,}
\text{if-true-branch (stmt),}
\text{proc-list),}
\text{list (list ('jump, 1 + label-cnt (cinfo)),}
\text{cons ('dl,}
\text{cons (label-cnt (cinfo),}
\text{'(nil (no-op))))))},
\text{code (translate (nullify (add-code (translate (make-cinfo (list (list ('push-local,
\text{if-condition (stmt)),
\text{'(fetch-temp-stk),
\text{list ('test-bool-and-jump,}
\text{'false,}
\text{label-cnt (cinfo))},
\text{label-alist (cinfo)},
\text{1 + (1 + label-cnt (cinfo))},
\text{cond-list,}
\text{if-true-branch (stmt),}
\text{proc-list),}
\text{list (list ('jump, 1 + label-cnt (cinfo)),}
\text{cons ('dl,}
\text{cons (label-cnt (cinfo),}
\text{'(nil (no-op))))))},
\text{cond-list,}
\text{if-false-branch (stmt),}
\text{proc-list))})}
\]
\[=\]
\[
\text{code (translate (add-code (translate (make-cinfo (list (list ('push-local,
\text{if-condition (stmt)),
\text{'(fetch-temp-stk),
\text{list ('test-bool-and-jump,}
\text{'false,}
\text{label-cnt (cinfo))},
\text{label-alist (cinfo)},
\text{1 + (1 + label-cnt (cinfo))},
\text{cond-list,}
\text{if-true-branch (stmt),}
\text{proc-list),}
\text{list (list ('jump, 1 + label-cnt (cinfo)),}
\text{cons ('dl,}
\text{cons (label-cnt (cinfo),}
\text{'(nil (no-op))))))},
\text{cond-list,}
\text{if-false-branch (stmt),}
\text{proc-list))})}
\]
\]
\[
\text{cond-list,}
\text{if-true-branch (stmt),}
\text{proc-list),}
\text{list (list ('jump,}
\text{1 + label-cnt (cinfo)),}
\text{cons ('dl,}
\text{cons (label-cnt (cinfo),}
\text{'(nil (no-op)))))},
\text{cond-list,}
\text{if-false-branch (stmt),}
\text{proc-list)))})
\wedge \text{(ok-cinfop (add-code (translate (make-cinfo (append (code (cinfo),}
\text{list (list ('push-local,}
\text{if-condition (stmt)),}
\text{'(fetch-temp-stk),}
\text{list ('test-bool-and-jump,}
\text{'false,}
\text{label-cnt (cinfo))},)
\text{label-alist (cinfo),}
\text{1 + (1 + label-cnt (cinfo))}),
\text{cond-list,}
\text{if-true-branch (stmt),}
\text{proc-list),}
\text{list (list ('jump, 1 + label-cnt (cinfo)),}
\text{cons ('dl,}
\text{cons (label-cnt (cinfo),}
\text{'(nil (no-op)))))}})
\rightarrow \text{(append (code (add-code (translate (make-cinfo (append (code (cinfo),}
\text{list (list ('push-local,}
\text{if-condition (stmt)),}
\text{'(fetch-temp-stk),}
\text{list ('test-bool-and-jump,}
\text{'false,}
\text{label-cnt (cinfo))},)
\text{label-alist (cinfo),}
\text{1 + (1 + label-cnt (cinfo))},
\text{cond-list,}
\text{if-true-branch (stmt),}
\text{proc-list),}
\text{38}
\end{array}
\]
\[
\text{list} \left( \text{list} \left( '\text{jump}, 1 + \text{label-cnt} \left( cinfo \right) \right), \\
\text{cons} \left( '\text{dl}, \\
\text{cons} \left( \text{label-cnt} \left( cinfo \right), \\
'\left( \text{nil} \left( \text{no-op} \right) \right) \right) \right) \right), \\
\text{code} \left( \text{translate} \left( \text{nullify} \left( \text{add-code} \left( \text{translate} \left( \text{make-cinfo} \left( \text{append} \left( \text{code} \left( cinfo \right), \\
\text{list} \left( \text{list} \left( '\text{push-local}, \\
\text{if-condition} \left( \text{stmt} \right), \\
'\left( \text{fetch-temp-stk} \right) \right), \\
\text{list} \left( '\text{test-bool-and-jump}, \\
'\text{false}, \\
\text{label-cnt} \left( cinfo \right) \right) \right), \\
\text{label-alist} \left( cinfo \right), \\
1 + (1 + \text{label-cnt} \left( cinfo \right)) \right), \\
\text{cond-list}, \\
\text{if-true-branch} \left( \text{stmt} \right), \\
\text{proc-list} \right) \right) \right) \right), \\
\text{list} \left( \text{list} \left( '\text{jump}, \\
1 + \text{label-cnt} \left( cinfo \right) \right), \\
\text{cons} \left( '\text{dl}, \\
\text{cons} \left( \text{label-cnt} \left( cinfo \right), \\
'\left( \text{nil} \left( \text{no-op} \right) \right) \right) \right) \right) \right), \\
\text{cond-list}, \\
\text{if-false-branch} \left( \text{stmt} \right), \\
\text{proc-list} \right) \right) \right) \\
\right) \\
\right) \\
= \text{code} \left( \text{translate} \left( \text{add-code} \left( \text{translate} \left( \text{make-cinfo} \left( \text{append} \left( \text{code} \left( cinfo \right), \\
\text{list} \left( \text{list} \left( '\text{push-local}, \\
\text{if-condition} \left( \text{stmt} \right), \\
'\left( \text{fetch-temp-stk} \right) \right), \\
\text{list} \left( '\text{test-bool-and-jump}, \\
'\text{false}, \\
\text{label-cnt} \left( cinfo \right) \right) \right), \\
\text{label-alist} \left( cinfo \right), \\
1 + (1 + \text{label-cnt} \left( cinfo \right)) \right), \\
\text{cond-list}, \\
\text{if-true-branch} \left( \text{stmt} \right), \\
\text{proc-list} \right) \right) \right) \right), \\
\text{list} \left( \text{list} \left( '\text{jump}, \\
1 + \text{label-cnt} \left( cinfo \right) \right), \\
\text{cons} \left( '\text{dl}, \\
\text{cons} \left( \text{label-cnt} \left( cinfo \right), \\
'\left( \text{nil} \left( \text{no-op} \right) \right) \right) \right) \right) \right), \\
\text{cond-list}, \\
\text{if-false-branch} \left( \text{stmt} \right), \\
\text{proc-list} \right) \right) \right) \\
\right) \\
\right) \right) \right) \\
\right)
if-false-branch (stmt, 
proc-list)))
∧ (ok-cinfop (make-cinfo (append (code (cinfo), 
list (list (list (push-local,
if-condition (stmt)),
(fetch-temp-stk),
list (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 (list (push-local,
if-condition (stmt)),
(fetch-temp-stk),
list (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 (list (push-local,
if-condition (stmt)),
(fetch-temp-stk),
list (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 (list (push-local,
if-condition (stmt)),
(fetch-temp-stk),
list (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)))

40
∧ (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 ('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 (code (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),

Prove the following lemma:

(prove-lemma new-code-begin-case (rewrite)
  (IMPLIES
    (AND
      (equal (car STMT) 'BEGIN-MG)
      (OK-CINFOP CINFO)
    )
    append (make-label-alist (when-labels (label-cnt (cinfo)), label-alist (cinfo)), 1 + (1 + label-cnt (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)))))))
                          43)
(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))

44
(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-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)
                                  (CONS (LABEL-CNT CINFO)))))))
                              (45)
'(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))))))
  (CONS (POP-GLOBAL C-C))))

(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))))))
  (CONS (POP-GLOBAL C-C))))

COND-LIST
(WHEN-HANDLER STMT)
PROC-LIST))

46
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)
  (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) = 'd1
          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
\[ \text{collect-labels} (\text{append} (\text{code}1, \text{code}2)) = \text{append} (\text{collect-labels} (\text{code}1), \text{collect-labels} (\text{code}2)) \]

Definition:
\[ \text{all-labels-unique} (\text{codelist}) = \text{no-duplicates} (\text{collect-labels} (\text{codelist})) \]

Event: Disable all-labels-unique.

Theorem: all-labels-unique-append
\[ \text{all-labels-unique} (\text{append} (x, y)) \rightarrow (\text{all-labels-unique} (x) \land \text{all-labels-unique} (y)) \]

Theorem: all-labels-unique-reduction
\[ (\neg \text{all-labels-unique} (y)) \rightarrow (\neg \text{all-labels-unique} (\text{append} (x, y))) \]

Event: Disable all-labels-unique-reduction.

Theorem: all-labels-unique-reduction2
\[ (\neg \text{all-labels-unique} (y)) \rightarrow (\neg \text{all-labels-unique} (\text{cons} (x, y))) \]

Event: Disable all-labels-unique-reduction2.

Theorem: find-labelp-rewrites-to-member
\[ \text{find-labelp} (\text{lab}, \text{code}) = (\text{lab} \in \text{collect-labels} (\text{code})) \]

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

Theorem: all-labels-unique-reduction3
\[ (\text{find-labelp} (\text{lab}, \text{code}1) \land \text{find-labelp} (\text{lab}, \text{code}2)) \rightarrow (\neg \text{all-labels-unique} (\text{append} (\text{code}1, \text{code}2))) \]

Event: Disable all-labels-unique-reduction3.

Theorem: no-duplicates-append-list
\[ \text{no-duplicates} (\text{append} (\text{lst}, \text{cons} (x, \text{cons} (y, \text{lst2})))) \rightarrow \text{no-duplicates} (\text{append} (\text{lst}, \text{list} (y))) \]

Theorem: no-duplicates-append-list2
\[ \text{no-duplicates} (\text{append} (\text{lst}, \text{cons} (y, \text{lst2}))) \rightarrow \text{no-duplicates} (\text{append} (\text{lst}, \text{list} (y))) \]
THEOREM: labels-unique-append2
\[ \text{all-labels-unique (append (} lst_1, \text{cons (} x, \text{cons (} y, \text{lst}_2)))) \]
\[ \rightarrow \text{all-labels-unique (append (} lst_1, \text{list (} y))) \]

THEOREM: find-labelp-member-collect-labels
\[ \text{find-labelp (} x, \text{code}) \rightarrow (x \in \text{collect-labels (} code)) \]

DEFINITION:
label-hole-big-enough (cinfo, cond-list, stmt, proc-list, y)
\[= \text{all-labels-unique (append (} \text{code (translate (} cinfo, \text{cond-list, stmt, proc-list))}, y)) \]

THEOREM: labels-unique-not-find-labelp
\[ (\text{all-labels-unique (append (} \text{code}_1, \text{code}_2)) \land \text{find-labelp (} \text{label}, \text{code}_2)) \]
\[ \rightarrow (\neg \text{find-labelp (} \text{label}, \text{code}_1)) \]

THEOREM: labels-unique-not-find-labelp1
\[ \text{all-labels-unique (append (} \text{lst}, \text{list (} \text{list ('}dl, \text{label}, \text{nil, w})))) \]
\[ \rightarrow (\text{find-labelp (} \text{label}, \text{lst}) = f) \]

DEFINITION:
ok-cond-list (lst)
\[= \text{if} \ \text{lst} \simeq \text{nil} \ \text{then} \ \text{lst} = \text{nil} \]
\[\quad \text{else} \ \text{(ok-mg-namep (car (} lst))} \]
\[\quad \lor (\text{car (} lst) \in \text{'}(\text{leave routineerror})) \]
\[\quad \land \ \text{ok-cond-list (} \text{cdr (} lst)) \ \text{endif} \]

THEOREM: identifier-plistp-make-cond-list-ok
\[ \text{identifier-plistp (} \text{lst}) \rightarrow \text{ok-cond-list (} \text{lst}) \]

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

THEOREM: cond-subsetp-preserves-ok-mg-statemp
\[ \text{cond-subsetp (} r\text{-cond-list, t-cond-list}) \]
\[ \land (\text{cc (} \text{mg-state}) \neq \text{'}\text{leave}) \]
\[ \land \ \text{ok-mg-statemp (} \text{mg-state, r-cond-list)}) \]
\[ \rightarrow \ \text{ok-mg-statemp (} \text{mg-state, t-cond-list}) \]
**Definition:**

ok-translation-parameters \((cinfo, cond-list, stmt, proc-list, y)\)
\[
= \ (\text{ok-cinfop} (cinfo)
  \land \ \text{ok-cond-list} (cond-list)
  \land \ \text{label-hole-big-enough} (cinfo, cond-list, stmt, proc-list, y))
\]

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

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

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

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

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

**Theorem:** not-find-labelp-push-parameters-code
\[
\text{find-labelp} (n, \text{push-parameters-code} (locals, actuals)) = \ f
\]

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

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

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

**Definition:**

label-cnt-big-enough \((lc, code)\)
\[
= \ \text{if} \ \text{code} \approx \text{nil} \ \text{then} \ t \ 
\text{elseif} \ \text{caar (code)} = \ 'd1 
\text{then} \ (\text{cadar (code)} < lc) \land \text{label-cnt-big-enough} (lc, \text{cdr (code)}) 
\text{else} \ \text{label-cnt-big-enough} (lc, \text{cdr (code)}) \ \text{endif}
\]
DEFINITION:
cond-conversion-induction-hint \((ls, n)\) =
if \(ls \simeq \text{nil}\) then \(t\)
else cond-conversion-induction-hint (cdr \((ls)\), 1 + \(n\)) endif

THEOREM: label-count-big-enough-not-find-labelp
label-cnt-big-enough \((lc, code)\) \(\rightarrow\) (find-labelp \((lc, code)\) = \(f\))

THEOREM: greater-label-count-big-enough
(label-cnt-big-enough \((n, code)\) \& (\(n \leq m\)))
\(\rightarrow\) label-cnt-big-enough \((m, code)\)

THEOREM: label-cnt-big-enough-distributes
(label-cnt-big-enough \((lc, append (lst1, lst2))\)
\(=\) (label-cnt-big-enough \((lc, lst1)\) \& label-cnt-big-enough \((lc, lst2)\))

THEOREM: label-cnt-lessp1
\((n < \text{label-cnt} \((cinfo))\)
\(\rightarrow\) \((n < \text{label-cnt} \((\text{translate} \((cinfo, \text{cond-list}, \text{stmt}, \text{proc-list})))\) = \(t\))

THEOREM: label-cnt-big-enough-distributes2
(label-cnt-big-enough \((n, lst1)\) \& label-cnt-big-enough \((n, lst2)\))
\(\rightarrow\) 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 + \text{length} \((lst)))\)),
\quad \text{cond-conversion} \((lst, 1 + (1 + lc), \text{cond-list}, \text{label-alist})\))

THEOREM: label-cnt-big-enough-for-proc-call-code
label-cnt-big-enough \((\text{label-cnt} \((cinfo), code \((cinfo)\))\)
\(\rightarrow\) label-cnt-big-enough \((\text{label-cnt} \((cinfo)\)
\quad + \quad (1 + (1 + \text{length} \((\text{call-conds} \((stmt)\))\))),
\quad \text{proc-call-code} \((cinfo, stmt, \text{cond-list}, \text{locals}, k)\))
Theorem: label-cnt-big-enough-for-predefined-proc-call-code
label-cnt-big-enough \( (n, \text{predefined-proc-call-sequence}(stmt, \text{label-alist})) \)

Theorem: label-cnt-stays-big-enough
label-cnt-big-enough (label-cnt \( (\text{info}, \text{code}(\text{info})) \))
\[ \rightarrow \quad \text{label-cnt-big-enough} \left( \text{label-cnt} \left( \text{translate} \left( \text{info}, \text{cond-list}, \text{stmt}, \text{proc-list} \right) \right), \text{code}(\text{translate}(\text{info}, \text{cond-list}, \text{stmt}, \text{proc-list})) \right) \]

Theorem: label-cnt-big-enough-add1
label-cnt-big-enough \( (x, y) \rightarrow \text{label-cnt-big-enough} \left( 1 + x, y \right) \)

Theorem: lesser-label-doesnt-disturb-no-duplicates
\( \text{no-duplicates}(\text{lst}) \land (x \notin \text{lst}) \rightarrow \text{no-duplicates}(\text{append}(\text{lst}, \text{list}(x))) \)

Theorem: find-labelp-reduces-to-member
\( \text{find-labelp}(x, \text{lst}) = (x \in \text{collect-labels}(\text{lst})) \)

Theorem: member-labels-unique-not-find-labelp
\( \text{all-labels-unique}(\text{append}(\text{code}, \text{code2})) \land \text{find-labelp}(\text{label}, \text{code2}) \rightarrow \neg \text{find-labelp}(\text{label}, \text{code}) \)

Theorem: no-duplicates-right-cons-reduction
\( \text{no-duplicates} \left( \text{collect-labels}(\text{lst}) \right) \rightarrow \left( \text{no-duplicates} \left( \text{append}(\text{collect-labels}(\text{lst}), \text{list}(x)) \right) \right) \)
\[ = \quad \neg \text{find-labelp}(x, \text{lst}) \]

Theorem: label-cnt-big-enough-not-find-labelp
\( \text{label-cnt-big-enough}(\text{lc}, \text{code}) \rightarrow (\text{find-labelp}(\text{lc}, \text{code}) = \text{f}) \)

Theorem: not-member-cond-conversion
\( (n < \text{lc}) \rightarrow (\text{find-labelp}(\text{lc}, \text{code}) = \text{f}) \)

Theorem: no-duplicates-cond-conversion
\( \text{no-duplicates}(\text{collect-labels}(\text{cond-conversion}(\text{conds}, \text{lc}, \text{cond-list}, \text{label-alist}))) \)

Theorem: no-duplicates-cond-conversion-base-case
\( \text{no-duplicates}(\text{append}(\text{collect-labels}(\text{cond-conversion}(\text{conds}, \text{lc}, \text{cond-list}, \text{label-alist})), \text{list}(1 + \text{lc}))) \)
**Theorem:** no-duplicates-proc-call  
\[(\text{no-duplicates} (\text{collect-labels} \, (\text{code}))) \land \text{label-cnt-big-enough} (lc, \text{code})\]  
\[\rightarrow \text{no-duplicates} (\text{append} (\text{collect-labels} \, (\text{code})),\]
\[\text{cons} (lc,\]
\[\text{append} (\text{collect-labels} \text{cond-conversion} (\text{conds},\]
\[1 + (1 + lc),\]
\[\text{cond-list},\]
\[\text{label-alist}),\]
\[\text{list} (1 + lc)))))\]

**Theorem:** collect-labels-push-actuals-code-nil  
\[\text{collect-labels} (\text{push-actuals-code} \, (\text{actuals})) = \text{nil}\]

**Theorem:** collect-labels-push-local-array-values-code-nil  
\[\text{collect-labels} (\text{push-local-array-values-code} \, (\text{array-value})) = \text{nil}\]

**Theorem:** collect-labels-push-locals-values-code-nil  
\[\text{collect-labels} (\text{push-locals-values-code} \, (\text{actuals})) = \text{nil}\]

**Theorem:** collect-labels-push-locals-addresses-code-nil  
\[\text{collect-labels} (\text{push-locals-addresses-code} \, (\text{actuals}, m)) = \text{nil}\]

**Theorem:** collect-labels-predefined-proc-call-code-nil  
\[\text{collect-labels} (\text{predefined-proc-call-sequence} \, (\text{stmt}, \text{label-alist})) = \text{nil}\]

**Theorem:** collect-labels-strip-label  
\[\text{collect-labels} (\text{cons} (\text{cons} (\text{’dl}, \text{cons} (\text{label}, x)), y)) = \text{cons} (\text{label}, \text{collect-labels} (y))\]

**Theorem:** labels-unique-loop-case  
\[((\text{car} \, (\text{stmt})) = \text{’loop-mg})\]
\[\land \text{no-duplicates} (\text{collect-labels} \, (\text{code} \, (\text{cinfo})))\]
\[\land \text{label-cnt-big-enough} (\text{label-cnt} \, (\text{cinfo}), \text{code} \, (\text{cinfo}))\]
\[\land ((\text{no-duplicates} (\text{collect-labels} \, (\text{code} \, (\text{make-cinfo} \, (\text{append} \, (\text{code} \, (\text{cinfo}),\]
\[\text{list} \, (\text{cons} (\text{’dl},\]
\[\text{cons} (\text{label-cnt} \, (\text{cinfo}),\]
\[\text{’nil},\]
\[\text{(no-op})))),\]
\[\text{cons} (\text{cons} (\text{’leave},\]
\[1 + \text{label-cnt} \, (\text{cinfo}),\]
\[\text{label-alist} \, (\text{cinfo})),\]
\[1 + (1 + \text{label-cnt} \, (\text{cinfo}))))))\]
\[\land \text{label-cnt-big-enough} (\text{label-cnt} \, (\text{make-cinfo} \, (\text{append} \, (\text{code} \, (\text{cinfo}),\]
\[\text{list} \, (\text{cons} (\text{’dl},\]
\[\text{cons} (\text{label-cnt} \, (\text{cinfo}),\]
\[53\]
\[
\text{'(nil (no-op))}, \quad \text{cons (cons ('leave, 1 + label-cnt (cinfo)), label-alist (cinfo)), 1 + (1 + label-cnt (cinfo))}, \quad \text{code (make-cinfo (append (code (cinfo), list (cons ('dl, cons (label-cnt (cinfo), 'nil (no-op)))))}, \quad \text{cons (cons ('leave, 1 + label-cnt (cinfo)), label-alist (cinfo)), 1 + (1 + label-cnt (cinfo)))})
\]

\[\rightarrow \text{no-duplicates (collect-labels (code (translate (make-cinfo (append (code (cinfo), list (cons ('dl, cons (label-cnt (cinfo), 'nil (no-op)))))}, \quad \text{cons (cons ('leave, 1 + label-cnt (cinfo)), label-alist (cinfo)), 1 + (1 + label-cnt (cinfo)))}, \quad \text{cond-list, loop-body (stmt), proc-list)))})}
\]

\[\rightarrow \text{no-duplicates (collect-labels (code (translate (cinfo, cond-list, stmt, proc-list)))})}
\]

**Event:** Disable labels-unique-loop-case.

**Theorem:** labels-unique-if-case-hyps1
\[
((\text{car (stmt)} = 'if-mg) \wedge \text{no-duplicates (collect-labels (code (cinfo))}) \wedge \text{label-cnt-big-enough (label-cnt (cinfo), code (cinfo))}) \rightarrow \text{(no-duplicates (collect-labels (code (make-cinfo (append (code (cinfo), list (list ('push-local, if-condition (stmt)), 'fetch-temp-stk), list ('test-bool-and-jump, 'false)))))})}
\]
\(\text{label-cnt}(\text{cinfo}))\),
\(\text{label-alist}(\text{cinfo})\),
\(1 + (1 + \text{label-cnt}(\text{cinfo})))\))
\(\wedge\) \(\text{label-cnt-big-enough}(\text{label-cnt}(\text{make-cinfo}(\text{append}(\text{code}(\text{cinfo})),
\text{list}(\text{list}('\text{push-local},
\text{if-condition}(\text{stmt})),
'\text{fetch-temp-stk}),
\text{list}('\text{test-bool-and-jump},
'\text{false},
\text{label-cnt}(\text{cinfo}))))),
\text{label-alist}(\text{cinfo}),
1 + (1 + \text{label-cnt}(\text{cinfo}))\))
\(\wedge\) \(\text{no-duplicates}(\text{collect-labels}(\text{code}(\text{cinfo})))\)
\(\wedge\) \(\text{label-cnt-big-enough}(\text{label-cnt}(\text{make-cinfo}(\text{append}(\text{code}(\text{cinfo})),
\text{list}(\text{list}('\text{push-local},
\text{if-condition}(\text{stmt})),
'\text{fetch-temp-stk}),
\text{list}('\text{test-bool-and-jump},
'\text{false},
\text{label-cnt}(\text{cinfo}))))),
\text{label-alist}(\text{cinfo}),
1 + (1 + \text{label-cnt}(\text{cinfo}))\))
\(\wedge\) \(\text{no-duplicates}(\text{collect-labels}(\text{code}(\text{translate}(\text{make-cinfo}(\text{append}(\text{code}(\text{cinfo})),
\text{list}(\text{list}('\text{push-local},
\text{if-condition}(\text{stmt})),
'\text{fetch-temp-stk}),
\text{list}('\text{test-bool-and-jump},
'\text{false},
\text{label-cnt}(\text{cinfo}))))),
\text{label-alist}(\text{cinfo}),
1 + (1 + \text{label-cnt}(\text{cinfo}))\))
\(\wedge\) \(\text{no-duplicates}(\text{collect-labels}(\text{code}(\text{add-code}(\text{translate}(\text{make-cinfo}(\text{append}(\text{code}(\text{cinfo},
\text{cond-list},
\text{if-true-branch}(\text{stmt}),
\text{proc-list})))))))\))

\text{EVENT: Disable labels-unique-if-case-hyps1.}

\text{THEOREM: label-cnt-big-enough-not-member}
\text{label-cnt-big-enough}(\text{lc}, \text{code}) \rightarrow (\text{lc} \not\in \text{collect-labels}(\text{code}))

\text{THEOREM: labels-unique-if-case-hyps2}
((\text{car}(\text{stmt}) = '\text{if-mg})
\wedge\) \(\text{no-duplicates}(\text{collect-labels}(\text{code}(\text{cinfo})))\)
\wedge\) \(\text{label-cnt-big-enough}(\text{label-cnt}(\text{cinfo}), \text{code}(\text{cinfo}))\)
\wedge\) \(\text{no-duplicates}(\text{collect-labels}(\text{code}(\text{translate}(\text{make-cinfo}(\text{append}(\text{code}(\text{cinfo}),
\text{list}(\text{list}('\text{push-local},
\text{if-condition}(\text{stmt})),
'\text{fetch-temp-stk}),
\text{list}('\text{test-bool-and-jump},
'\text{false},
\text{label-cnt}(\text{cinfo}))))),
\text{label-alist}(\text{cinfo}),
1 + (1 + \text{label-cnt}(\text{cinfo}))\))
\wedge\) \(\text{no-duplicates}(\text{collect-labels}(\text{code}(\text{add-code}(\text{translate}(\text{make-cinfo}(\text{append}(\text{code}(\text{cinfo},
\text{cond-list},
\text{if-true-branch}(\text{stmt}),
\text{proc-list}))))))))\))

55
\begin{verbatim}
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)))))),
\end{verbatim}
Theorem: labels-unique-if-case
\[\begin{align*}
\text{((car (stmt) = 'if-mg) } & \land \text{ no-duplicates (collect-labels (code (cinfo)))} \\
\land \text{ label-cnt-big-enough (label-cnt (cinfo), code (cinfo))} & \land \text{(no-duplicates (collect-labels (code (add-code (translate (make-cinfo (append (code (cinfo)), list (list ('push-local, if-condition (stmt) ' (fetch-temp-stk), list ('test-bool-and-, 'false, label-cnt (cinfo))))) label-alist (cinfo), 1 + (1 + label-cnt (cinfo))), list (list ('jump, 1 + label-cnt (cinfo)), cons ('dl, cons (label-cnt (cinfo), (nil (no-op))))))) list (list ('jump, 1 + label-cnt (cinfo)), cons ('dl, cons (label-cnt (cinfo), (nil (no-op)))))))))} \\
\land \text{ label-cnt-big-enough (label-cnt (add-code (translate (make-cinfo (append (code (cinfo)), list (list ('push-local, if-condition (stmt) ' (fetch-temp-stk), list ('test-bool-and-, 'false, label-cnt (cinfo))))) label-alist (cinfo), 1 + (1 + label-cnt (cinfo))), list (list ('jump, 1 + label-cnt (cinfo)), cons ('dl, cons (label-cnt (cinfo), (nil (no-op))))))))}
\end{align*}\]
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)))))))
→ no-duplicates (collect-labels (code (translate (add-code (translate (make-cinfo (append (code (cinfo),
list (list ('push-
if-co
'(fetch-
list ('tes
'false
label-
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,
\[ \text{cons (label-cnt (cinfo),}
  \text{'(nil}
  \text{(no-op))))},
\]
\[ \text{cond-list},
  \text{if-false-branch (stmt),}
  \text{proc-list)))))}
\]
\[ \land \ (\text{no-duplicates (collect-labels (code (make-cinfo (append (code (cinfo),}
  \text{list (list ('push-local,}
    \text{if-condition (stmt)),}
    \text{'(fetch-temp-stk),}
    \text{list ('test-bool-and-jump,}
    \text{'false,}
    \text{label-cnt (cinfo))))),}
  \text{label-alist (cinfo),}
  \text{1 + (1 + label-cnt (cinfo))))))})
\]
\[ \land \ \text{label-cnt-big-enough (label-cnt (make-cinfo (append (code (cinfo),}
  \text{list (list ('push-local,}
    \text{if-condition (stmt)),}
    \text{'(fetch-temp-stk),}
    \text{list ('test-bool-and-jump,}
    \text{'false,}
    \text{label-cnt (cinfo))))),}
  \text{label-alist (cinfo),}
  \text{1 + (1 + label-cnt (cinfo))))),}
  \text{code (make-cinfo (append (code (cinfo),}
    \text{list (list ('push-local,}
      \text{if-condition (stmt)),}
      \text{'(fetch-temp-stk),}
      \text{list ('test-bool-and-jump,}
      \text{'false,}
      \text{label-cnt (cinfo))))),}
    \text{label-alist (cinfo),}
    \text{1 + (1 + label-cnt (cinfo))))))})
\]
\[ \rightarrow \ \text{no-duplicates (collect-labels (code (translate (make-cinfo (append (code (cinfo),}
  \text{list (list ('push-local,}
    \text{if-condition (stmt)),}
    \text{'(fetch-temp-stk),}
    \text{list ('test-bool-and-jump,}
    \text{'false,}
    \text{label-cnt (cinfo))))),}
  \text{label-alist (cinfo),}
  \text{1 + (1 + label-cnt (cinfo))))),}
\]
\[ \text{cond-list},
  \text{if-true-branch (stmt),}
\]
no-duplicates (collect-labels (code (translate (cinfo, cond-list, 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-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))))))),
label-cnt-big-enough (label-cnt (add-code (set-label-alist (translate (make-cinfo (code (cinfo),
append (make-label-alist (label-alist (cinfo)),
1 + (1 + label-cnt (cinfo)))))))

∧ label-cnt-big-enough (label-cnt (add-code (set-label-alist (translate (make-cinfo (code (cinfo),
append (make-label-alist (label-alist (cinfo)),
1 + (1 + label-cnt (cinfo)))))))))

∧ label-cnt-big-enough (label-cnt (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))))),
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))))
                            62)
(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

64
(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
(no-duplicates (collect-labels (code (cinfo))))
∧ label-cnt-big-enough (label-cnt (cinfo), code (cinfo)))
→ no-duplicates (collect-labels (code (translate (cinfo,
cond-list,
stmt,
proc-list)))))

;; Note: many of the following lemmas may never be used, particularly the ones
Theorem: \texttt{translate-proc-list-assoc1}
\[(\text{definedp} \ (\text{subr}, \ \text{proc-list}) \land \text{ok-mg-def-plistp1} \ (\text{proc-list}, \ \text{proc-list2})) \rightarrow \ (\text{translate-def} \ (\text{assoc} \ (\text{subr}, \ \text{proc-list}), \ \text{proc-list2}) = \ \text{assoc} \ (\text{subr}, \ \text{translate-proc-list1} \ (\text{proc-list}, \ \text{proc-list2})))\]

Event: Disable \texttt{translate-proc-list-assoc1}.

Theorem: \texttt{translate-proc-list-assoc}
\[(\text{user-defined-procp} \ (\text{subr}, \ \text{proc-list}) \land \text{ok-mg-def-plistp} \ (\text{proc-list})) \rightarrow \ (\text{translate-def} \ (\text{assoc} \ (\text{subr}, \ \text{proc-list}), \ \text{proc-list}) = \ \text{assoc} \ (\text{subr}, \ \text{translate-proc-list} \ (\text{proc-list})))\]

Event: Disable \texttt{translate-proc-list-assoc}.

Theorem: \texttt{translate-proc-list-assoc2}
\[(\text{user-defined-procp} \ (\text{subr}, \ \text{proc-list}) \land \text{ok-mg-def-plistp} \ (\text{proc-list})) \rightarrow \ (\text{assoc} \ (\text{subr}, \ \text{translate-proc-list} \ (\text{proc-list})) = \ \text{translate-def} \ (\text{assoc} \ (\text{subr}, \ \text{proc-list}), \ \text{proc-list}))\]

Event: Disable \texttt{translate-proc-list-assoc2}.

Theorem: \texttt{translate-definedp1}
\[(\text{ok-mg-def-plistp1} \ (\text{lst1}, \ \text{lst2}) \land \text{definedp} \ (x, \ \text{lst1})) \rightarrow \ \text{definedp} \ (x, \ \text{translate-proc-list1} \ (\text{lst1}, \ \text{lst2}))\]

Event: Disable \texttt{translate-definedp1}.

Theorem: \texttt{assoc-mg-simple-variable-assignment-translate-proc-list}
\[\text{assoc} \ (\text{mg-simple-variable-assignment}, \ \text{translate-proc-list} \ (\text{proc-list})) = \ \text{MG-SIMPLE-VARIABLE-ASSIGNMENT-TRANSLATION}\]

Theorem: \texttt{assoc-mg-simple-constant-assignment-translate-proc-list}
\[\text{assoc} \ (\text{mg-simple-constant-assignment}, \ \text{translate-proc-list} \ (\text{proc-list})) = \ \text{MG-SIMPLE-CONSTANT-ASSIGNMENT-TRANSLATION}\]

Theorem: \texttt{assoc-mg-simple-variable-eq-translate-proc-list}
\[\text{assoc} \ (\text{mg-simple-variable-eq}, \ \text{translate-proc-list} \ (\text{proc-list})) = \ \text{MG-SIMPLE-VARIABLE-EQ-TRANSLATION}\]

Theorem: \texttt{assoc-mg-simple-constant-eq-translate-proc-list}
\[\text{assoc} \ (\text{mg-simple-constant-eq}, \ \text{translate-proc-list} \ (\text{proc-list})) = \ \text{MG-SIMPLE-CONSTANT-EQ-TRANSLATION}\]

66
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-pplist (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)))},

67
→ (cddr (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)
∧ ok-mg-def-plistp1 (proc-list, proc-list2))
→ 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) ∧ ok-mg-def-plistp (proc-list))
→ 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

68
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 (cadr (call-actuals ($stmt$)),
                                               mg-alist ($mg$-state))))),

Definition:
predefined-proc-call-clock \((\text{stmt}, \text{mg-state})\)
\[= (\text{clock-predefined-proc-call-sequence } (\text{call-name } (\text{stmt})))
+ \text{clock-predefined-proc-call-body-translation } (\text{stmt}, \text{mg-state}))
\]

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

;; Removed the definition of clock-r

**DEFINITION:**
\[
clock (\text{stmt}, \text{proc-list}, \text{mg-state}, n)
= \begin{cases} 
0 & \text{if } (n \simeq 0) \lor (\neg \text{normal } (\text{mg-state})) \\
\text{case on } \text{car } (\text{stmt}): \\
\text{case } = \text{no-op-mg} & 0 \\
\text{case } = \text{signal-mg} & 3 \\
\text{case } = \text{prog2-mg} & \begin{cases} 
\text{clock } (\text{prog2-left-branch } (\text{stmt}), \\
\text{proc-list}, \\
\text{mg-state}, \\
 n - 1) + 
\text{clock } (\text{prog2-right-branch } (\text{stmt}), \\
\text{proc-list}, \\
\text{mg-meaning } (\text{prog2-left-branch } (\text{stmt}), \\
\text{proc-list}, \\
\text{mg-state}, \\
 n - 1), \\
 n - 1)
\end{cases} \\
\text{case } = \text{loop-mg} & \begin{cases} 
\text{if } \neg \text{normal } (\text{mg-meaning } (\text{loop-body } (\text{stmt}), \\
\text{proc-list}, \\
\text{mg-state}, \\
 n - 1)) \\
\text{then if } \text{cc } (\text{mg-meaning } (\text{loop-body } (\text{stmt}), \\
\text{proc-list}, \\
\text{mg-state}, \\
 n - 1)) \\
\text{then } 3 + \text{clock } (\text{loop-body } (\text{stmt}), \\
\text{proc-list}, \\
\text{mg-state}, \\
 n - 1) \\
\text{else } 1 + \text{clock } (\text{loop-body } (\text{stmt}), \\
\text{proc-list}, \\
\text{mg-state}, \\
 n - 1)
\end{cases}
\end{cases}
\]

71
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)
\[ n - 1 \in \text{when-labels}(\text{stmt}) \]

\textbf{then if} normal(mg-meaning(when-handler(\text{stmt}),
\text{proc-list},
set-condition(mg-meaning(begin-body(\text{stmt}),
\text{proc-list},
mg-state,
\(n - 1\)),
'normal),
\(n - 1\))
\textbf{then} \text{clock}(\text{begin-body(\text{stmt}),
\text{proc-list},
mg-state,}
\(n - 1\))
+ (3 + \text{clock}(\text{when-handler(\text{stmt}),
\text{proc-list},
set-condition(mg-meaning(begin-body(\text{stmt}),
\text{proc-list},
mg-state,}
\(n - 1\)),
'normal),
\(n - 1\))
\textbf{else} \text{clock}(\text{begin-body(\text{stmt}),
\text{proc-list},
mg-state,}
\(n - 1\))
+ (2 + \text{clock}(\text{when-handler(\text{stmt}),
\text{proc-list},
set-condition(mg-meaning(begin-body(\text{stmt}),
\text{proc-list},
mg-state,}
\(n - 1\)),
'normal),
\(n - 1\)) \textbf{endif}

\textbf{elseif} normal(mg-meaning(begin-body(\text{stmt}),
\text{proc-list},
mg-state,}
\(n - 1\))
\textbf{then} 2 + \text{clock}(\text{begin-body(\text{stmt}),
\text{proc-list},
mg-state,}
\(n - 1\))
\textbf{else} \text{clock}(\text{begin-body(\text{stmt}),
\text{proc-list},
mg-state,}
\(n - 1\))

73
\[ \text{mg-state,} \]
\[ n - 1) \] \text{endif}

\text{case = proc-call-mg then}
\text{data-length (def-locals (fetch-called-def (stmt, proc-list)))}
\[ + \text{length (def-locals (fetch-called-def (stmt, proc-list)))} \]
\[ + \text{length (call-actuals (stmt))} \]
\[ + 1 \]
\[ + \text{clock (def-body (fetch-called-def (stmt, proc-list)), proc-list, make-call-environment (mg-state, stmt, fetch-called-def (stmt, proc-list)), n - 1)} \]
\[ + 5 \]
\[ + \text{if normal (mg-meaning (def-body (fetch-called-def (stmt, proc-list)), proc-list, make-call-environment (mg-state, stmt, fetch-called-def (stmt, proc-list)), n - 1)} \]
\[ \text{then 1} \]
\[ \text{else 3 endif} \]

\text{case = predefined-proc-call-mg then}
\text{predefined-proc-call-clock (stmt, mg-state)}
\text{otherwise 0 endcase endif}

\text{THEOREM: clock-prog2}
\text{(car (stmt) = 'prog2-mg)}
\[ \rightarrow \text{(clock (stmt, proc-list, mg-state, n)} \]
\[ = \text{if } (n \not\equiv 0) \land \text{normal (mg-state)} \]
\[ \text{then clock (prog2-left-branch (stmt), proc-list, mg-state, n - 1)} \]
\[ + \text{clock (prog2-right-branch (stmt), proc-list, mg-meaning (prog2-left-branch (stmt), proc-list, mg-state, n - 1)}, n - 1) \]
\[ \text{else 0 endif}) \]
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))
      = 'leave
      then 3 + clock (if-body (stmt),
                     proc-list,
                     mg-state,
                     n − 1)
      else 1 + clock (if-body (stmt),
                      proc-list,
                      mg-state,
                      n − 1) endif
else 0 endif)
\[\text{then } 5 + \text{clock(if-false-branch}(stmt), \text{proc-list}, \text{mg-state}, n - 1)\]

\[\text{else } 4 + \text{clock(if-false-branch}(stmt), \text{proc-list}, \text{mg-state}, n - 1) \text{ endif}\]

\[\text{elseif normal(mg-meaning(if-true-branch}(stmt), \text{proc-list}, \text{mg-state}, n - 1))}\]

\[\text{then } 5 + \text{clock(if-true-branch}(stmt), \text{proc-list}, \text{mg-state}, n - 1)\]

\[\text{else } 3 + \text{clock(if-true-branch}(stmt), \text{proc-list}, \text{mg-state}, n - 1) \text{ endif}\]

\[\text{else 0 endif}\]

**Theorem:** clock-begin

\((\text{car}(stmt) = \text{'begin-mg}) \rightarrow (\text{clock}(stmt, \text{proc-list}, \text{mg-state}, n) = \text{if}(n \not\equiv 0) \land \text{normal}(\text{mg-state})\]

\[\text{then if cc(mg-meaning(begin-body}(stmt), \text{proc-list}, \text{mg-state}, n - 1)) \in \text{when-labels}(stmt)\]

\[\text{then if normal(mg-meaning(when-handler}(stmt), \text{proc-list}, \text{set-condition}(mg-meaning(begin-body}(stmt), \text{proc-list}, \text{mg-state}, n - 1), \text{'normal}, n - 1))\]

\[\text{then clock(begin-body}(stmt), \text{proc-list}, \text{mg-state}, n - 1)\]
Theorem: clock-proc-call

\( \text{car}\ (\text{stmt}) = \text{'proc-call-mg} \) \\
\rightarrow \ (\text{clock}\ (\text{stmt}, \text{proc-list}, \text{mg-state}, n) \\
= \quad \text{if } (n \not\equiv 0) \land \text{normal}\ (\text{mg-state}) \\
\text{then} \ \text{data-length}\ (\text{def-locals}\ (\text{fetch-called-def}\ (\text{stmt}, \text{proc-list}))) \\
+ \ \text{length}\ (\text{def-locals}\ (\text{fetch-called-def}\ (\text{stmt}, \text{proc-list}))) \\
+ \ \text{length}\ (\text{call-actuals}\ (\text{stmt})) \\
+ \ 1 \\
+ \ \text{clock}\ (\text{def-body}\ (\text{fetch-called-def}\ (\text{stmt}, \text{proc-list})), \\
\text{proc-list}, \\
\text{mg-state}, \\
\text{set-condition}\ (\text{mg-meaning}\ (\text{begin-body}\ (\text{stmt}), \text{proc-list}, \\
\text{mg-state}, \\
n - 1), \\
\text{'normal}), \\
n - 1)) \\
\text{else}\ \text{clock}\ (\text{begin-body}\ (\text{stmt}), \text{proc-list}, \text{mg-state}, \\
n - 1) \\
+ \ (2 + \text{clock}\ (\text{when-handler}\ (\text{stmt}), \\
\text{proc-list}, \\
\text{set-condition}\ (\text{mg-meaning}\ (\text{begin-body}\ (\text{stmt}), \text{proc-list}, \text{mg-state}, \\
n - 1), \\
\text{'normal}), \\
n - 1)) \\
\text{endif} \\
\text{elseif}\ \text{normal}\ (\text{mg-meaning}\ (\text{begin-body}\ (\text{stmt}), \text{proc-list}, \text{mg-state}, \\
n - 1)) \\
\text{then} \ 2 + \text{clock}\ (\text{begin-body}\ (\text{stmt}), \text{proc-list}, \text{mg-state}, \\
n - 1) \\
\text{else}\ \text{clock}\ (\text{begin-body}\ (\text{stmt}), \text{proc-list}, \text{mg-state}, \\
n - 1) \\
\text{endif} \\
\text{else}\ \ 0 \ \text{endif} \)
make-call-environment (mg-state, stmt, fetch-called-def (stmt, proc-list)),

\( n - 1 \) + 5 + \text{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 \}) \text{ then 1 else 3 endif}
else 0 endif)

\textbf{Theorem:} clock-predefined-proc-call
(car (stmt) = \texttt{predefined-proc-call-mg}) \rightarrow (\text{clock (stmt, proc-list, mg-state, n)}
\begin{align*}
&= \text{if} (n \not\equiv 0) \land \text{normal (mg-state)} \\
&\quad \text{then predefined-proc-call-clock (stmt, mg-state)} \\
&\quad \text{else 0 endif})
\end{align*}

\textbf{Definition:}
\text{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 (\texttt{\textquotesingle{}c-c\textquotesingle{}, mg-cond-to-p-nat (cc (mg-state), cond-list)}), MG-MAX-CTRL-STK-SIZE, MG-MAX-TEMP-STK-SIZE, MG-WORD-SIZE, \texttt{\textquotesingle{}run\textquotesingle{}})

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

\textbf{Theorem:} map-up-vars-inverts-map-down
(all-cars-unique (mg-vars)
\[ \text{Thorem: } \text{cond-subset-preserves-ok-cc} \]

\[ ((cc \neq '\text{leave}) \wedge \text{cond-subsetp (r-cond-list, t-cond-list)} \wedge \text{ok-cc (cc, r-cond-list))} \rightarrow \text{ok-cc (cc, t-cond-list)} \]

\[ \text{Theorem: map-up-inverts-map-down} \]

\[ \text{(all-cars-unique (mg-alist (mg-state)))} \wedge \text{ok-mg-statep (mg-state, r-cond-list)} \wedge \text{cond-subsetp (r-cond-list, t-cond-list)} \wedge \text{mg-vars-list-ok-in-p-state (mg-alist (mg-state), bindings (top (ctrl-stk)), temp-stk)} \wedge \text{no-p-aliasing (bindings (top (ctrl-stk)), mg-alist (mg-state))} \wedge (cc (mg-state) \neq '\text{leave}) \wedge (\neg \text{resource-errorp (mg-state)})] \rightarrow \text{(map-up (map-down (mg-state, proc-list, ctrl-stk, temp-stk, addr, t-cond-list), signature (mg-alist (mg-state)), t-cond-list))} = \text{mg-state} \]

;; These are used in the proofs which follow!

\[ \text{Theorem: call-exact-time-hyps1} \]

\[ ((\text{car (stmt)}) = '\text{proc-call-mg}) \wedge \text{ok-mg-statement (stmt, r-cond-list, name-alist, proc-list)} \wedge \text{ok-mg-def-plistp (proc-list))} \rightarrow \text{ok-mg-statement (def-body (fetch-called-def (stmt, proc-list), make-cond-list (fetch-called-def (stmt, proc-list), make-name-alist (fetch-called-def (stmt, proc-list)), proc-list))} \]
Theorem: resources-adequate-temp-stk-not-max
\(\lnot \text{resources-inadequatep}(\text{stmt},\)
\(\text{proc-list},\)
\(\text{list (length (temp-stk), p-ctrl-stk-size (ctrl-stk))))\)
\(\rightarrow ((\text{length (temp-stk)} < \text{MG-MAX-TEMP-STK-SIZE}) = \text{t})\)

Theorem: plus-difference-cancellation
\(((x - y) \neq 0) \rightarrow (((x - y) + y) = \text{fix}(x))\)

Theorem: lessp-difference-lemma1
\(((n < (r + l)) \land (r < (m - l))) \rightarrow ((n < m) = \text{t})\)

Theorem: resources-adequate-temp-stk-not-max2
\(\lnot \text{resources-inadequatep}(\text{stmt},\)
\(\text{proc-list},\)
\(\text{list (length (temp-stk), p-ctrl-stk-size (ctrl-stk))))\)
\(\land (\text{car (stmt)} = \text{'predefined-proc-call-mg})\)
\(\land (n < \text{(predefined-proc-call-temp-stk-requirement (call-name (stmt))}}\)
\(\text{+ length (temp-stk))))))\)
\(\rightarrow ((n < \text{MG-MAX-TEMP-STK-SIZE}) = \text{t})\)

Theorem: lessp-difference-lemma3
\(((n \leq p) \land (p < (m - c))) \rightarrow ((m < (n + c)) = \text{f})\)

Event: Disable lessp-difference-lemma3.

Theorem: resources-adequate-ctrl-stk-not-max
\(\lnot \text{resources-inadequatep}(\text{stmt},\)
\(\text{proc-list},\)
\(\text{list (length (temp-stk), p-ctrl-stk-size (ctrl-stk))))\)
\(\land (\text{car (stmt)} = \text{'predefined-proc-call-mg})\)
\(\land (n \leq \text{predefined-proc-call-p-frame-size (call-name (stmt))})\)
\(\rightarrow ((\text{MG-MAX-CTRL-STK-SIZE} < (n + p-ctrl-stk-size (ctrl-stk))) = \text{f})\)

Theorem: lessp-transitive3
\(((y < n) \land (n < (m - x))) \rightarrow (((x + y) < m) = \text{t})\)

Theorem: lessp-difference
\((y < (m - x)) \rightarrow (((x + y) < m) = \text{t})\)

Theorem: resources-proc-call-temp-stk-ok
\(\text{(car (stmt)} = \text{'proc-call-mg})\)
\(\land (\lnot \text{resources-inadequatep}(\text{stmt},\)
\(\text{proc-list},\)
\(\text{list (length (temp-stk),}\)
\(\text{p-ctrl-stk-size (ctrl-stk))))\)
\[ p\text{-ctrl-stk-size}(ctrl-stk) \]

\[ \rightarrow (((\text{length}(\text{temp-stk})
+ \\text{data-length}(\text{def-locals}(\text{fetch-called-def}(stmt, proc-list))))
+ \text{length}(\text{def-locals}(\text{fetch-called-def}(stmt, proc-list))))
+ \text{length}(\text{call-actuals}(stmt)) < \text{MG-MAX-TEMP-STK-SIZE}\]

\[ = t \]

**EVENT:** Disable resources-proc-call-temp-stk-ok.

**THEOREM:** user-defined-def-locals-nil

\[ \text{ok-mg-def-plistp}(proc-list) \land (\text{car}(stmt) = \text{'proc-call-mg}) \land \text{ok-mg-statement}(stmt, r-cond-list, name-alist, proc-list)) \rightarrow \text{length}(\text{caddr}(\text{assoc}(\text{call-name}(stmt), \text{translate-proc-list}(proc-list)))) = 0 \]

**EVENT:** Disable user-defined-def-locals-nil.

**THEOREM:** user-defined-def-formals-rewrite

\[ \text{ok-mg-def-plistp}(proc-list) \land (\text{car}(stmt) = \text{'proc-call-mg}) \land \text{ok-mg-statement}(stmt, r-cond-list, name-alist, proc-list)) \rightarrow \text{length}(\text{cadr}(\text{assoc}(\text{call-name}(stmt), \text{translate-proc-list}(proc-list)))) = \text{length}(\text{def-locals}(\text{assoc}(\text{call-name}(stmt), proc-list))) + \text{length}(\text{def-formals}(\text{assoc}(\text{call-name}(stmt), proc-list)))) \]

**EVENT:** Disable user-defined-def-formals-rewrite.

**THEOREM:** difference-preserves-lessp2

\[ (n < m) \rightarrow (((n - k) < m) = t) \]

**THEOREM:** plus-lessp

\[ ((n + m + x) < (m + n)) = f \]

**THEOREM:** resources-proc-call-ctrl-stk-ok

\[ ((\text{car}(stmt) = \text{'proc-call-mg}) \land (\neg \text{resources-inadequatep}(stmt, proc-list)) \land \text{list}(\text{length}(\text{temp-stk}), p\text{-ctrl-stk-size}(ctrl-stk)))) \land \text{ok-mg-statement}(stmt, r-cond-list, name-alist, proc-list) \land \text{ok-mg-def-plistp}(proc-list) \]
∧ user-defined-procp (subr, proc-list))
→ ((MG-MAX-CTRL-STK-SIZE
    < (2
    + length (cadr (assoc (call-name (stmt),
                         translate-proc-list (proc-list)))))
    + length (caddr (assoc (call-name (stmt),
                          translate-proc-list (proc-list)))))
    + p-ctrl-stk-size (ctrl-stk))) = f

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

EVENT: Make the library "c5".
Index

add-code-doesnt-affect-other-fields, 27
all-cars-unique, 78, 79
all-labels-unique, 48, 49, 52
all-labels-unique-append, 48
all-labels-unique-reduction, 48
all-labels-unique-reduction2, 48
all-labels-unique-reduction3, 48
array-length, 2, 4, 5
assoc-mg-array-element-assignment-translate-proc-list, 67
assoc-mg-boolean-and-translate-proc-list, 67
assoc-mg-boolean-not-translate-proc-list, 67
assoc-mg-boolean-or-translate-proc-list, 67
assoc-mg-index-array-translate-proc-list, 67
assoc-mg-integer-add-translate-proc-list, 67
assoc-mg-integer-extract-translate-proc-list, 67
assoc-mg-integer-minus-translate-proc-list, 67
assoc-mg-integer-semantic-translate-proc-list, 67
assoc-mg-simple-constant-assign-ment-translate-proc-list, 66
assoc-mg-simple-constant-eq-tranlate-proc-list, 66
assoc-mg-simple-variable-assign-ment-translate-proc-list, 66
assoc-mg-simple-variable-eq-tranlate-proc-list, 66
assoc-user-defined-proc2, 67
begin-body, 20, 23, 30, 31, 42, 60, 61, 72, 73, 76, 77
begin-translation, 23
bindings, 2, 78, 79
call-actuals, 2, 7–10, 69, 70, 74, 77, 81
call-conds, 7, 21, 24, 51
call-exact-time-hyps1, 79
call-name, 7, 10, 69, 71, 80–82
call-translation, 23
car-definedp-defined-proc, 68
car-definedp-defined-proc1, 68
car, 49, 71, 73, 75, 76, 78, 79
cinfop, 26
clock, 71–78
clock-begin, 76
clock-if, 75
clock-loop, 75
clock-predefined-proc-call, 78
clock-predefined-proc-call-body -translation, 69, 71
clock-predefined-proc-call-sequence, 68, 71
clock-proc-call, 77
clock-prog2, 74
code-add-code-commute, 27
code-always-plistp, 25
code-doesnt-affect-other-fields, 26
collect-labels, 47–49, 52–57, 59, 60, 65
collect-labels-distributes, 48
collect-labels-plistp, 47
collect-labels-predefined-proc-call-code-nil, 53
collect-labels-push-actuals-code-nil, 53
collect-labels-push-local-array-values-code-nil, 53
collect-labels-push-locals-addreses-code-nil, 53
nullify, 3, 26–28, 31–35, 37, 39–42, 47
nullify-cancels-add-code, 27
nullify-code-nil, 27
nullify-doesnt-affect-proc-call-code, 27
nullify-induction-hint, 27–31
nullify-translate-idempotence, 26
nullify-translate-idempotence2, 26
nullify-translate-leaves-nearly-equal, 26
ok-cc, 79
ok-cinfop, 4, 25, 31–38, 40–42, 47, 50
ok-cond-list, 49, 50
ok-mg-def-plistp, 49, 66–68, 79, 81
ok-mg-def-plistp1, 66, 68