```
; ------------------------------------------------------------
; was lr-eval5-1.events
; ------------------------------------------------------------
```

EVENT: Start with the library `"app-c-d-e"` using the compiled version.

THEOREM: axiom-53
subrp (*fn*) $\rightarrow$ (formals (*fn*) = **f**)

EVENT: Disable proper-p-statep-restructuring.

```
;; Function for testing s->r
```

DEFINITION:
change-elements (*list*)
= **if** listp (*list*)
  then if truep (car (*list*)) **then** cons (FALSE, change-elements (cdr (*list*)))
    else cons (TRUE, change-elements (cdr (*list*))) **endif**

elseif truep ($list$) **then** FALSE
else TRUE **endif**

EVENT: Disable deposit.

EVENT: Disable fetch.

EVENT: Disable add-addr.

EVENT: Disable sub-addr.

EVENT: Disable offset.

EVENT: Disable area-name.

EVENT: Disable errorp.

EVENT: Disable p-current-program.

```
;; The following is inspired by the lemma length-put of Piton.
;; Now in Piton-basis  A. Flatau  8-Oct-1990
;(prove-lemma MY-LENGTH-PUT (rewrite)
;  (equal (length (put val n lst))
;  (if (lessp n (length lst))
;      (length lst)
;      (add1 n)))
;  ((enable put)))
;
;(disable my-length-put)

;; This is similar to the lemma GET-PUT from Piton, but for the commented
;; out hypothesis.
```

THEOREM: my-get-put
$((k \in \mathbf{N}) \land (n \in \mathbf{N}))$
$\rightarrow \quad (\text{get}(k, \text{put}(val, n, lst))$
$\quad = \quad \textbf{if } k = n \textbf{ then } val$
$\quad\quad \textbf{else } \text{get}(k, lst) \textbf{ endif})$

EVENT: Disable my-get-put.

THEOREM: listp-cdr-p-frame
listp (cdr (p-frame (*bindings*, *ret-pc*)))

THEOREM: equal-cddr-p-frame-nil
cddr (p-frame (*bindings*, *ret-pc*)) = **nil**

```
#||
;; The following is used to test handling of temp variables
(defn FOO (state name)
  (let ((prog (app name state)))
    (cons state (cons (car prog) (cons (cadr prog) (caddr prog))))))

;(setq ss
;      (logic->s '(change-elements (cons '*1*true (app x y)))
;  '((x . (*1*true *1*true . *1*false))
;    (y . (*1*true . *1*false)))
;   '(change-elements app)))
;(setq lrs (s->lr ss 'main 50 50 50 32))
;(setq foop
;      '(FOO (STATE NAME)
;     ((APP NAME STATE)
;      (CDR ((TEMP-FETCH) (APP NAME STATE))))
;     (CONS STATE
;   (CONS (CAR ((TEMP-EVAL) (APP NAME STATE)))
; (CONS (CAR ((TEMP-EVAL)
;     (CDR ((TEMP-FETCH) (APP NAME STATE)))))
;       (CAR (CDR ((TEMP-FETCH)
;  (CDR ((temp-fetch)
;        (APP NAME STATE))))))))))
;
;(setq ss1 (s-state (s-expr ss)
;    (s-params ss)
;    (s-temps ss)
;    (s-consts ss)
;    (put-assoc (cdr foop) 'foo (s-progs ss))
;    'run))
;
;(setq ss2 (s-state '(FOO (CHANGE-ELEMENTS (CONS '(ADDR (heap . 4))
; (APP ((temp-eval) X) Y)))
;  ((temp-fetch) X))
;    (s-params ss1)
;    (make-temps-entries '(x))
;    (s-consts ss1)
;    (s-progs ss1)
```

```
;     'run))
||#
```

DEFINITION:
s-l-eval-equiv-hyps (*flag*, s, c)
= (s-good-statep (s, c)
  ∧ good-posp (*flag*, s-pos (s), s-body (s-prog (s)))
  ∧ (s-err-flag (s-eval (*flag*, s, c)) = `'run`))

DEFINITION:
s-l-eval-flag-run-hyps (*flag*, s, c)
= (s-good-statep (s, c)
  ∧ s-all-temps-setp (*flag*,
    if *flag* = `'list` **then** s-expr-list (s)
    else s-expr (s) **endif**,
    temp-alist-to-set (s-temps (s)))
  ∧ s-all-progs-temps-setp (s-progs (s))
  ∧ **if** *flag* = `'list`
    then f ∉ l-eval (*flag*,
      s-expand-temps (*flag*, s-expr-list (s)),
      s-params (s),
      c)
    else l-eval (*flag*,
      s-expand-temps (*flag*, s-expr (s)),
      s-params (s),
      c) **endif**
  ∧ s-check-temps-setp (s-temps (s)))

```
;; ***** The LR-level (R for Resource, L for logic).  *****
;; We used to have an LR-STATE shell.  Now we just use a P-STATE shell.
;; However we refer to LR-STATES which are P-STATEs with LR level programs.
;; The function LR->P compiles an LR-STATE to a Piton state, by compiling
;; the programs and converting the P-PC to a Piton PC.
;; We use P-STATE shells instead of LR-STATE shell because we used to have
;; define functions analogous to P-OBJECTP (and functions that called
;; P-OBJECTP) that took LR-STATES or parts thereof.

;; We use the Piton notion of a PROPER state.  It should be the case that
;; all the LR-STATEs we are interested in are PROPER-P-STATEPs after we
;; apply LR->P to them.

;; An LR PC object is a combination of a Piton PC object and an S level
;; S-PNAME and S-POS.  The translation of (s-pname s) and (s-pos s) from
;; the S level is: (TAG 'PC (CONS (S-PNAME S) (S-POS S)))
```

```
;; Each element of P-PROG-SEGMENT is a program.  A program is a list
;; of the form:
;;
;; (name (formal1 formal2 ... formaln)
;;       ((temp1 init1)
;;        ...
;;        (tempk initk))
;;       body)
;;
;; The name and each formal and temp is a symbol.  The initial values
;; of the temps are tagged values.  Body is a form similar to that for
;; the S level, but temporary expressions have been replaced the name of
;; a temporary variable added to them
;; e.g. ((S-TEMP-EVAL) <expr>) -> ((S-TEMP-EVAL) <expr> <var>).
;; In the case of (S-TEMP-FETCH) <expr> is never used but we put it
;; in for consistency and so it is easier to convert back to s-states.
;; Also the numbers in the S level quote constructs have been replaced
;; by data-addresses that should contain pointers to the appropriate
;; structure in the heap.

;; Roughly speaking, a function application of FUN binds the formals to
;; the top n elements of the temp-stk (removing them from that stack and
;; building a ctrl-stk frame), binds the temps to the corresponding tagged
;; values (also in the ctrl-stk frame), and executes each instruction.

;; Producing LR-code from S-code.
```

DEFINITION: LR-UNDEFINED-TAG = 0

```
; Used in node to indicate
; uninitialized temporary variable
```

DEFINITION: LR-INIT-TAG = 1

```
; Used in initial nodes that have
; not been used
```

DEFINITION: LR-FALSE-TAG = 2

DEFINITION: LR-TRUE-TAG = 3

DEFINITION: LR-ADD1-TAG = 4

DEFINITION: LR-CONS-TAG = 5

DEFINITION: LR-PACK-TAG = 6

DEFINITION: LR-MINUS-TAG = 7

DEFINITION: LR-HEAP-NAME = 'heap

DEFINITION: LR-NODE-SIZE = 4

DEFINITION: LR-UNDEF-ADDR = tag ('addr, '(heap . 0))

DEFINITION: LR-F-ADDR = add-addr (LR-UNDEF-ADDR, LR-NODE-SIZE)

DEFINITION: LR-T-ADDR = add-addr (LR-F-ADDR, LR-NODE-SIZE)

DEFINITION: LR-0-ADDR = add-addr (LR-T-ADDR, LR-NODE-SIZE)

DEFINITION: LR-FP-ADDR = tag ('addr, '(free-ptr . 0))

DEFINITION: LR-ANSWER-ADDR = tag ('addr, '(answer . 0))

DEFINITION: lr-fetch-fp (*data-seg*) = fetch (LR-FP-ADDR, *data-seg*)

DEFINITION:
LR-MINIMUM-HEAP-SIZE = offset (add-addr (LR-0-ADDR, LR-NODE-SIZE))

```
;; The heap is a (presumably large) Piton data area.  It contains Nodes
;; which are four words.  One word is for the tag, one for the reference
;; count, and two for the contents.  Some data-types only require one word
;; for the contents (e.g. NUMBERPs) in that case one word is wasted.  Some
;; (user-defined) data-types require more than two words. In this case the
;; second word is a pointer to another node.  This contains up to three
;; words of data, the fourth word (if the data type needs more that four
;; words) is used to link another node with the same format.  The heap is
;; the Piton data area named HEAP.

;; LR-NEW-NODE returns another node to be stuck in memory
```

DEFINITION:
lr-new-node (*tag*, *ref-count*, *value1*, *value2*)
= list (*tag*, *ref-count*, *value1*, *value2*)

DEFINITION: LR-REF-COUNT-OFFSET = 1

DEFINITION: LR-CAR-OFFSET = 2

DEFINITION: LR-CDR-OFFSET = 3

DEFINITION: LR-UNPACK-OFFSET = 2

DEFINITION: LR-UNBOX-NAT-OFFSET = 2

DEFINITION: LR-NEGATIVE-GUTS-OFFSET = 2

DEFINITION:
lr-boundary-offsetp (*offset*) = ((*offset* **mod** LR-NODE-SIZE) = 0)

DEFINITION:
lr-boundary-nodep (*node*) = lr-boundary-offsetp (offset (*node*))

DEFINITION:
lr-nodep (*addr*, *data-seg*)
= ((type (*addr*) = `'addr`)
  ∧ (cddr (*addr*) = **nil**)
  ∧ listp (*addr*)
  ∧ adpp (untag (*addr*), *data-seg*)
  ∧ lr-boundary-nodep (*addr*)
  ∧ (area-name (*addr*) = LR-HEAP-NAME))

```
;; LR-GOOD-POINTERP checks that an addr is a node and its ref count field
;; is a natural.
```

DEFINITION:
lr-good-pointerp (*addr*, *data-seg*)
= (lr-nodep (*addr*, *data-seg*)
  ∧ (type (fetch (add-addr (*addr*, LR-REF-COUNT-OFFSET), *data-seg*))
    = `'nat`))

DEFINITION:
lr-expr (*p*) = cur-expr (offset (p-pc (*p*)), program-body (p-current-program (*p*)))

EVENT: Disable lr-expr.

DEFINITION:
lr-expr-list (*p*)
= restn (car (last (offset (p-pc (*p*)))),
    cur-expr (butlast (offset (p-pc (*p*))),
        program-body (p-current-program (*p*))))

EVENT: Disable lr-expr-list.

```
;;; Debugging Stuff.
```

DEFINITION:
mark-instr (*instruction-list*, *n*)

```
=  if n ≃ 0
   then cons (list ('pc->, car (instruction-list)), cdr (instruction-list))
   else cons (car (instruction-list),
              mark-instr (cdr (instruction-list), n − 1)) endif
```

DEFINITION:
fix-program-segment (*programs*, *pc*)

```
=  if listp (programs)
   then let prog be car (programs)
        in
        if car (prog) = area-name (pc)
        then cons (append (list (car (prog), cadr (prog), caddr (prog)),
                           mark-instr (program-body (prog), offset (pc))),
                   fix-program-segment (cdr (programs), pc))
        else cons (car (prog),
                   fix-program-segment (cdr (programs), pc)) endif endlet
   else nil endif
```

DEFINITION:
fix-data-segment (*data-segment*)

```
=  put-value (append (firstn (offset (lr-fetch-fp (data-segment)),
                              value (LR-HEAP-NAME, data-segment)),
                      length (value (LR-HEAP-NAME, data-segment))
                      − offset (lr-fetch-fp (data-segment))),
              LR-HEAP-NAME,
              data-segment)
```

DEFINITION:
find-non-proper-instr (*lst*, *name*, *p*)

```
=  if listp (lst)
   then if legal-labelp (car (lst))
           ∧ proper-p-instructionp (unlabel (car (lst)), name, p)
        then find-non-proper-instr (cdr (lst), name, p)
        else car (lst) endif
   else nil endif
```

DEFINITION:
find-non-proper-programs (*progs*, *p*)

```
=  if listp (progs)
   then if proper-p-programp (car (progs), p)
        then cons (name (car (progs)),
                   find-non-proper-programs (cdr (progs), p))
        else cons (list ('not,
```

```
                              name (car (progs)),
                              find-non-proper-instr (program-body (car (progs)),
                                                     name (car (progs)),
                                                     p)),
                          find-non-proper-programs (cdr (progs), p)) endif
    else nil endif
```

DEFINITION:

```
pps (state)
=   list ('p-state,
        p-pc (state),
        p-ctrl-stk (state),
        p-temp-stk (state),
        let p be p-current-program (state)
        in
        append (list (name (p), formal-vars (p), temp-var-dcls (p)),
                mark-instr (program-body (p), offset (p-pc (state)))) endlet,
        fix-data-segment (p-data-segment (state)),
        p-psw (state))
```

DEFINITION:

```
lr-nodify-tag (tag)
=   if untag (tag) = LR-FALSE-TAG then 'false
    elseif untag (tag) = LR-TRUE-TAG then 'true
    elseif untag (tag) = LR-ADD1-TAG then 'add1
    elseif untag (tag) = LR-CONS-TAG then 'cons
    elseif untag (tag) = LR-PACK-TAG then 'pack
    else 'unknown endif
```

DEFINITION:

```
lr-nodify (number, nodes, final)
=   if listp (nodes)
    then cons (list ('node,
                     number,
                     lr-nodify-tag (car (nodes)),
                     caddr (nodes),
                     cadddr (nodes)),
               lr-nodify (number + LR-NODE-SIZE, cddddr (nodes), final))
    else final endif
```

DEFINITION:

```
lr-fix-data-segment (data-seg)
=   put-value (lr-nodify (0,
                          firstn (offset (lr-fetch-fp (data-seg)),
                                  value (LR-HEAP-NAME, data-seg)),
```

```
                    length (value (LR-HEAP-NAME, data-seg))
                    -   offset (lr-fetch-fp (data-seg))),
               LR-HEAP-NAME,
               data-seg)
```

DEFINITION:

```
lrps (state)
=  p-state (p-pc (state),
            p-ctrl-stk (state),
            p-temp-stk (state),
            p-prog-segment (state),
            lr-fix-data-segment (p-data-segment (state)),
            p-max-ctrl-stk-size (state),
            p-max-temp-stk-size (state),
            p-word-size (state),
            p-psw (state))
```

`;; Returns the object denoted by addr in the heap.`

DEFINITION:

```
lr-abs (addr, data-seg, n)
=  if n ≃ 0 then nil
   else let tag be untag (fetch (addr, data-seg))
        in
        if tag = LR-FALSE-TAG then f
        elseif tag = LR-TRUE-TAG then t
        elseif tag = LR-ADD1-TAG
        then untag (fetch (add-addr (addr, LR-UNBOX-NAT-OFFSET),
                           data-seg))
        elseif tag = LR-CONS-TAG
        then cons (lr-abs (fetch (add-addr (addr, LR-CAR-OFFSET),
                                  data-seg),
                           data-seg,
                           n - 1),
                   lr-abs (fetch (add-addr (addr, LR-CDR-OFFSET),
                                  data-seg),
                           data-seg,
                           n - 1))
        elseif tag = LR-PACK-TAG
        then pack (lr-abs (fetch (add-addr (addr,
                                            LR-UNPACK-OFFSET),
                                  data-seg),
                           data-seg,
                           n - 1))
        else - untag (fetch (add-addr (addr,
```

LR-NEGATIVE-GUTS-OFFSET),
data-seg)) **endif endlet endif**

DEFINITION:
top-stk (*p-or-p-state*)
= **let** *temp-stk* **be** **if** p-statep (*p-or-p-state*)
then p-temp-stk (*p-or-p-state*)
else p-temp-stk (*p-or-p-state*) **endif**,
data-segment **be** **if** p-statep (*p-or-p-state*)
then p-data-segment (*p-or-p-state*)
else p-data-segment (*p-or-p-state*) **endif**
in
lr-abs (top (*temp-stk*), *data-segment*, 1000) **endlet**

```
;; This is accessed by the Piton accessors: NAME, FORMAL-VARS, TEMP-VAR-DCLS
;; and PROGRAM-BODY.  Also LOCAL-VARS.
```

DEFINITION:
lr-make-program (*name*, *formals*, *temps*, *body*)
= cons (*name*, cons (*formals*, cons (*temps*, *body*)))

```
#||
stolen from matt kaufmann's code for gensym, but modified to probably be
less useful but simplier.

here is a sequence of events for generating a new symbol.
the main function is near the end, and is called gensym.
gensym returns a pair the new symbol and the next number list to try.
here are some examples:

>(r-loop)

trace mode: off   abbreviated output mode:  on
type ? for help.
*(gensym (unpack 'a*) '(49) '(a*0 a*1 a*2 a*3))
 '(a*4 53)
*(gensym (unpack 'a*) '(53) '(a*0 a*1 a*2 a*3 a*4))
 '(a*5 54)
*(gensym (unpack 'a*) '(50) '(a*2))
 '(a*3 52)
*(gensym (unpack 'a*) '(50) '(a*0))
 '(a*2 51)
*(gensym (unpack 'a) '(48) '(a*0 a*1))
 '(a0 49)
*(gensym (unpack 'a) '(48) '(a b))
```

```
 '(a0 49)
*(gensym (unpack 'a*2*) '(51) '(a*2*3))
 '(a*2*4 53)
*(gensym (unpack 'b*) '(50) '(a*0 a*1 a*2 a*3))
 '(b*2 51)
*ok
exiting r-loop.
nil

||#
```

DEFINITION: ASCII-0 = `48`

DEFINITION: ASCII-1 = `49`

DEFINITION: ASCII-9 = `57`

DEFINITION: ASCII-DASH = `45`

DEFINITION: LIST-ASCII-0 = list (ASCII-0)

DEFINITION: LIST-ASCII-1 = list (ASCII-1)

DEFINITION:
increment-numlist (*numlist*)
= **if** listp (*numlist*)
  then if car (*numlist*) = ASCII-9
    then cons (ASCII-0, increment-numlist (cdr (*numlist*)))
    else cons (1 + car (*numlist*), cdr (*numlist*)) **endif**
  else LIST-ASCII-1 **endif**

DEFINITION:
make-symbol (*initial*, *digit-list*)
= pack (append (append (*initial*, *digit-list*), `0`))

EVENT: Disable make-symbol.

DEFINITION:
count-codelist1 (*numlist*)
= **if** listp (*numlist*)
  then car (*numlist*) + (`10` $*$ count-codelist1 (cdr (*numlist*)))
  else `0` **endif**

DEFINITION:
subseqp (*list1*, *list2*)
= ((length (*list2*) $\not<$ length (*list1*))
  $\wedge$ (firstn (length (*list1*), *list2*) = *list1*))

EVENT: Disable subseqp.

DEFINITION:
count-codelist (*initial*, *ascii-list*)
= **if** subseqp (*initial*, *ascii-list*)
  then count-codelist1 (restn (length (*initial*), *ascii-list*))
  else 0 **endif**

EVENT: Disable count-codelist.

DEFINITION:
max-count-codelist (*initial*, *list*)
= **if** listp (*list*)
  then max (count-codelist (*initial*, unpack (car (*list*))),
      max-count-codelist (*initial*, cdr (*list*)))
  else 0 **endif**

THEOREM: increment-num-list-count-code-list1
count-codelist1 (*num-list*) < count-codelist1 (increment-numlist (*num-list*))

THEOREM: subseqp-append
subseqp (plist (x), append (x, *anything*))

THEOREM: count-codelist-make-symbol
(x = make-symbol (*initial*, *num-list*))
$\rightarrow$ (count-codelist (plist (*initial*), unpack (x))
  = count-codelist1 (*num-list*))

THEOREM: member-make-symbol-max-count-code-list
(make-symbol (*initial*, *num-list*) $\in$ *atom-list*)
$\rightarrow$ (max-count-codelist (plist (*initial*), *atom-list*)
  $\not<$ count-codelist1 (*num-list*))

```
;; Returns a pair, the new symbol and the next number to use
```

DEFINITION:
gensym (*initial*, *num-list*, *atom-list*)
= **if** make-symbol (*initial*, *num-list*) $\in$ *atom-list*
  then gensym (*initial*, increment-numlist (*num-list*), *atom-list*)
  else cons (make-symbol (*initial*, *num-list*),
      increment-numlist (*num-list*)) **endif**

THEOREM: gensym-is-new
car (gensym (*initial*, *num-list*, *atom-list*)) $\notin$ *atom-list*

```
;; MAKE-TEMP-NAME-ALIST takes a temps-alist triple a la S-TEMPS and
;; returns an alist with entries of the form:
;; (<temp expression> . <variable>) where <variable> is guaranteed to
;; occur only once in the resulting alist and is guaranteed not to occur
;; in FORMALS.
```

DEFINITION:
lr-make-temp-name-alist-1 (*initial*, *num-list*, *temp-list*, *formals*)
= **if** listp (*temp-list*)
  then let *gensym* **be** gensym (*initial*, *num-list*, *formals*)
    in
    cons (cons (car (*temp-list*), car (*gensym*)),
      lr-make-temp-name-alist-1 (*initial*,
        cdr (*gensym*),
        cdr (*temp-list*),
        formals)) **endlet**
  else nil endif

DEFINITION:
lr-make-temp-name-alist (*temp-list*, *formals*)
= lr-make-temp-name-alist-1 (unpack (`'t*`), LIST-ASCII-0, *temp-list*, *formals*)

DEFINITION:
lr-new-cons (*car*, *cdr*)
= lr-new-node (tag (`'nat`, LR-CONS-TAG), tag (`'nat`, 1), *car*, *cdr*)

```
;; Deposit LIST of objects at ADDR, ADDR+1, ADDR+2, ... in DATA-SEG.
```

DEFINITION:
deposit-a-list (*list*, *addr*, *data-seg*)
= **if** listp (*list*)
  then deposit (car (*list*),
    addr,
    deposit-a-list (cdr (*list*), add1-addr (*addr*), *data-seg*))
  else *data-seg* **endif**

DEFINITION:
lr-init-heap-contents (*addr*, *size*)
= **if** *size* $\simeq$ 0 **then** list (tag (`'nat`, LR-INIT-TAG))
  else append (lr-new-node (tag (`'nat`, LR-INIT-TAG),
    add-addr (*addr*, LR-NODE-SIZE),
    tag (`'nat`, 0),

```
                    tag ('nat, 0)),
          lr-init-heap-contents (add-addr (addr, LR-NODE-SIZE),
                                           size - 1)) endif
```

DEFINITION:

```
lr-add-to-data-seg (data-seg, new-node)
=  if (length (value (LR-HEAP-NAME, data-seg)) - 1)
      ≮  (offset (fetch (LR-FP-ADDR, data-seg)) + length (new-node))
   then deposit (fetch (add-addr (fetch (LR-FP-ADDR, data-seg),
                                  LR-REF-COUNT-OFFSET),
                        data-seg),
                 LR-FP-ADDR,
                 deposit-a-list (new-node,
                                 fetch (LR-FP-ADDR, data-seg),
                                 data-seg))
   else data-seg endif
```

DEFINITION:

```
lr-init-data-seg (heap-size)
=  deposit-a-list (list (tag ('nat, LR-FALSE-TAG),
                         tag ('nat, 1),
                         LR-UNDEF-ADDR,
                         LR-UNDEF-ADDR),
                   LR-F-ADDR,
                   deposit-a-list (list (tag ('nat, LR-UNDEFINED-TAG),
                                         tag ('nat, 1),
                                         LR-UNDEF-ADDR,
                                         LR-UNDEF-ADDR),
                                   LR-UNDEF-ADDR,
                                   list (list (area-name (LR-FP-ADDR),
                                               add-addr (LR-F-ADDR,
                                                         LR-NODE-SIZE)),
                                         list (area-name (LR-ANSWER-ADDR),
                                               tag ('nat, 0)),
                                         cons (LR-HEAP-NAME,
                                               lr-init-heap-contents (tag ('addr,
                                                                           cons (LR-HEAP-NAME,
                                                                                 0)),
                                                                      heap-size)))))
```

DEFINITION:

```
count-list (flag, object)
=  if flag = 'list
   then if listp (object)
        then count-list (t, car (object))
```

```
                +    count-list ('list, cdr (object))
          else 1 endif
     elseif listp (object)
     then 1 + (1 + (count-list (t, car (object))
                      +    count-list (t, cdr (object))))
     elseif object ∈ N  then 1 + count (object)
     else 1 endif
```

THEOREM: not-equal-0-count-list
count-list (*flag*, *object*) $\neq$ 0

THEOREM: lessp-count-list-cdr-count-list-whole
listp (*object*)
$\rightarrow$ (count-list ('list, cdr (*object*)) $<$ count-list ('list, *object*))

THEOREM: lessp-count-not-list-car-count-list-whole
listp (*object*)
$\rightarrow$ (count-list (t, car (*object*)) $<$ count-list ('list, *object*))

`;; LR-COMPILE-QUOTE returns a pair, the new HEAP and the new TABLE.`

DEFINITION:
lr-compile-quote (*flag*, *object*, *heap*, *table*)

```
=    if flag = 'list
     then if listp (object)
          then let car-pair  be lr-compile-quote (t,
                                                  car (object),
                                                  heap,
                                                  table)
               in
               lr-compile-quote ('list,
                                 cdr (object),
                                 car (car-pair),
                                 cdr (car-pair)) endlet
          else cons (heap, table) endif
     elseif definedp (object, table)  then cons (heap, table)
     elseif listp (object)
     then let pair  be lr-compile-quote ('list,
                                         list (car (object), cdr (object)),
                                         heap,
                                         table)
          in
          cons (lr-add-to-data-seg (car (pair),
                                    lr-new-cons (cdr (assoc (car (object),
                                                             cdr (pair))),
```

```
                                          cdr (assoc (cdr (object),
                                                      cdr (pair))))),
                cons (cons (object, fetch (LR-FP-ADDR, car (pair))),
                      cdr (pair))) endlet
    elseif object ∈ N
    then cons (lr-add-to-data-seg (heap,
                                   lr-new-node (tag ('nat, LR-ADD1-TAG),
                                                tag ('nat, 1),
                                                tag ('nat, object),
                                                LR-UNDEF-ADDR)),
               cons (cons (object, fetch (LR-FP-ADDR, heap)), table))
    elseif truep (object)
    then cons (lr-add-to-data-seg (heap,
                                   lr-new-node (tag ('nat, LR-TRUE-TAG),
                                                tag ('nat, 1),
                                                LR-UNDEF-ADDR,
                                                LR-UNDEF-ADDR)),
               cons (cons (object, fetch (LR-FP-ADDR, heap)), table))
    else cons (heap, cons (cons (object, LR-UNDEF-ADDR), table)) endif

;; LR-DATA-SEG-TABLE-BODY returns a pair, the CAR is the extension of
;; DATA-SEG with any constants laid down in it, the CDR is an alist
;; mapping objects in the logic to addresses in the new DATA-SEG
;; where they are represented.  The initial TABLE is such an alist

DEFINITION:
lr-data-seg-table-body (flag, expr, data-seg, table)
=   if flag = 'list
    then if listp (expr)
         then let dst1 be lr-data-seg-table-body (t,
                                                  car (expr),
                                                  data-seg,
                                                  table)
              in
              lr-data-seg-table-body ('list,
                                      cdr (expr),
                                      car (dst1),
                                      cdr (dst1)) endlet
         else cons (data-seg, table) endif
    elseif listp (expr)
    then if (car (expr) = S-TEMP-FETCH)
            ∨ (car (expr) = S-TEMP-EVAL)
            ∨ (car (expr) = S-TEMP-TEST)
         then lr-data-seg-table-body (t, cadr (expr), data-seg, table)
```

```
        elseif car (expr) = 'quote
        then lr-compile-quote (t, cadr (expr), data-seg, table)
        else lr-data-seg-table-body ('list,
                                     cdr (expr),
                                     data-seg,
                                     table) endif
   else cons (data-seg, table) endif
```

DEFINITION:

```
lr-data-seg-table-list (progs, data-seg, table)
=  if listp (progs)
   then lr-data-seg-table-list (cdr (progs),
                                car (lr-data-seg-table-body (t,
                                                             s-body (car (progs)),
                                                             data-seg,
                                                             table)),
                                cdr (lr-data-seg-table-body (t,
                                                             s-body (car (progs)),
                                                             data-seg,
                                                             table)))
   else cons (data-seg, table) endif
```

DEFINITION:

```
lr-init-data-seg-table (params, data-seg, table)
=  if listp (params)
   then let ds-tab be lr-compile-quote (t, cdar (params), data-seg, table)
        in
        lr-init-data-seg-table (cdr (params),
                                car (ds-tab),
                                cdr (ds-tab)) endlet
   else cons (data-seg, table) endif
```

DEFINITION:

```
lr-data-seg-table (progs, params, heap-size)
=  let init-ds-table1 be lr-compile-quote ('list,
                                           list (t, 0),
                                           lr-init-data-seg (heap-size),
                                           list (cons (f, LR-F-ADDR)))
   in
   let init-ds-table2 be lr-init-data-seg-table (params,
                                                 car (init-ds-table1),
                                                 cdr (init-ds-table1))
   in
   lr-data-seg-table-list (progs,
                           car (init-ds-table2),
```

```
                    cdr (init-ds-table2)) endlet endlet

DEFINITION:
pair-formals-with-addresses (formals, table)
=  if listp (formals)
   then cons (cons (caar (formals), cdr (assoc (cdar (formals), table))),
              pair-formals-with-addresses (cdr (formals), table))
   else nil endif

DEFINITION:
lr-make-initial-temps (temp-vars)
=  if listp (temp-vars)
   then cons (cons (car (temp-vars), LR-UNDEF-ADDR),
              lr-make-initial-temps (cdr (temp-vars)))
   else nil endif

DEFINITION:
lr-initial-cstk (params, temp-alist, table, pc)
=  list (p-frame (append (pair-formals-with-addresses (params, table),
                          lr-make-initial-temps (strip-cdrs (temp-alist))),
                 pc))

DEFINITION:
lr-compile-body (flag, body, temp-alist, const-table)
=  if flag = 'list
   then if listp (body)
        then cons (lr-compile-body (t, car (body), temp-alist, const-table),
                   lr-compile-body ('list,
                                    cdr (body),
                                    temp-alist,
                                    const-table))
        else nil endif
   elseif listp (body)
   then if (car (body) = S-TEMP-FETCH)
           ∨ (car (body) = S-TEMP-EVAL)
           ∨ (car (body) = S-TEMP-TEST)
        then list (car (body),
                   lr-compile-body (t, cadr (body), temp-alist, const-table),
                   value (cadr (body), temp-alist))
        elseif car (body) = 'quote
        then list ('quote, value (cadr (body), const-table))
        else cons (car (body),
                   lr-compile-body ('list,
                                    cdr (body),
                                    temp-alist,
```

```
                                        const-table)) endif
    else body endif
```

DEFINITION:

```
lr-make-temp-var-dcls (temp-alist)
=   if listp (temp-alist)
    then cons (list (cdar (temp-alist), LR-UNDEF-ADDR),
               lr-make-temp-var-dcls (cdr (temp-alist)))
    else nil endif
```

DEFINITION:

```
lr-compile-programs (programs, const-table)
=   if listp (programs)
    then let prog be car (programs)
         in
         let temp-alist be lr-make-temp-name-alist (s-temp-list (prog),
                                                    s-formals (prog))
         in
         cons (lr-make-program (car (prog),
                                s-formals (prog),
                                lr-make-temp-var-dcls (temp-alist),
                                lr-compile-body (t,
                                                 s-body (prog),
                                                 temp-alist,
                                                 const-table)),
               lr-compile-programs (cdr (programs), const-table)) endlet endlet
    else nil endif
```

DEFINITION:

```
lr-p-c-size (flag, expr)
=   if flag = 'list
    then if listp (expr)
         then lr-p-c-size (t, car (expr))
              +  lr-p-c-size ('list, cdr (expr))
         else 0 endif
    elseif listp (expr)
    then if car (expr) = 'if
         then lr-p-c-size (t, cadr (expr))
              +  lr-p-c-size (t, caddr (expr))
              +  lr-p-c-size (t, cadddr (expr))
              +  4
         elseif car (expr) = S-TEMP-FETCH then 1
         elseif car (expr) = S-TEMP-EVAL
         then lr-p-c-size (t, cadr (expr)) + 1
         elseif car (expr) = S-TEMP-TEST
```

```
          then lr-p-c-size (t, cadr (expr)) + 7
          elseif car (expr) = 'quote then 1
          else lr-p-c-size ('list, cdr (expr)) + 1 endif
     else 1 endif
```

DEFINITION:

```
lr-p-c-size-list (n, expr-list)
=    if n ≃ 0 then 0
     elseif n < length (expr-list)
     then lr-p-c-size (t, get (n, expr-list))
          +   lr-p-c-size-list (n − 1, expr-list)
     else lr-p-c-size-list (length (expr-list) − 1, expr-list) endif
```

```
;; LR-P-PC-1 returns the number of Piton instructions before the start of
;; the expression denoted by POS in the compilation of EXPR.
```

DEFINITION:

```
lr-p-pc-1 (expr, pos)
=    if ¬ listp (pos) then 0
     elseif ¬ listp (expr) then 0
     elseif car (pos) ≃ 0 then 0
     elseif car (expr) = 'if
     then if car (pos) ≃ 0 then 0
          elseif car (pos) = 1 then lr-p-pc-1 (cadr (expr), cdr (pos))
          elseif car (pos) = 2
          then 3
                +   lr-p-c-size (t, cadr (expr))
                +   lr-p-pc-1 (caddr (expr), cdr (pos))
          else lr-p-c-size (t, cadr (expr))
               +   lr-p-c-size (t, caddr (expr))
               +   lr-p-pc-1 (cadddr (expr), cdr (pos))
               +   4 endif
     elseif car (expr) = S-TEMP-FETCH then 0
     elseif car (expr) = S-TEMP-EVAL then lr-p-pc-1 (cadr (expr), cdr (pos))
     elseif car (expr) = S-TEMP-TEST
     then lr-p-pc-1 (cadr (expr), cdr (pos)) + 4
     elseif car (expr) = 'quote then 0
     else lr-p-c-size-list (car (pos) − 1, expr)
          +   lr-p-pc-1 (get (car (pos), expr), cdr (pos)) endif
```

DEFINITION:

```
lr-p-pc (l)
=    tag ('pc,
          cons (area-name (p-pc (l)),
                lr-p-pc-1 (program-body (p-current-program (l)), offset (p-pc (l)))))
```

EVENT: Disable lr-p-pc.

DEFINITION:
s->lr1 (*s*, *l*, *table*)
= p-state (tag (`'pc`, cons (s-pname (*s*), s-pos (*s*))),
p-ctrl-stk (*l*),
p-temp-stk (*l*),
lr-compile-programs (s-progs (*s*), *table*),
p-data-segment (*l*),
p-max-ctrl-stk-size (*l*),
p-max-temp-stk-size (*l*),
p-word-size (*l*),
s-err-flag (*s*))

EVENT: Disable s->lr1.

```
;; Returns an P-STATE.
;; FREE-HEAP-SIZE is number of free nodes in resulting P-STATE.
```

DEFINITION:
s->lr (*s*, *fheap-size*, *max-ctrl*, *max-temp*, *word-size*)
= **let** *temp-alist* **be** lr-make-temp-name-alist (strip-cars (s-temps (*s*)),
strip-cars (s-params (*s*))),
dataseg-table **be** lr-data-seg-table (s-progs (*s*),
s-params (*s*),
fheap-size)
in
let *return-pc* **be** tag (`'pc`,
cons (s-pname (*s*),
lr-p-pc-1 (lr-compile-body (**t**,
s-body (s-prog (*s*)),
temp-alist,
cdr (*dataseg-table*)),
s-pos (*s*))))
in
s->lr1 (*s*,
p-state (**nil**,
lr-initial-cstk (s-params (*s*),
temp-alist,
cdr (*dataseg-table*),
return-pc),
nil,
nil,

car (*dataseg-table*),
max-ctrl,
max-temp,
word-size,
nil),
cdr (*dataseg-table*)) **endlet endlet**

EVENT: Disable s->lr.

DEFINITION:
lr-params (*frame*, *p*)
= firstn (length (formal-vars (p-current-program (*p*))), bindings (*frame*))

EVENT: Disable lr-params.

DEFINITION:
lr-temps (*frame*, *p*)
= restn (length (formal-vars (p-current-program (*p*))), bindings (*frame*))

EVENT: Disable lr-temps.

DEFINITION:
lr-set-expr (*s1*, *s2*, *pos*)
= p-state (tag (`'pc`, cons (area-name (p-pc (*s2*)), *pos*)),
p-ctrl-stk (*s1*),
p-temp-stk (*s1*),
p-prog-segment (*s2*),
p-data-segment (*s1*),
p-max-ctrl-stk-size (*s1*),
p-max-temp-stk-size (*s1*),
p-word-size (*s1*),
p-psw (*s1*))

DEFINITION:
lr-set-error (*s*, *flag*)
= p-state (p-pc (*s*),
p-ctrl-stk (*s*),
p-temp-stk (*s*),
p-prog-segment (*s*),
p-data-segment (*s*),
p-max-ctrl-stk-size (*s*),
p-max-temp-stk-size (*s*),
p-word-size (*s*),
flag)

DEFINITION:
lr-set-pos (s, *pos*)
= p-state (tag (`'pc`, cons (area-name (p-pc (s)), *pos*)),
    p-ctrl-stk (s),
    p-temp-stk (s),
    p-prog-segment (s),
    p-data-segment (s),
    p-max-ctrl-stk-size (s),
    p-max-temp-stk-size (s),
    p-word-size (s),
    p-psw (s))

DEFINITION:
lr-set-tstk (s, *temp-stk*)
= p-state (p-pc (s),
    p-ctrl-stk (s),
    temp-stk,
    p-prog-segment (s),
    p-data-segment (s),
    p-max-ctrl-stk-size (s),
    p-max-temp-stk-size (s),
    p-word-size (s),
    p-psw (s))

DEFINITION:
lr-pop-tstk (s)
= **if** p-psw (s) = `'run`
  then if listp (p-temp-stk (s)) **then** lr-set-tstk (s, pop (p-temp-stk (s)))
    else lr-set-error (s, `'lr-pop-tstk-empty-stack`) **endif**
  else s **endif**

DEFINITION:
lr-push-tstk (s, *value*)
= **if** p-psw (s) = `'run`
  then if length (p-temp-stk (s)) < p-max-temp-stk-size (s)
    then lr-set-tstk (s, push (*value*, p-temp-stk (s)))
    else lr-set-error (s, `'lr-push-tstk-full-stack`) **endif**
  else s **endif**

EVENT: Disable lr-push-tstk.

DEFINITION:
lr-if-ok (l)
= **if** p-max-temp-stk-size (l) $\not<$ (`1` + length (p-temp-stk (l))) **then** l
  else lr-set-error (l, `'if-temp-stk-overflow`) **endif**

EVENT: Disable lr-if-ok.

DEFINITION:
lr-set-temp (s, *value*, *var-name*)
= **if** p-psw (s) = `'run`
  then p-state (p-pc (s),
    set-local-var-value (*value*, *var-name*, p-ctrl-stk (s)),
    p-temp-stk (s),
    p-prog-segment (s),
    p-data-segment (s),
    p-max-ctrl-stk-size (s),
    p-max-temp-stk-size (s),
    p-word-size (s),
    p-psw (s))
  else s **endif**

EVENT: Disable lr-set-temp.

DEFINITION:
lr-eval-temp-setp (s)
= (local-var-value (caddr (lr-expr (s)), p-ctrl-stk (s)) $\neq$ LR-UNDEF-ADDR)

EVENT: Disable lr-eval-temp-setp.

DEFINITION:
lr-do-temp-fetch (s)
= **if** lr-eval-temp-setp (s)
  then lr-push-tstk (s, local-var-value (caddr (lr-expr (s)), p-ctrl-stk (s)))
  else lr-set-error (s, `'temp-fetch-not-set`) **endif**

EVENT: Disable lr-do-temp-fetch.

DEFINITION:
lr-pop-cstk (s)
= **if** p-psw (s) = `'run`
  then p-state (p-pc (s),
    pop (p-ctrl-stk (s)),
    p-temp-stk (s),
    p-prog-segment (s),
    p-data-segment (s),
    p-max-ctrl-stk-size (s),
    p-max-temp-stk-size (s),

p-word-size (s),
p-psw (s))
else s **endif**

EVENT: Disable lr-pop-cstk.

DEFINITION:
lr-type-contents-p (*object*, *tag*, *contents*)
$=$ ((type (*object*) $=$ *tag*) $\land$ (untag (*object*) $=$ *contents*))

```
;; The following functions are used for the Piton code and to compute the LR
;; value for certain classes of functions (e.g. all shell accessors).

;; NOTE: The 'clock' functions get a Piton state.  This is the state just
;; BEFORE the execution of the appropriate CALL instruction.  Therefore
;; to look at the parameters, it is necessary to look at the temp stack.
;; The clock function return the number of Piton instructions necessary to
;; run the CALL and the code for the SUBR.

;; Recognizers
```

DEFINITION:
p-recognizer-code (*name*, *tag*)
$=$ list (*name*,
'`nil`,
'`nil`,
'`(fetch)`,
list ('`push-constant`, tag ('`nat`, *tag*)),
'`(eq)`,
'`(test-bool-and-jump f false)`,
list ('`push-constant`, LR-T-ADDR),
'`(ret)`,
list ('`dl`, '`false`, '`nil`, list ('`push-constant`, LR-F-ADDR)),
'`(ret)`)

DEFINITION: p-recognizer-clock (*p-state*, *tag*) $=$ `7`

```
;; Accessor
```

DEFINITION:
p-accessor-code (*name*, *tag*, *default*, *offset*)
$=$ list (*name*,
'`(x)`,
'`nil`,
'`(push-local x)`,

'(fetch),
list ('push-constant, tag ('nat, *tag*)),
'(eq),
'(test-bool-and-jump t arg1),
list ('push-constant, *default*),
'(ret),
'(dl arg1 nil (push-local x)),
list ('push-constant, tag ('nat, *offset*)),
'(add-addr),
'(fetch),
'(ret))

DEFINITION:
p-accessor-clock (p, *tag*)
= **if** fetch (top (p-temp-stk (p)), p-data-segment (p)) = tag ('nat, *tag*)
then 11
else 8 **endif**

```
;; Now comes the actual code and values
```

DEFINITION:
P-CAR-CODE = p-accessor-code ('car, LR-CONS-TAG, LR-0-ADDR, LR-CAR-OFFSET)

DEFINITION: p-car-clock (p) = p-accessor-clock (p, LR-CONS-TAG)

EVENT: Disable p-car-clock.

DEFINITION:
P-CDR-CODE = p-accessor-code ('cdr, LR-CONS-TAG, LR-0-ADDR, LR-CDR-OFFSET)

DEFINITION: p-cdr-clock (p) = p-accessor-clock (p, LR-CONS-TAG)

EVENT: Disable p-cdr-clock.

DEFINITION:
P-CONS-CODE
= list ('cons,
'nil,
'((temp (nat 0))),
'(push-global free-ptr),
list ('push-constant, tag ('nat, LR-CDR-OFFSET)),
'(add-addr),
'(deposit),

'(push-global free-ptr),
list ('push-constant, tag ('nat, LR-CAR-OFFSET)),
'(add-addr),
'(deposit),
'(push-global free-ptr),
'(push-global free-ptr),
list ('push-constant, tag ('nat, LR-REF-COUNT-OFFSET)),
'(add-addr),
'(set-local temp),
'(fetch),
'(push-constant (nat 1)),
'(push-local temp),
'(deposit),
list ('push-constant, tag ('nat, LR-CONS-TAG)),
'(push-global free-ptr),
'(deposit),
'(pop-global free-ptr),
'(ret))

DEFINITION: p-cons-clock (p) = 23

EVENT: Disable p-cons-clock.

DEFINITION:
P-FALSE-CODE
= list ('false,
'nil,
'nil,
list ('push-constant, LR-F-ADDR),
'(ret))

DEFINITION: p-false-clock (p) = 3

EVENT: Disable p-false-clock.

```
;; FALSEP TAKES ONE IMPLICIT ARG ON STACK.
```

DEFINITION:
P-FALSEP-CODE
= list ('falsep,
'nil,
'nil,
list ('push-constant, LR-F-ADDR),
'(eq),

'(test-bool-and-jump t true),
list ('push-constant, LR-F-ADDR),
'(ret),
list ('dl, 'true, 'nil, list ('push-constant, LR-T-ADDR)),
'(ret))

DEFINITION: p-falsep-clock (p) = 6

EVENT: Disable p-falsep-clock.

```
;; Takes an implicit arg
```

DEFINITION:
P-LISTP-CODE = p-recognizer-code ('listp, LR-CONS-TAG)

DEFINITION:
p-listp-clock (p) = p-recognizer-clock (p, LR-CONS-TAG)

EVENT: Disable p-listp-clock.

DEFINITION:
P-NLISTP-CODE
= list ('nlistp,
'nil,
'nil,
'(fetch),
list ('push-constant, tag ('nat, LR-CONS-TAG)),
'(eq),
'(test-bool-and-jump f true),
list ('push-constant, LR-F-ADDR),
'(ret),
list ('dl, 'true, 'nil, list ('push-constant, LR-T-ADDR)),
'(ret))

DEFINITION: p-nlistp-clock (p) = 7

EVENT: Disable p-nlistp-clock.

DEFINITION:
P-TRUE-CODE
= list ('true, 'nil, 'nil, list ('push-constant, LR-T-ADDR), '(ret))

DEFINITION: p-true-clock (p) = 3

EVENT: Disable p-true-clock.

```
;; The old code for TRUEP is shown below.  I used to ensure that there was
;; only one occurence of TRUE in the data-segment [namely at address
;; (lr-t-addr)], however only TRUEP took advantage of this.  LR-PROPER-HEAPP
;; has been changed to not require only one occurrence, although only one
;; should appear.  However this means we actually have to test the tag, a
;; small performance penalty for some simplicity and freedom in the spec.
```

DEFINITION:
P-TRUEP-CODE = p-recognizer-code ('truep, LR-TRUE-TAG)

```
;(defn P-TRUEP-CODE ()
;  (list 'truep '() '()
; (list 'PUSH-CONSTANT (lr-t-addr))
; '(EQ)
; '(TEST-BOOL-AND-JUMP T TRUE)
; (list 'PUSH-CONSTANT (lr-f-addr))
; '(RET)
; (list 'DL 'TRUE '() (list 'PUSH-CONSTANT (lr-t-addr)))
; '(RET)))
```

DEFINITION:
p-truep-clock (p) = p-recognizer-clock (p, LR-FALSE-TAG)

```
;(defn P-TRUEP-CLOCK (p) 6)
```

EVENT: Disable p-truep-clock.

DEFINITION:
P-RUNTIME-SUPPORT-PROGRAMS
= list (P-CAR-CODE,
P-CDR-CODE,
P-CONS-CODE,
P-FALSE-CODE,
P-FALSEP-CODE,
P-LISTP-CODE,
P-NLISTP-CODE,
P-TRUE-CODE,
P-TRUEP-CODE)

EVENT: Disable p-runtime-support-programs.

DEFINITION:
lr-convert-digit-to-ascii (*digit*) = (ASCII-0 + *digit*)

DEFINITION:
lr-convert-num-to-ascii (*number*, *list*)
= **if** *number* < `10` **then** cons (lr-convert-digit-to-ascii (*number*), *list*)
else lr-convert-num-to-ascii (*number* ÷ `10`,
cons (lr-convert-digit-to-ascii (*number* **mod** `10`),
list)) **endif**

DEFINITION:
lr-make-label (*n*)
= pack (cons (car (unpack (`'l`)),
cons (ASCII-DASH, append (lr-convert-num-to-ascii (*n*, **nil**), `0`))))

EVENT: Disable lr-make-label.

DEFINITION:
label-instrs (*instrs*, *n*)
= **if** listp (*instrs*)
then cons (dl (lr-make-label (*n*), **nil**, car (*instrs*)),
label-instrs (cdr (*instrs*), 1 + *n*))
else nil endif

DEFINITION:
comp-temp-test (*expr*, *instrs*, *n*)
= append (list (list (`'push-local`, caddr (*expr*)),
list (`'push-constant`, LR-UNDEF-ADDR),
`'(eq)`,
list (`'test-bool-and-jump`,
`'f`,
lr-make-label (*n* + `6` + length (*instrs*)))),
append (*instrs*,
list (list (`'set-local`, caddr (*expr*)),
list (`'jump`,
lr-make-label (*n* + `7` + length (*instrs*))),
list (`'push-local`, caddr (*expr*)))))

DEFINITION:
comp-if (*test-instrs*, *then-instrs*, *else-instrs*, *n*)
= append (*test-instrs*,
append (list (list (`'push-constant`, LR-F-ADDR),
`'(eq)`,
list (`'test-bool-and-jump`,

```
                    't,
                    lr-make-label (n
                                   +  4
                                   +  length (test-instrs)
                                   +  length (then-instrs)))),
               append (then-instrs,
                    cons (list ('jump,
                                lr-make-label (n
                                               +  4
                                               +  length (test-instrs)
                                               +  length (then-instrs)
                                               +  length (else-instrs))),
                          else-instrs))))
```

```
;; COMP-BODY-1 returns a list of Piton instructions to compile EXPR.
;; N is the number of Piton instructions previously generated, it is used
;; to generate unique labels.
```

DEFINITION:

```
comp-body-1 (flag, expr, n)
=   if flag = 'list
    then if listp (expr)
         then append (comp-body-1 (t, car (expr), n),
                      comp-body-1 ('list,
                                   cdr (expr),
                                   n + lr-p-c-size (t, car (expr))))
         else nil endif
    elseif listp (expr)
    then if car (expr) = 'if
         then comp-if (comp-body-1 (t, cadr (expr), n),
                       comp-body-1 (t,
                                    caddr (expr),
                                    n + 3 + lr-p-c-size (t, cadr (expr))),
                       comp-body-1 (t,
                                    cadddr (expr),
                                    n
                                    +  4
                                    +  lr-p-c-size (t, cadr (expr))
                                    +  lr-p-c-size (t, caddr (expr))),
                       n)
         elseif car (expr) = S-TEMP-FETCH
         then list (list ('push-local, caddr (expr)))
         elseif car (expr) = S-TEMP-EVAL
         then append (comp-body-1 (t, cadr (expr), n),
```

```
                        list (list ('set-local, caddr (expr))))
            elseif car (expr) = S-TEMP-TEST
            then comp-temp-test (expr, comp-body-1 (t, cadr (expr), n + 4), n)
            elseif car (expr) = 'quote
            then list (list ('push-constant, cadr (expr)))
            else append (comp-body-1 ('list, cdr (expr), n),
                         if definedp (car (expr),
                                      P-RUNTIME-SUPPORT-PROGRAMS)
                         then list (list ('call, car (expr)))
                         else list (list ('call,
                                          user-fname (car (expr)))) endif) endif
     else list (list ('push-local, expr)) endif
```

EVENT: Disable comp-body-1.

DEFINITION:

```
comp-body (body)
=    label-instrs (append (comp-body-1 (t, body, 0), '((ret))), 0)
```

EVENT: Disable comp-body.

DEFINITION:

```
comp-programs-1 (programs)
=    if listp (programs)
     then cons (lr-make-program (name (car (programs)),
                                 formal-vars (car (programs)),
                                 temp-var-dcls (car (programs)),
                                 comp-body (program-body (car (programs)))),
                comp-programs-1 (cdr (programs)))
     else nil endif
```

DEFINITION:

```
comp-programs (programs)
=    cons (lr-make-program (name (car (programs)),
                            formal-vars (car (programs)),
                            temp-var-dcls (car (programs)),
                            label-instrs (append (comp-body-1 (t,
                                                               program-body (car (programs)),
                                                               0),
                                                  list (list ('set-global,
                                                              area-name (LR-ANSWER-ADDR)),
                                                        '(ret))),
                                          0)),
           append (comp-programs-1 (cdr (programs)), P-RUNTIME-SUPPORT-PROGRAMS))
```

EVENT: Disable comp-programs.

DEFINITION:

lr-proper-exprp (*flag*, *expr*, *pnames*, *formals*, *temps*, *table*)

```
=  if flag = 'list
   then if listp (expr)
        then lr-proper-exprp (t, car (expr), pnames, formals, temps, table)
             ∧  lr-proper-exprp ('list,
                                 cdr (expr),
                                 pnames,
                                 formals,
                                 temps,
                                 table)
        else expr = nil endif
   elseif litatom (expr) then expr ∈ formals
   elseif expr ≃ nil then f
   elseif ¬ plistp (expr) then f
   elseif car (expr) = S-TEMP-FETCH
   then (caddr (expr) ∈ temps) ∧ (length (expr) = 3)
   elseif (car (expr) = S-TEMP-EVAL) ∨ (car (expr) = S-TEMP-TEST)
   then (caddr (expr) ∈ temps)
        ∧  (length (expr) = 3)
        ∧  lr-proper-exprp (t, cadr (expr), pnames, formals, temps, table)
   elseif car (expr) = 'quote
   then (type (cadr (expr)) = 'addr)
        ∧  (cadr (expr) ∈ strip-cdrs (table))
        ∧  (length (cdr (expr)) = arity (car (expr)))
   elseif subrp (car (expr))
   then (length (cdr (expr)) = arity (car (expr)))
        ∧  ((car (expr) = 'if)
            ∨  definedp (car (expr), P-RUNTIME-SUPPORT-PROGRAMS))
        ∧  (car (expr) ∉ pnames)
        ∧  lr-proper-exprp ('list,
                            cdr (expr),
                            pnames,
                            formals,
                            temps,
                            table)
   elseif body (car (expr))
   then (length (cdr (expr)) = arity (car (expr)))
        ∧  (car (expr) ∈ pnames)
        ∧  lr-proper-exprp ('list,
                            cdr (expr),
```

$pnames$,
$formals$,
$temps$,
$table$)

else f endif

DEFINITION:
all-undef-addrs ($list$)
= **if** listp ($list$)
then (car ($list$) = LR-UNDEF-ADDR) $\land$ all-undef-addrs (cdr ($list$))
else t endif

DEFINITION:
lr-programs-properp-1 ($programs$, $program\text{-}names$, $table$)
= **if** listp ($programs$)
then all-litatoms (formal-vars (car ($programs$)))
$\land$ all-litatoms (strip-cars (temp-var-dcls (car ($programs$))))
$\land$ all-undef-addrs (strip-cadrs (temp-var-dcls (car ($programs$))))
$\land$ lr-proper-exprp (**t**,
program-body (car ($programs$)),
$program\text{-}names$,
formal-vars (car ($programs$)),
strip-cars (temp-var-dcls (car ($programs$))),
$table$)
$\land$ lr-programs-properp-1 (cdr ($programs$), $program\text{-}names$, $table$)
else t endif

EVENT: Disable lr-programs-properp-1.

DEFINITION:
lr-programs-properp (l, $table$)
= (definedp (area-name (p-pc (l)), p-prog-segment (l))
$\land$ (caar (p-prog-segment (l)) = `'main`)
$\land$ all-user-fnamesp (cdr (strip-cars (p-prog-segment (l))))
$\land$ lr-programs-properp-1 (p-prog-segment (l),
strip-logic-fnames (cdr (p-prog-segment (l))),
$table$))

EVENT: Disable lr-programs-properp.

THEOREM: lr-p-c-size-flag-not-list-not-0
($flag \neq$ `'list`) $\rightarrow$ (lr-p-c-size ($flag$, $expr$) $\neq 0$)

THEOREM: difference-decreases
$((x \not< y) \land (y \not\simeq 0)) \rightarrow (((x - y) < x) = \mathbf{t})$

DEFINITION:
lr->p (p)
= p-state (lr-p-pc (p),
p-ctrl-stk (p),
p-temp-stk (p),
comp-programs (p-prog-segment (p)),
p-data-segment (p),
p-max-ctrl-stk-size (p),
p-max-temp-stk-size (p),
p-word-size (p),
p-psw (p))

EVENT: Disable lr->p.

DEFINITION:
p-set-pc (p, pc)
= p-state (pc,
p-ctrl-stk (p),
p-temp-stk (p),
p-prog-segment (p),
p-data-segment (p),
p-max-ctrl-stk-size (p),
p-max-temp-stk-size (p),
p-word-size (p),
p-psw (p))

```
;; It should be the case that (P-CURRENT-INSTRUCTION p) = (CALL subr)
;; therefore we need to run P one more step than the clock functions
;; below to do the CALL.
```

DEFINITION:
p-run-subr ($subr$, p)
= **case on** $subr$:
case = car
then p (p, p-car-clock (p))
case = cdr
then p (p, p-cdr-clock (p))
case = $cons$
then p (p, p-cons-clock (p))
case = $false$
then p (p, p-false-clock (p))
case = $falsep$
then p (p, p-falsep-clock (p))
case = $listp$

then p (p, p-listp-clock (p))
case = *nlistp*
then p (p, p-nlistp-clock (p))
case = *true*
then p (p, p-true-clock (p))
case = *truep*
then p (p, p-truep-clock (p))
otherwise p-halt (p, `'bad-subr`) **endcase**

EVENT: Disable p-run-subr.

DEFINITION:
lr-return-pc (l)
= add-addr (lr-p-pc (l), lr-p-c-size (`'list`, cdr (lr-expr (l))))

EVENT: Disable lr-return-pc.

DEFINITION:
lr-apply-subr (l, *new-l*)
= **let** *res* **be** p-run-subr (car (lr-expr (l)),
p-set-pc (lr->p (*new-l*), lr-return-pc (l)))
in
p-state (p-pc (*new-l*),
p-ctrl-stk (*res*),
p-temp-stk (*res*),
p-prog-segment (*new-l*),
p-data-segment (*res*),
p-max-ctrl-stk-size (*res*),
p-max-temp-stk-size (*res*),
p-word-size (*res*),
p-psw (*res*)) **endlet**

EVENT: Disable lr-apply-subr.

DEFINITION:
lr-funcall (l, *new-l*)
= **let** *prog* **be** definition (user-fname (car (lr-expr (l))),
p-prog-segment (l)),
newest-l **be** p-set-pc (lr->p (*new-l*), lr-return-pc (l))
in
if p-call-okp (list (`'call`, user-fname (car (lr-expr (l)))),
newest-l)
then p-state (tag (`'pc`, cons (user-fname (car (lr-expr (l))), **nil**)),

push (make-p-call-frame (formal-vars (*prog*),
p-temp-stk (*new-l*),
temp-var-dcls (*prog*),
add1-addr (p-pc (*newest-l*))),
p-ctrl-stk (*new-l*)),
popn (length (formal-vars (*prog*)), p-temp-stk (*new-l*)),
p-prog-segment (*new-l*),
p-data-segment (*new-l*),
p-max-ctrl-stk-size (*new-l*),
p-max-temp-stk-size (*new-l*),
p-word-size (*new-l*),
`'run`)
else p-halt (*new-l*, x-y-error-msg (`'p`, `'call`)) **endif endlet**

EVENT: Disable lr-funcall.

```
;; The following lemmas are needed to admit LR-EVAL
```

THEOREM: p-accessors-lr-set-expr
(p-pc (lr-set-expr (*s1*, *s2*, *pos*)) = tag (`'pc`, cons (area-name (p-pc (*s2*)), *pos*)))
$\wedge$ (p-ctrl-stk (lr-set-expr (*s1*, *s2*, *pos*)) = p-ctrl-stk (*s1*))
$\wedge$ (p-temp-stk (lr-set-expr (*s1*, *s2*, *pos*)) = p-temp-stk (*s1*))
$\wedge$ (p-prog-segment (lr-set-expr (*s1*, *s2*, *pos*)) = p-prog-segment (*s2*))
$\wedge$ (p-data-segment (lr-set-expr (*s1*, *s2*, *pos*)) = p-data-segment (*s1*))
$\wedge$ (p-max-ctrl-stk-size (lr-set-expr (*s1*, *s2*, *pos*))
= p-max-ctrl-stk-size (*s1*))
$\wedge$ (p-max-temp-stk-size (lr-set-expr (*s1*, *s2*, *pos*))
= p-max-temp-stk-size (*s1*))
$\wedge$ (p-word-size (lr-set-expr (*s1*, *s2*, *pos*)) = p-word-size (*s1*))
$\wedge$ (p-psw (lr-set-expr (*s1*, *s2*, *pos*)) = p-psw (*s1*))

EVENT: Disable lr-set-expr.

THEOREM: p-accessors-lr-set-tstk
(p-pc (lr-set-tstk (*s*, *ts*)) = p-pc (*s*))
$\wedge$ (p-ctrl-stk (lr-set-tstk (*s*, *ts*)) = p-ctrl-stk (*s*))
$\wedge$ (p-temp-stk (lr-set-tstk (*s*, *ts*)) = *ts*)
$\wedge$ (p-prog-segment (lr-set-tstk (*s*, *ts*)) = p-prog-segment (*s*))
$\wedge$ (p-data-segment (lr-set-tstk (*s*, *ts*)) = p-data-segment (*s*))
$\wedge$ (p-max-ctrl-stk-size (lr-set-tstk (*s*, *ts*)) = p-max-ctrl-stk-size (*s*))
$\wedge$ (p-max-temp-stk-size (lr-set-tstk (*s*, *ts*)) = p-max-temp-stk-size (*s*))
$\wedge$ (p-word-size (lr-set-tstk (*s*, *ts*)) = p-word-size (*s*))
$\wedge$ (p-psw (lr-set-tstk (*s*, *ts*)) = p-psw (*s*))

Event: Disable lr-set-tstk.

Theorem: p-accessors-lr-set-error
(p-pc (lr-set-error (s, *flag*)) = p-pc (s))
$\land$ (p-ctrl-stk (lr-set-error (s, *flag*)) = p-ctrl-stk (s))
$\land$ (p-temp-stk (lr-set-error (s, *flag*)) = p-temp-stk (s))
$\land$ (p-prog-segment (lr-set-error (s, *flag*)) = p-prog-segment (s))
$\land$ (p-data-segment (lr-set-error (s, *flag*)) = p-data-segment (s))
$\land$ (p-max-ctrl-stk-size (lr-set-error (s, *flag*)) = p-max-ctrl-stk-size (s))
$\land$ (p-max-temp-stk-size (lr-set-error (s, *flag*)) = p-max-temp-stk-size (s))
$\land$ (p-word-size (lr-set-error (s, *flag*)) = p-word-size (s))
$\land$ (p-psw (lr-set-error (s, *flag*)) = *flag*)

Event: Disable lr-set-error.

Theorem: p-accessors-lr-set-pos
(p-pc (lr-set-pos (s, *pos*)) = tag (`'pc`, cons (area-name (p-pc (s)), *pos*)))
$\land$ (p-ctrl-stk (lr-set-pos (s, *pos*)) = p-ctrl-stk (s))
$\land$ (p-temp-stk (lr-set-pos (s, *pos*)) = p-temp-stk (s))
$\land$ (p-prog-segment (lr-set-pos (s, *pos*)) = p-prog-segment (s))
$\land$ (p-data-segment (lr-set-pos (s, *pos*)) = p-data-segment (s))
$\land$ (p-max-ctrl-stk-size (lr-set-pos (s, *pos*)) = p-max-ctrl-stk-size (s))
$\land$ (p-max-temp-stk-size (lr-set-pos (s, *pos*)) = p-max-temp-stk-size (s))
$\land$ (p-word-size (lr-set-pos (s, *pos*)) = p-word-size (s))
$\land$ (p-psw (lr-set-pos (s, *pos*)) = p-psw (s))

Event: Disable lr-set-pos.

Theorem: p-accessors-lr-pop-tstk
(p-pc (lr-pop-tstk (s)) = p-pc (s))
$\land$ (p-ctrl-stk (lr-pop-tstk (s)) = p-ctrl-stk (s))
$\land$ (p-prog-segment (lr-pop-tstk (s)) = p-prog-segment (s))
$\land$ (p-data-segment (lr-pop-tstk (s)) = p-data-segment (s))
$\land$ (p-max-ctrl-stk-size (lr-pop-tstk (s)) = p-max-ctrl-stk-size (s))
$\land$ (p-max-temp-stk-size (lr-pop-tstk (s)) = p-max-temp-stk-size (s))
$\land$ (p-word-size (lr-pop-tstk (s)) = p-word-size (s))

Theorem: p-temp-stk-lr-pop-tstk
p-temp-stk (lr-pop-tstk (s))
= **if** listp (p-temp-stk (s)) $\land$ (p-psw (s) = `'run`)
then pop (p-temp-stk (s))
else p-temp-stk (s) **endif**

Event: Disable lr-pop-tstk.

THEOREM: area-name-tag
area-name (tag (tag, adp)) $=$ adp-name (adp)

THEOREM: offset-tag
offset (tag (tag, adp)) $=$ adp-offset (adp)

THEOREM: p-current-program-lr-set-expr
p-current-program (lr-set-expr ($s1$, $s2$, pos)) $=$ p-current-program ($s2$)

THEOREM: p-current-program-lr-set-pos
p-current-program (lr-set-pos (s, pos)) $=$ p-current-program (s)

THEOREM: lr-expr-lr-set-expr
lr-expr (lr-set-expr ($s1$, $s2$, dv (offset (p-pc ($s2$)), n))) $=$ get (n, lr-expr ($s2$))

THEOREM: lr-expr-lr-set-pos-t
lr-expr (lr-set-pos (s, dv (offset (p-pc (s)), n))) $=$ get (n, lr-expr (s))

THEOREM: lr-expr-flag-list-car
listp (offset (p-pc (p))) $\rightarrow$ (car (lr-expr-list (p)) $=$ lr-expr (p))

THEOREM: number-cons-lr-expr-t-list
(listp (lr-expr-list (p)) $\land$ listp (offset (p-pc (p))))
$\rightarrow$ (number-cons (lr-expr (p)) $<$ number-cons (lr-expr-list (p)))

THEOREM: lr-expr-lr-set-expr-nx
(listp (offset (p-pc (p))) $\land$ listp (lr-expr-list (p)))
$\rightarrow$ (lr-expr-list (lr-set-expr ($p1$, p, nx (offset (p-pc (p)))))
$=$ cdr (lr-expr-list (p)))

THEOREM: lr-expr-list-lr-set-pos-dv-1
listp (lr-expr (p))
$\rightarrow$ (lr-expr-list (lr-set-pos (p, dv (offset (p-pc (p)), `1`)))
$=$ cdr (lr-expr (p)))

```
;; If FLAG is 'LIST then state contains a list of expressions,
;; otherwise it is just one.
;; Returns a P-STATE.  The result is left on the temp stack.
;; If the error flag of the resulting state is 'HALT then we terminated
;; normally.  If the flag is 'RUN we have not terminated yet.
;; If the flag is anything else we got an error.
```

DEFINITION:
lr-eval ($flag$, l, c)
$=$ **if** p-psw (l) $\neq$ `'run` **then** l
elseif $flag$ $=$ `'list`

```
then if offset (p-pc (l)) ≃ nil
       then lr-set-error (l, 'bad-list-position)
       elseif listp (lr-expr-list (l))
       then lr-eval ('list,
                     lr-set-expr (lr-eval (t, l, c), l, nx (offset (p-pc (l)))),
                     c)
       else l endif
elseif c ≃ 0 then lr-set-error (l, 'out-of-time)
elseif litatom (lr-expr (l))
then lr-push-tstk (l, local-var-value (lr-expr (l), p-ctrl-stk (l)))
elseif lr-expr (l) ≃ nil then lr-set-error (l, 'bad-expression)
elseif car (lr-expr (l)) = 'if
then let test be lr-if-ok (lr-eval (t,
                                    lr-set-pos (l,
                                                dv (offset (p-pc (l)), 1)),
                                    c))
       in
       if p-psw (test) = 'run
       then if top (p-temp-stk (test)) ≠ LR-F-ADDR
              then lr-eval (t,
                            lr-set-expr (lr-pop-tstk (test),
                                         l,
                                         dv (offset (p-pc (l)), 2)),
                            c)
              else lr-eval (t,
                            lr-set-expr (lr-pop-tstk (test),
                                         l,
                                         dv (offset (p-pc (l)), 3)),
                            c) endif
       else test endif endlet
elseif car (lr-expr (l)) = S-TEMP-EVAL
then let l1 be lr-eval (t, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c)
       in
       lr-set-temp (l1, top (p-temp-stk (l1)), caddr (lr-expr (l))) endlet
elseif car (lr-expr (l)) = S-TEMP-TEST
then let l1 be lr-eval (t, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c)
       in
       if p-max-temp-stk-size (l)
          ≮  (2 + length (p-temp-stk (l)))
       then if lr-eval-temp-setp (l) then lr-do-temp-fetch (l)
              else lr-set-temp (l1,
                                top (p-temp-stk (l1)),
                                caddr (lr-expr (l))) endif
       else lr-set-error (l,
```

```
                          'lr-temp-setp-temp-stack-overflow) endif endlet
    elseif car (lr-expr (l)) = S-TEMP-FETCH then lr-do-temp-fetch (l)
    elseif car (lr-expr (l)) = 'quote
    then lr-push-tstk (l, cadr (lr-expr (l)))
    elseif p-psw (lr-eval ('list, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c))
          ≠ 'run
    then lr-eval ('list, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c)
    elseif subrp (car (lr-expr (l)))
    then lr-apply-subr (l,
                        lr-eval ('list,
                                 lr-set-pos (l, dv (offset (p-pc (l)), 1)),
                                 c))
    elseif litatom (car (lr-expr (l)))
    then let fs be lr-funcall (l,
                               lr-eval ('list,
                                        lr-set-pos (l,
                                                    dv (offset (p-pc (l)), 1)),
                                        c))
         in
         lr-set-expr (lr-pop-cstk (lr-eval (t, fs, c - 1)),
                      l,
                      offset (p-pc (l))) endlet
    else lr-set-error (l, 'bad-instruction) endif
```

```
;; Proper LR STATES

;; Sometimes we only need to know that LR-PROPER-P-AREASP holds on
;; a data-segment instead of LR-PROPER-P-DATA-SEGMENTP
```

DEFINITION:
lr-proper-p-areasp (*data-seg*)

```
=   if data-seg ≃ nil then data-seg = nil
    else let area be car (data-seg)
         in
         litatom (car (area))
         ∧  listp (cdr (area))
         ∧  (¬ definedp (car (area), cdr (data-seg)))
         ∧  lr-proper-p-areasp (cdr (data-seg)) endlet endif
```

```
;; First we prove that LR-EVAL preserves PROPER-P-STATEP.
```

THEOREM: p-accessors-lr-funcall
(p-prog-segment (lr-funcall (l, *new-l*)) = p-prog-segment (*new-l*))
∧ (p-data-segment (lr-funcall (l, *new-l*)) = p-data-segment (*new-l*))

$\land$ (p-max-ctrl-stk-size (lr-funcall (l, *new-l*))
= p-max-ctrl-stk-size (*new-l*))
$\land$ (p-max-temp-stk-size (lr-funcall (l, *new-l*))
= p-max-temp-stk-size (*new-l*))
$\land$ (p-word-size (lr-funcall (l, *new-l*)) = p-word-size (*new-l*))

THEOREM: p-accessors-lr-push-tstk
(p-pc (lr-push-tstk (s, v)) = p-pc (s))
$\land$ (p-ctrl-stk (lr-push-tstk (s, v)) = p-ctrl-stk (s))
$\land$ (p-prog-segment (lr-push-tstk (s, v)) = p-prog-segment (s))
$\land$ (p-data-segment (lr-push-tstk (s, v)) = p-data-segment (s))
$\land$ (p-max-ctrl-stk-size (lr-push-tstk (s, v)) = p-max-ctrl-stk-size (s))
$\land$ (p-max-temp-stk-size (lr-push-tstk (s, v)) = p-max-temp-stk-size (s))
$\land$ (p-word-size (lr-push-tstk (s, v)) = p-word-size (s))

THEOREM: p-accessors-lr-if-ok
(p-pc (lr-if-ok (l)) = p-pc (l))
$\land$ (p-ctrl-stk (lr-if-ok (l)) = p-ctrl-stk (l))
$\land$ (p-temp-stk (lr-if-ok (l)) = p-temp-stk (l))
$\land$ (p-prog-segment (lr-if-ok (l)) = p-prog-segment (l))
$\land$ (p-data-segment (lr-if-ok (l)) = p-data-segment (l))
$\land$ (p-max-ctrl-stk-size (lr-if-ok (l)) = p-max-ctrl-stk-size (l))
$\land$ (p-max-temp-stk-size (lr-if-ok (l)) = p-max-temp-stk-size (l))
$\land$ (p-word-size (lr-if-ok (l)) = p-word-size (l))

THEOREM: p-accessors-lr-set-temp
(p-pc (lr-set-temp (s, v, n)) = p-pc (s))
$\land$ (p-ctrl-stk (lr-set-temp (s, v, n))
= **if** p-psw (s) = `'run`
then set-local-var-value (v, n, p-ctrl-stk (s))
else p-ctrl-stk (s) **endif**)
$\land$ (p-temp-stk (lr-set-temp (s, v, n)) = p-temp-stk (s))
$\land$ (p-prog-segment (lr-set-temp (s, v, n)) = p-prog-segment (s))
$\land$ (p-data-segment (lr-set-temp (s, v, n)) = p-data-segment (s))
$\land$ (p-max-ctrl-stk-size (lr-set-temp (s, v, n)) = p-max-ctrl-stk-size (s))
$\land$ (p-max-temp-stk-size (lr-set-temp (s, v, n)) = p-max-temp-stk-size (s))
$\land$ (p-word-size (lr-set-temp (s, v, n)) = p-word-size (s))
$\land$ (p-psw (lr-set-temp (s, v, n)) = p-psw (s))

THEOREM: p-accessors-lr-do-temp-fetch
(p-pc (lr-do-temp-fetch (s)) = p-pc (s))
$\land$ (p-ctrl-stk (lr-do-temp-fetch (s)) = p-ctrl-stk (s))
$\land$ (p-prog-segment (lr-do-temp-fetch (s)) = p-prog-segment (s))
$\land$ (p-data-segment (lr-do-temp-fetch (s)) = p-data-segment (s))
$\land$ (p-max-ctrl-stk-size (lr-do-temp-fetch (s)) = p-max-ctrl-stk-size (s))

$\land$ (p-max-temp-stk-size (lr-do-temp-fetch (s)) = p-max-temp-stk-size (s))
$\land$ (p-word-size (lr-do-temp-fetch (s)) = p-word-size (s))

THEOREM: p-accessors-lr-pop-cstk
(p-pc (lr-pop-cstk (s)) = p-pc (s))
$\land$ (p-ctrl-stk (lr-pop-cstk (s))
= **if** p-psw (s) = `'run` **then** pop (p-ctrl-stk (s))
else p-ctrl-stk (s) **endif**)
$\land$ (p-temp-stk (lr-pop-cstk (s)) = p-temp-stk (s))
$\land$ (p-prog-segment (lr-pop-cstk (s)) = p-prog-segment (s))
$\land$ (p-data-segment (lr-pop-cstk (s)) = p-data-segment (s))
$\land$ (p-max-ctrl-stk-size (lr-pop-cstk (s)) = p-max-ctrl-stk-size (s))
$\land$ (p-max-temp-stk-size (lr-pop-cstk (s)) = p-max-temp-stk-size (s))
$\land$ (p-word-size (lr-pop-cstk (s)) = p-word-size (s))
$\land$ (p-psw (lr-pop-cstk (s)) = p-psw (s))

THEOREM: lr-eval-if-p-psw-1
(($flag \neq$ `'list`)
$\land$ (p-psw (lr-eval ($flag$, l, c)) = `'run`)
$\land$ (car (lr-expr (l)) = `'if`)
$\land$ (p-psw (l) = `'run`)
$\land$ ($c \not\simeq$ `0`)
$\land$ listp (lr-expr (l)))
$\rightarrow$ (p-psw (lr-eval (**t**, lr-set-pos (l, dv (offset (p-pc (l)), `1`)), c)) = `'run`)

EVENT: Disable lr-eval-if-p-psw-1.

THEOREM: adp-offset-untag-add-addr
adp-offset (untag (add-addr ($addr$, n))) = (offset ($addr$) + n)

THEOREM: adp-offset-untag-sub-addr
adp-offset (untag (sub-addr ($addr$, n))) = (offset ($addr$) $-$ n)

THEOREM: adp-name-untag-sub-addr
adp-name (untag (sub-addr ($addr$, n))) = adp-name (untag ($addr$))

THEOREM: adp-offset-cons
adp-offset (cons ($area\text{-}name$, $offset$)) = $offset$

THEOREM: p-accessors-lr->p
(p-pc (lr->p (l)) = lr-p-pc (l))
$\land$ (p-ctrl-stk (lr->p (l)) = p-ctrl-stk (l))
$\land$ (p-temp-stk (lr->p (l)) = p-temp-stk (l))
$\land$ (p-prog-segment (lr->p (l)) = comp-programs (p-prog-segment (l)))
$\land$ (p-data-segment (lr->p (l)) = p-data-segment (l))

$\land$ (p-max-ctrl-stk-size (lr->p (l)) = p-max-ctrl-stk-size (l))
$\land$ (p-max-temp-stk-size (lr->p (l)) = p-max-temp-stk-size (l))
$\land$ (p-word-size (lr->p (l)) = p-word-size (l))
$\land$ (p-psw (lr->p (l)) = p-psw (l))

THEOREM: type-lr-p-pc
type (lr-p-pc (l)) = `'pc`

THEOREM: cddr-nil-lr-p-pc
cddr (lr-p-pc (l)) = **nil**

THEOREM: listp-untag-lr-p-pc
listp (untag (lr-p-pc (l)))

THEOREM: numberp-cdr-lr-p-pc
cdr (untag (lr-p-pc (l))) $\in$ **N**

THEOREM: car-untag-lr-p-pc
car (untag (lr-p-pc (p))) = car (untag (p-pc (p)))

THEOREM: area-name-lr-p-pc
area-name (lr-p-pc (p)) = area-name (p-pc (p))

THEOREM: definedp-comp-programs-1-definedp-orig
definedp (x, comp-programs-1 (*programs*)) = definedp (x, *programs*)

THEOREM: definedp-append
definedp (x, append (*l1*, *l2*)) = (definedp (x, *l1*) $\lor$ definedp (x, *l2*))

THEOREM: definedp-comp-programs-definedp-orig
definedp (x, *programs*) $\rightarrow$ definedp (x, comp-programs (*programs*))

THEOREM: p-accessors-p-halt
(p-pc (p-halt (p, psw)) = p-pc (p))
$\land$ (p-ctrl-stk (p-halt (p, psw)) = p-ctrl-stk (p))
$\land$ (p-temp-stk (p-halt (p, psw)) = p-temp-stk (p))
$\land$ (p-prog-segment (p-halt (p, psw)) = p-prog-segment (p))
$\land$ (p-data-segment (p-halt (p, psw)) = p-data-segment (p))
$\land$ (p-max-ctrl-stk-size (p-halt (p, psw)) = p-max-ctrl-stk-size (p))
$\land$ (p-max-temp-stk-size (p-halt (p, psw)) = p-max-temp-stk-size (p))
$\land$ (p-word-size (p-halt (p, psw)) = p-word-size (p))
$\land$ (p-psw (p-halt (p, psw)) = psw)

EVENT: Disable p-halt.

THEOREM: p-accessors-p-set-pc
(p-pc (p-set-pc (p, pc)) = pc)
∧ (p-ctrl-stk (p-set-pc (p, pc)) = p-ctrl-stk (p))
∧ (p-temp-stk (p-set-pc (p, pc)) = p-temp-stk (p))
∧ (p-prog-segment (p-set-pc (p, pc)) = p-prog-segment (p))
∧ (p-data-segment (p-set-pc (p, pc)) = p-data-segment (p))
∧ (p-max-ctrl-stk-size (p-set-pc (p, pc)) = p-max-ctrl-stk-size (p))
∧ (p-max-temp-stk-size (p-set-pc (p, pc)) = p-max-temp-stk-size (p))
∧ (p-word-size (p-set-pc (p, pc)) = p-word-size (p))
∧ (p-psw (p-set-pc (p, pc)) = p-psw (p))

EVENT: Disable p-set-pc.

THEOREM: p-psw-not-run
(p-psw ($p\text{-}state$) ≠ **'run**) → (p ($p\text{-}state$, $clock$) = $p\text{-}state$)

THEOREM: p-psw-p-halt-x-y-error-msg
p (p-halt ($p\text{-}state$, x-y-error-msg (x, y)), n)
= p-halt ($p\text{-}state$, x-y-error-msg (x, y))

EVENT: Disable p-psw-p-halt-x-y-error-msg.

THEOREM: p-accessors-p-run-subr
(p-prog-segment (p-run-subr ($subr$, p)) = p-prog-segment (p))
∧ (p-max-ctrl-stk-size (p-run-subr ($subr$, p)) = p-max-ctrl-stk-size (p))
∧ (p-max-temp-stk-size (p-run-subr ($subr$, p)) = p-max-temp-stk-size (p))
∧ (p-word-size (p-run-subr ($subr$, p)) = p-word-size (p))

THEOREM: p-accessors-lr-apply-subr
(p-pc (lr-apply-subr ($l1$, $l2$)) = p-pc ($l2$))
∧ (p-prog-segment (lr-apply-subr ($l1$, $l2$)) = p-prog-segment ($l2$))
∧ (p-max-ctrl-stk-size (lr-apply-subr ($l1$, $l2$))
= p-max-ctrl-stk-size ($l2$))
∧ (p-max-temp-stk-size (lr-apply-subr ($l1$, $l2$))
= p-max-temp-stk-size ($l2$))
∧ (p-word-size (lr-apply-subr ($l1$, $l2$)) = p-word-size ($l2$))

THEOREM: p-prog-segment-lr-eval
p-prog-segment (lr-eval ($flag$, l, c)) = p-prog-segment (l)

THEOREM: p-max-ctrl-stk-size-lr-eval
p-max-ctrl-stk-size (lr-eval ($flag$, l, c)) = p-max-ctrl-stk-size (l)

THEOREM: p-max-temp-stk-size-lr-eval
p-max-temp-stk-size (lr-eval ($flag$, l, c)) = p-max-temp-stk-size (l)

THEOREM: p-word-size-lr-eval
p-word-size (lr-eval (*flag*, *l*, *c*)) = p-word-size (*l*)

THEOREM: area-name-p-pc-lr-eval
area-name (p-pc (lr-eval (*flag*, *l*, *c*))) = area-name (p-pc (*l*))

THEOREM: lr-programs-properp-lr-eval
lr-programs-properp (lr-eval (*flag*, *l*, *c*), *table*)
= lr-programs-properp (*l*, *table*)

THEOREM: definedp-deposit
definedp (*tag*, deposit (*anything*, *addr*, *data-seg*)) = definedp (*tag*, *data-seg*)

THEOREM: deposit-a-list-cons-opener
deposit-a-list (cons (*x*, *list*), *addr*, *data-seg*)
= deposit (*x*, *addr*, deposit-a-list (*list*, add1-addr (*addr*), *data-seg*))

THEOREM: deposit-a-list-nil
deposit-a-list (**nil**, *addr*, *data-seg*) = *data-seg*

EVENT: Disable deposit-a-list.

THEOREM: assoc-put-assoc-3
assoc (*name1*, put-assoc (*val*, *name2*, *alist*))
= **if** *name1* = *name2*
then if definedp (*name1*, *alist*) **then** cons (*name1*, *val*)
else f endif
else assoc (*name1*, *alist*) **endif**

EVENT: Disable assoc-put-assoc-3.

THEOREM: adpp-lessp-offset-deposit
((*offset* < length (cdr (assoc (*name*, *data-seg*)))) ∧ definedp (*name*, *data-seg*))
→ (*offset* < length (cdr (assoc (*name*, deposit (*anything*, *anywhere*, *data-seg*)))))

THEOREM: adpp-deposit-anything-at-all
adpp (*adp*, *data-seg*) → adpp (*adp*, deposit (*anything*, *addr2*, *data-seg*))

EVENT: Disable adpp-lessp-offset-deposit.

EVENT: Disable adpp-deposit-anything-at-all.

THEOREM: adpp-untag-definedp-area-name
adpp (untag (*addr*), *data-seg*) → definedp (area-name (*addr*), *data-seg*)

EVENT: Disable adpp-untag-definedp-area-name.

THEOREM: adpp-cons-pack-definedp-area-name
adpp (cons (pack (xxx), *offset*), *data-seg*) $\rightarrow$ definedp (pack (xxx), *data-seg*)

THEOREM: adpp-untag-numberp-offset
adpp (untag (*addr*), *data-seg*) $\rightarrow$ (offset (*addr*) $\in$ **N**)

EVENT: Disable adpp-untag-numberp-offset.

THEOREM: adpp-untag-listp
adpp (untag (*addr*), *data-seg*) $\rightarrow$ listp (untag (*addr*))

EVENT: Disable adpp-untag-listp.

THEOREM: adpp-add-addr-0
(adpp (untag (*addr*), *data-seg*)
$\wedge$ (cddr (*addr*) = **nil**)
$\wedge$ (type (*addr*) = `'addr`)
$\wedge$ ($n \simeq 0$))
$\rightarrow$ (add-addr (*addr*, n) = *addr*)

EVENT: Disable adpp-add-addr-0.

THEOREM: adpp-untag-lessp-offset
adpp (untag (*addr*), *data-seg*)
$\rightarrow$ (offset (*addr*) $<$ length (cdr (assoc (area-name (*addr*), *data-seg*))))

EVENT: Disable adpp-untag-lessp-offset.

THEOREM: adpp-same-signature
same-signature (*data-seg2*, *data-seg1*)
$\rightarrow$ (adpp (*adp*, *data-seg2*) = adpp (*adp*, *data-seg1*))

EVENT: Disable adpp.

THEOREM: p-objectp-similar-p-states
(p-objectp (*object*, *p0*)
$\wedge$ same-signature (p-data-segment (*p0*), p-data-segment (*p1*))
$\wedge$ (p-prog-segment (*p0*) = p-prog-segment (*p1*))
$\wedge$ (p-word-size (*p0*) = p-word-size (*p1*)))
$\rightarrow$ p-objectp (*object*, *p1*)

THEOREM: all-p-objectps-lr->p-similar-states
(all-p-objectps (lst, $p0$)
$\wedge$ (p-word-size ($p0$) = p-word-size ($p1$))
$\wedge$ (p-prog-segment ($p0$) = p-prog-segment ($p1$))
$\wedge$ same-signature (p-data-segment ($p0$), p-data-segment ($p1$)))
$\rightarrow$ all-p-objectps (lst, $p1$)

THEOREM: proper-p-data-segmentp-lr->p-similar-states
(proper-p-data-segmentp ($data\text{-}seg$, $p0$)
$\wedge$ (p-word-size ($p0$) = p-word-size ($p1$))
$\wedge$ (p-prog-segment ($p0$) = p-prog-segment ($p1$))
$\wedge$ same-signature (p-data-segment ($p0$), p-data-segment ($p1$)))
$\rightarrow$ proper-p-data-segmentp ($data\text{-}seg$, $p1$)

THEOREM: proper-p-temp-var-dclsp-lr->p-similar-states
(proper-p-temp-var-dclsp ($temp\text{-}var\text{-}dcls$, $p0$)
$\wedge$ (p-word-size ($p0$) = p-word-size ($p1$))
$\wedge$ (p-prog-segment ($p0$) = p-prog-segment ($p1$))
$\wedge$ same-signature (p-data-segment ($p0$), p-data-segment ($p1$)))
$\rightarrow$ proper-p-temp-var-dclsp ($temp\text{-}var\text{-}dcls$, $p1$)

THEOREM: proper-p-instructionp-similar-p-states
(proper-p-instructionp (ins, $name$, $p0$)
$\wedge$ same-signature (p-data-segment ($p0$), p-data-segment ($p1$))
$\wedge$ (p-prog-segment ($p0$) = p-prog-segment ($p1$))
$\wedge$ (p-word-size ($p0$) = p-word-size ($p1$)))
$\rightarrow$ proper-p-instructionp (ins, $name$, $p1$)

THEOREM: proper-labeled-p-instructionsp-lr->p-similar-states
(proper-labeled-p-instructionsp (lst, $name$, $p0$)
$\wedge$ same-signature (p-data-segment ($p0$), p-data-segment ($p1$))
$\wedge$ (p-prog-segment ($p0$) = p-prog-segment ($p1$))
$\wedge$ (p-word-size ($p0$) = p-word-size ($p1$)))
$\rightarrow$ proper-labeled-p-instructionsp (lst, $name$, $p1$)

THEOREM: proper-p-prog-segmentp-lr->p-similar-states
(proper-p-prog-segmentp ($programs$, $p0$)
$\wedge$ same-signature (p-data-segment ($p0$), p-data-segment ($p1$))
$\wedge$ (p-prog-segment ($p0$) = p-prog-segment ($p1$))
$\wedge$ (p-word-size ($p0$) = p-word-size ($p1$)))
$\rightarrow$ proper-p-prog-segmentp ($programs$, $p1$)

THEOREM: proper-p-temp-stkp-lr->p-similar-states
(proper-p-temp-stkp ($temp\text{-}stk$, $p0$)
$\wedge$ same-signature (p-data-segment ($p0$), p-data-segment ($p1$))

$\wedge$ (p-prog-segment ($p0$) = p-prog-segment ($p1$))
$\wedge$ (p-word-size ($p0$) = p-word-size ($p1$)))
$\rightarrow$ proper-p-temp-stkp (*temp-stk*, *p1*)

THEOREM: proper-p-alistp-lr->p-similar-states
(proper-p-alistp (*bindings*, *p0*)
$\wedge$ same-signature (p-data-segment ($p0$), p-data-segment ($p1$))
$\wedge$ (p-prog-segment ($p0$) = p-prog-segment ($p1$))
$\wedge$ (p-word-size ($p0$) = p-word-size ($p1$)))
$\rightarrow$ proper-p-alistp (*bindings*, *p1*)

THEOREM: proper-p-ctrl-stkp-lr->p-similar-states
(proper-p-ctrl-stkp (*ctrl-stk*, *name*, *p0*)
$\wedge$ same-signature (p-data-segment ($p0$), p-data-segment ($p1$))
$\wedge$ (p-prog-segment ($p0$) = p-prog-segment ($p1$))
$\wedge$ (p-word-size ($p0$) = p-word-size ($p1$)))
$\rightarrow$ proper-p-ctrl-stkp (*ctrl-stk*, *name*, *p1*)

```
;; Now we prove what the result of running the SUBRPs.  We
;; start with a sample state (that the rewriter can match with
;; P-APPLY-SUBR-STATE) and run it.  We are only interested in TEMP-STK and
;; DATA-SEGMENT of the result.  However the running of the Piton code can
;; be a bit tedious, so we try and prove both parts at once with the
;; following function P-GOOD-RESULTP.  This also has the not ERRORP check
;; inside of it so that we should only have one instance of the Piton
;; interpreter (P) in each theorem.  This should hopefully reduce the time
;; (and pain) of proving these theorems.
```

DEFINITION:
p-good-resultp (*p*, *data-seg*, *temp-stk*, *ctrl-stk*, *pc*)
= **if** p-psw (p) $\neq$ **'run then t**
else (p-data-segment (p) = *data-seg*)
$\wedge$ (p-temp-stk (p) = *temp-stk*)
$\wedge$ listp (*ctrl-stk*)
$\wedge$ (p-ctrl-stk (p) = *ctrl-stk*)
$\wedge$ (p-pc (p) = *pc*) **endif**

THEOREM: assoc-append-1
assoc (x, append (*list1*, *list2*))
= **if** definedp (x, *list1*) **then** assoc (x, *list1*)
else assoc (x, *list2*) **endif**

EVENT: Disable assoc-append-1.

THEOREM: lr-programs-properp-1-all-user-fnamesp-not-user-fnamep

(all-user-fnamesp (strip-cars (*programs*)) ∧ (¬ user-fnamep (*x*)))
→ (¬ definedp (*x*, *programs*))

THEOREM: definitions-subrps-lr-programs-properp
lr-programs-properp (*l*, *table*)
→ ((assoc (**'car**, comp-programs (p-prog-segment (*l*))) = P-CAR-CODE)
  ∧ (assoc (**'cdr**, comp-programs (p-prog-segment (*l*)))
    = P-CDR-CODE)
  ∧ (assoc (**'cons**, comp-programs (p-prog-segment (*l*)))
    = P-CONS-CODE)
  ∧ (assoc (**'false**, comp-programs (p-prog-segment (*l*)))
    = P-FALSE-CODE)
  ∧ (assoc (**'falsep**, comp-programs (p-prog-segment (*l*)))
    = P-FALSEP-CODE)
  ∧ (assoc (**'listp**, comp-programs (p-prog-segment (*l*)))
    = P-LISTP-CODE)
  ∧ (assoc (**'nlistp**, comp-programs (p-prog-segment (*l*)))
    = P-NLISTP-CODE)
  ∧ (assoc (**'true**, comp-programs (p-prog-segment (*l*)))
    = P-TRUE-CODE)
  ∧ (assoc (**'truep**, comp-programs (p-prog-segment (*l*)))
    = P-TRUEP-CODE))

EVENT: Disable lr-programs-properp-1-all-user-fnamesp-not-user-fnamep.

EVENT: Disable definitions-subrps-lr-programs-properp.

```
;; and now some openers for p-good-resultp
```

THEOREM: p-good-resultp-p-state-opener
p-good-resultp (p-state (*pc*,
    ctrl-stk,
    temp-stk,
    prog-seg,
    data-seg,
    max-ctrl-stk-size,
    max-temp-stk-size,
    word-size,
    'run),
  result-data-seg,
  result-temp-stk,
  result-ctrl-stk,
  result-pc)
= ((*data-seg* = *result-data-seg*)

$\wedge$ $(\textit{temp-stk} = \textit{result-temp-stk})$
$\wedge$ listp $(\textit{result-ctrl-stk})$
$\wedge$ $(\textit{ctrl-stk} = \textit{result-ctrl-stk})$
$\wedge$ $(\textit{pc} = \textit{result-pc}))$

THEOREM: p-good-resultp-p-halt-errorp-opener
$(\textit{psw} \neq \texttt{'run})$
$\rightarrow$ p-good-resultp (p-halt $(p,\ \textit{psw})$, $\textit{data-seg},\ \textit{temp-stk},\ \textit{ctrl-stk},\ \textit{pc})$

EVENT: Disable p-good-resultp.

THEOREM: all-p-objectps-bad-type
((get $(\textit{offset},\ \textit{lst}) \neq$ list (type (get $(\textit{offset},\ \textit{lst})$), untag (get $(\textit{offset},\ \textit{lst}))))$
$\wedge$ $(\textit{offset} \in \mathbf{N})$
$\wedge$ $(\textit{offset} <$ length $(\textit{lst})))$
$\rightarrow$ $(\neg$ all-p-objectps $(\textit{lst},\ p))$

THEOREM: proper-p-data-segmentp-bad-type
((fetch $(\textit{addr},\ \textit{data-seg})$
$\neq$ list (type (fetch $(\textit{addr},\ \textit{data-seg})$), untag (fetch $(\textit{addr},\ \textit{data-seg}))))$
$\wedge$ adpp (untag $(\textit{addr})$, $\textit{data-seg}))$
$\rightarrow$ $(\neg$ proper-p-data-segmentp $(\textit{data-seg},\ p))$

THEOREM: p-current-program-p-state
p-current-program (p-state $(\textit{pc}$,
$\textit{ctrl-stk}$,
$\textit{temp-stk}$,
$\textit{prog-seg}$,
$\textit{data-seg}$,
$\textit{max-ctrl-stk-size}$,
$\textit{max-temp-stk-size}$,
$\textit{word-size}$,
$\textit{psw}))$
$=$ assoc (area-name $(\textit{pc})$, $\textit{prog-seg})$

THEOREM: p-current-instruction-opener
p-current-instruction (p-state $(\textit{pc}$,
$\textit{temp-stk}$,
$\textit{ctrl-stk}$,
$\textit{prog-segment}$,
$\textit{data-segment}$,
$\textit{max-ctrl-stk-size}$,
$\textit{max-temp-stk-size}$,
$\textit{word-size}$,

$$
\begin{array}{ll}
 & \qquad\qquad\qquad psw)) \\
= & \text{unlabel}(\text{get}(\text{offset}(pc), \\
 & \qquad \text{program-body}(\text{assoc}(\text{area-name}(pc),\ \textit{prog-segment}))))
\end{array}
$$

EVENT: Disable p-current-instruction-opener.

THEOREM: fetch-deposit

$$
\begin{array}{l}
((\text{offset}(\textit{addr1}) \in \mathbf{N}) \land (\text{offset}(\textit{addr2}) \in \mathbf{N})) \\
\rightarrow \quad (\text{fetch}(\textit{addr1},\ \text{deposit}(\textit{value},\ \textit{addr2},\ \textit{data-seg})) \\
\qquad = \quad \textbf{if } \text{definedp}(\text{area-name}(\textit{addr2}),\ \textit{data-seg}) \\
\qquad\qquad \textbf{then if } \text{area-name}(\textit{addr1}) = \text{area-name}(\textit{addr2}) \\
\qquad\qquad\qquad \textbf{then if } \text{offset}(\textit{addr1}) = \text{offset}(\textit{addr2}) \ \textbf{then } \textit{value} \\
\qquad\qquad\qquad\qquad \textbf{else } \text{fetch}(\textit{addr1},\ \textit{data-seg}) \textbf{ endif} \\
\qquad\qquad\qquad \textbf{else } \text{fetch}(\textit{addr1},\ \textit{data-seg}) \textbf{ endif} \\
\qquad\qquad \textbf{else } \text{fetch}(\textit{addr1},\ \textit{data-seg}) \textbf{ endif})
\end{array}
$$

```
;; add-addr
```

THEOREM: area-name-add-addr
$\text{area-name}(\text{add-addr}(\textit{addr},\ n)) = \text{area-name}(\textit{addr})$

THEOREM: offset-add-addr
$\text{offset}(\text{add-addr}(\textit{addr},\ n)) = (\text{offset}(\textit{addr}) + n)$

THEOREM: adp-name-untag-add-addr
$\text{adp-name}(\text{untag}(\text{add-addr}(\textit{addr},\ n))) = \text{area-name}(\textit{addr})$

THEOREM: add-addr-of-non-number
$(n \notin \mathbf{N}) \rightarrow (\text{add-addr}(\textit{addr},\ n) = \text{add-addr}(\textit{addr},\ \mathtt{0}))$

THEOREM: add-addr-add-addr
$\text{add-addr}(\text{add-addr}(\textit{addr},\ n),\ m) = \text{add-addr}(\textit{addr},\ n + m)$

THEOREM: listp-untag-add-addr
$\text{listp}(\text{untag}(\text{add-addr}(\textit{addr},\ n)))$

THEOREM: type-add-addr
$\text{type}(\text{add-addr}(\textit{addr},\ n)) = \text{type}(\textit{addr})$

THEOREM: cddr-add-addr
$\text{cddr}(\text{add-addr}(\textit{addr},\ n)) = \textbf{nil}$

THEOREM: area-name-lr-return-pc
$\text{area-name}(\text{lr-return-pc}(l)) = \text{area-name}(\text{p-pc}(l))$

THEOREM: listp-untag-lr-return-pc
listp (untag (lr-return-pc (l)))

THEOREM: type-lr-return-pc
type (lr-return-pc (l)) = `'pc`

THEOREM: cddr-lr-return-pc
cddr (lr-return-pc (l)) = **nil**

THEOREM: numberp-offset-return-pc
offset (lr-return-pc (l)) $\in \mathbf{N}$

THEOREM: numberp-cdr-untag-return-pc
cdr (untag (lr-return-pc (l))) $\in \mathbf{N}$

THEOREM: car-untag-lr-return-pc
car (untag (lr-return-pc (l))) = car (untag (p-pc (l)))

`;; sub-addr`

THEOREM: area-name-sub-addr
area-name (sub-addr ($addr$, n)) = area-name ($addr$)

THEOREM: cddr-sub-addr
cddr (sub-addr ($addr$, n)) = **nil**

THEOREM: type-sub-addr
type (sub-addr ($addr$, n)) = type ($addr$)

THEOREM: listp-untag-sub-addr
listp (untag (sub-addr ($addr$, n)))

THEOREM: offset-sub-addr
offset (sub-addr ($addr$, n)) = (offset ($addr$) $-$ n)

`;; LR-BOUNDARY-NODEP`

THEOREM: lr-boundary-nodep-sub-addr
lr-boundary-nodep ($addr$)
$\rightarrow$ lr-boundary-nodep (sub-addr ($addr$, identity (LR-NODE-SIZE)))

THEOREM: lr-boundary-nodep-add-addr-lr-node-size
lr-boundary-nodep ($addr$)
$\rightarrow$ lr-boundary-nodep (add-addr ($addr$, identity (LR-NODE-SIZE)))

EVENT: Disable lr-boundary-nodep.

```
;; LR-NODEP
```

THEOREM: lr-nodep-opener

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{lr-nodep}\,(\mathit{addr},\ \mathit{data\text{-}seg})} \\
= & ((\text{type}\,(\mathit{addr}) = \texttt{'addr}) \\
& \quad \land \quad (\text{cddr}\,(\mathit{addr}) = \mathbf{nil}) \\
& \quad \land \quad \text{listp}\,(\mathit{addr}) \\
& \quad \land \quad \text{adpp}\,(\text{untag}\,(\mathit{addr}),\ \mathit{data\text{-}seg}) \\
& \quad \land \quad \text{lr-boundary-nodep}\,(\mathit{addr}) \\
& \quad \land \quad (\text{area-name}\,(\mathit{addr}) = \text{identity}\,(\textsc{lr-heap-name})))
\end{array}
$$

EVENT: Disable lr-nodep.

```
;; LR-GOOD-POINTERP
```

THEOREM: lr-good-pointerp-opener

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{lr-good-pointerp}\,(\mathit{addr},\ \mathit{data\text{-}seg})} \\
= & ((\text{type}\,(\mathit{addr}) = \texttt{'addr}) \\
& \quad \land \quad (\text{cddr}\,(\mathit{addr}) = \mathbf{nil}) \\
& \quad \land \quad \text{listp}\,(\mathit{addr}) \\
& \quad \land \quad \text{adpp}\,(\text{untag}\,(\mathit{addr}),\ \mathit{data\text{-}seg}) \\
& \quad \land \quad \text{lr-boundary-nodep}\,(\mathit{addr}) \\
& \quad \land \quad (\text{area-name}\,(\mathit{addr}) = \text{identity}\,(\textsc{lr-heap-name})) \\
& \quad \land \quad (\text{type}\,(\text{fetch}\,(\text{add-addr}\,(\mathit{addr},\ \text{identity}\,(\textsc{lr-ref-count-offset})), \\
& \qquad\qquad\qquad\qquad \mathit{data\text{-}seg})) \\
& \qquad\quad = \quad \texttt{'nat}))
\end{array}
$$

EVENT: Disable lr-good-pointerp.

THEOREM: equal-plus-remainder-0-fact

$$
\begin{array}{ll}
\multicolumn{2}{l}{(((\mathit{offset1}\ \mathbf{mod}\ \mathit{max}) = 0)} \\
\land & ((\mathit{offset2}\ \mathbf{mod}\ \mathit{max}) = 0) \\
\land & (n < \mathit{max}) \\
\land & (m < \mathit{max}) \\
\land & (\mathit{offset1} \in \mathbf{N}) \\
\land & (\mathit{offset2} \in \mathbf{N})) \\
\rightarrow & (((n + \mathit{offset1}) = (m + \mathit{offset2})) \\
& \quad = \quad ((\text{fix}\,(n) = \text{fix}\,(m)) \land (\mathit{offset1} = \mathit{offset2})))
\end{array}
$$

THEOREM: lr-boundary-offsetp-equal-plus-fact

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-boundary-offsetp}\,(\mathit{offset1})} \\
\land & \text{lr-boundary-offsetp}\,(\mathit{offset2}) \\
\land & (n < \textsc{lr-node-size}) \\
\land & (m < \textsc{lr-node-size})
\end{array}
$$

$$
\begin{array}{ll}
\wedge & (\mathit{offset1} \in \mathbf{N}) \\
\wedge & (\mathit{offset2} \in \mathbf{N})) \\
\rightarrow & (((n + \mathit{offset1}) = (m + \mathit{offset2})) \\
& \quad = \quad ((\mathrm{fix}\,(n) = \mathrm{fix}\,(m)) \wedge (\mathit{offset1} = \mathit{offset2})))
\end{array}
$$

THEOREM: good-posp-list-nx-t-simple

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{good-posp}\,(\texttt{'list}, \mathit{pos}, \mathit{body})} \\
\wedge & \text{listp}\,(\mathit{pos}) \\
\wedge & (\text{car}\,(\text{last}\,(\mathit{pos})) < \text{length}\,(\text{cur-expr}\,(\text{butlast}\,(\mathit{pos}), \mathit{body})))) \\
\rightarrow & (\text{good-posp}\,(\texttt{'list}, \text{nx}\,(\mathit{pos}), \mathit{body}) \wedge \text{good-posp1}\,(\mathit{pos}, \mathit{body}))
\end{array}
$$

THEOREM: lr-programs-properp-1-lr-proper-exprp

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-programs-properp-1}\,(\mathit{progs}, \mathit{program\text{-}names}, \mathit{table}) \wedge (\mathit{prog} \in \mathit{progs}))} \\
\rightarrow & \text{lr-proper-exprp}\,(\mathbf{t}, \\
& \qquad \text{program-body}\,(\mathit{prog}), \\
& \qquad \mathit{program\text{-}names}, \\
& \qquad \text{formal-vars}\,(\mathit{prog}), \\
& \qquad \text{strip-cars}\,(\text{temp-var-dcls}\,(\mathit{prog})), \\
& \qquad \mathit{table})
\end{array}
$$

THEOREM: lr-proper-exprp-list-lr-proper-get-t

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{lr-proper-exprp}\,(\texttt{'list}, \mathit{expr}, \mathit{pnames}, \mathit{formals}, \mathit{temps}, \mathit{table})} \\
\rightarrow & (\text{lr-proper-exprp}\,(\mathbf{t}, \text{get}\,(n, \mathit{expr}), \mathit{pnames}, \mathit{formals}, \mathit{temps}, \mathit{table}) \\
& \quad = \quad (n < \text{length}\,(\mathit{expr})))
\end{array}
$$

THEOREM: lr-proper-exprp-t-lr-proper-get-t

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{car}\,(\mathit{expr}) \neq \text{S-TEMP-FETCH})} \\
\wedge & (\text{car}\,(\mathit{expr}) \neq \text{S-TEMP-EVAL}) \\
\wedge & (\text{car}\,(\mathit{expr}) \neq \text{S-TEMP-TEST}) \\
\wedge & (\text{car}\,(\mathit{expr}) \neq \texttt{'quote}) \\
\wedge & (n \not\simeq \texttt{0}) \\
\wedge & \text{lr-proper-exprp}\,(\mathbf{t}, \mathit{expr}, \mathit{pnames}, \mathit{formals}, \mathit{temps}, \mathit{table})) \\
\rightarrow & (\text{lr-proper-exprp}\,(\mathbf{t}, \text{get}\,(n, \mathit{expr}), \mathit{pnames}, \mathit{formals}, \mathit{temps}, \mathit{table}) \\
& \quad = \quad (n < \text{length}\,(\mathit{expr})))
\end{array}
$$

EVENT: Disable lr-proper-exprp-list-lr-proper-get-t.

THEOREM: lr-proper-exprp-lr-proper-exprp-cur-expr

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-proper-exprp}\,(\mathbf{t}, \mathit{body}, \mathit{pnames}, \mathit{formals}, \mathit{temps}, \mathit{table}) \wedge \text{good-posp1}\,(\mathit{pos}, \mathit{body}))} \\
\rightarrow & \text{lr-proper-exprp}\,(\mathbf{t}, \text{cur-expr}\,(\mathit{pos}, \mathit{body}), \mathit{pnames}, \mathit{formals}, \mathit{temps}, \mathit{table})
\end{array}
$$

THEOREM: lr-programs-properp-lr-programs-properp-1

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-programs-properp}\,(l, \mathit{table}) \wedge (\mathit{prog\text{-}seg} = \text{p-prog-segment}\,(l)))} \\
\rightarrow & (\text{lr-programs-properp-1}\,(\text{p-prog-segment}\,(l), \\
& \qquad \text{strip-logic-fnames}\,(\text{cdr}\,(\mathit{prog\text{-}seg})),
\end{array}
$$

$$\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad \mathit{table}) \\
\quad \land \quad \text{definedp}\,(\text{area-name}\,(\text{p-pc}\,(l)),\ \mathit{prog\text{-}seg}))
\end{array}$$

THEOREM: lr-programs-properp-lr-proper-exprp-lr-expr

$$\begin{array}{l}
(\text{lr-programs-properp}\,(l,\ \mathit{table}) \\
\land \quad \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l)))) \\
\to \quad \text{lr-proper-exprp}\,(\mathtt{t}, \\
\qquad\qquad\qquad \text{lr-expr}\,(l), \\
\qquad\qquad\qquad \text{strip-logic-fnames}\,(\text{cdr}\,(\text{p-prog-segment}\,(l))), \\
\qquad\qquad\qquad \text{formal-vars}\,(\text{p-current-program}\,(l)), \\
\qquad\qquad\qquad \text{strip-cars}\,(\text{temp-var-dcls}\,(\text{p-current-program}\,(l))), \\
\qquad\qquad\qquad \mathit{table})
\end{array}$$

THEOREM: lr-proper-exprp-length-cur-expr

$$\begin{array}{l}
(\text{lr-proper-exprp}\,(\mathtt{t},\ \mathit{expr},\ \mathit{pnames},\ \mathit{formals},\ \mathit{temps},\ \mathit{table}) \\
\land \quad \text{listp}\,(\mathit{expr}) \\
\land \quad (\text{subrp}\,(\text{car}\,(\mathit{expr})) \lor \text{body}\,(\text{car}\,(\mathit{expr}))) \\
\land \quad (\text{car}\,(\mathit{expr}) \neq \mathtt{'quote})) \\
\to \quad (\text{length}\,(\mathit{expr}) = (1 + \text{arity}\,(\text{car}\,(\mathit{expr}))))
\end{array}$$

THEOREM: listp-comp-body-1

$$\begin{array}{l}
\text{listp}\,(\text{comp-body-1}\,(\mathit{flag},\ \mathit{body},\ n)) \\
= \quad \textbf{if}\ \mathit{flag} = \mathtt{'list}\ \textbf{then}\ \text{listp}\,(\mathit{body}) \\
\qquad \textbf{else t endif}
\end{array}$$

THEOREM: car-append

$$\text{listp}\,(x) \to (\text{car}\,(\text{append}\,(x,\ y)) = \text{car}\,(x))$$

THEOREM: length-cdr-comp-if-comp-body

$$\begin{array}{l}
\text{length}\,(\text{comp-if}\,(\text{comp-body-1}\,(\mathtt{t},\ \mathit{test},\ \mathit{n1}), \\
\qquad\qquad\qquad \text{comp-body-1}\,(\mathtt{t},\ \mathit{then},\ \mathit{n2}), \\
\qquad\qquad\qquad \text{comp-body-1}\,(\mathtt{t},\ \mathit{else},\ \mathit{n3}), \\
\qquad\qquad\qquad n)) \\
= \quad (\text{length}\,(\text{comp-body-1}\,(\mathtt{t},\ \mathit{test},\ \mathit{n1})) \\
\qquad + \quad \text{length}\,(\text{comp-body-1}\,(\mathtt{t},\ \mathit{then},\ \mathit{n2})) \\
\qquad + \quad \text{length}\,(\text{comp-body-1}\,(\mathtt{t},\ \mathit{else},\ \mathit{n3})) \\
\qquad + \quad \mathtt{4})
\end{array}$$

THEOREM: lr-p-c-size-list-0-opener

$$\text{lr-p-c-size-list}\,(\mathtt{0},\ \mathit{expr}) = \mathtt{0}$$

THEOREM: lr-p-c-size-list-add1-opener

$$\begin{array}{l}
((1 + n) < \text{length}\,(\mathit{expr})) \\
\to \quad (\text{lr-p-c-size-list}\,(1 + n,\ \mathit{expr}) \\
\qquad = \quad (\text{lr-p-c-size}\,(\mathtt{t},\ \text{cadr}\,(\mathit{expr})) + \text{lr-p-c-size-list}\,(n,\ \text{cdr}\,(\mathit{expr}))))
\end{array}$$

THEOREM: length-comp-body-1-lr-p-c-size
length (comp-body-1 (*flag*, *body*, *n*)) = lr-p-c-size (*flag*, *body*)

EVENT: Disable lr-p-c-size-list-add1-opener.

THEOREM: length-label-instrs
length (label-instrs (*instrs*, *n*)) = length (*instrs*)

THEOREM: length-comp-body-lr-p-c-size
length (comp-body (*body*)) = (1 + lr-p-c-size (**t**, *body*))

THEOREM: lr-p-c-size-flag-list
lr-p-c-size (`'list`, cdr (*expr*)) = lr-p-c-size-list (length (*expr*) − 1, *expr*)

THEOREM: lr-proper-exprp-car-if-cadr
(lr-proper-exprp (**t**, *body*, *pnames*, *formals*, *temps*, *table*)
 ∧ listp (*body*)
 ∧ (car (*body*) = `'if`))
→ lr-proper-exprp (**t**, cadr (*body*), *pnames*, *formals*, *temps*, *table*)

THEOREM: lr-proper-exprp-car-if-caddr
(lr-proper-exprp (**t**, *body*, *pnames*, *formals*, *temps*, *table*)
 ∧ listp (*body*)
 ∧ (car (*body*) = `'if`))
→ lr-proper-exprp (**t**, caddr (*body*), *pnames*, *formals*, *temps*, *table*)

THEOREM: lr-proper-exprp-car-if-cadddr
(lr-proper-exprp (**t**, *body*, *pnames*, *formals*, *temps*, *table*)
 ∧ listp (*body*)
 ∧ (car (*body*) = `'if`))
→ lr-proper-exprp (**t**, cadddr (*body*), *pnames*, *formals*, *temps*, *table*)

THEOREM: good-posp-list-t-offset-program-body
(good-posp (`'list`, offset (p-pc (*l*)), program-body (p-current-program (*l*)))
 ∧ listp (lr-expr-list (*l*))
 ∧ listp (offset (p-pc (*l*))))
→ good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))

THEOREM: good-posp-list-nx-offset-program-body
(good-posp (`'list`, offset (p-pc (*l*)), program-body (p-current-program (*l*)))
 ∧ listp (offset (p-pc (*l*)))
 ∧ listp (lr-expr-list (*l*)))
→ good-posp (`'list`,
  nx (offset (p-pc (*l*))),
  program-body (p-current-program (*l*)))

THEOREM: name-formal-vars-temp-var-dcls-program-body-cons

$$
\begin{array}{l}
(\text{name}(\text{cons}(\mathit{name},\ \mathit{rest})) = \mathit{name}) \\
\land\quad (\text{formal-vars}(\text{cons}(\mathit{name},\ \text{cons}(\mathit{formal\text{-}vars},\ \mathit{rest}))) = \mathit{formal\text{-}vars}) \\
\land\quad (\text{temp-var-dcls}(\text{cons}(\mathit{name}, \\
\qquad\qquad\qquad \text{cons}(\mathit{formal\text{-}vars},\ \text{cons}(\mathit{temp\text{-}var\text{-}dcls},\ \mathit{program\text{-}body})))) \\
\qquad = \quad \mathit{temp\text{-}var\text{-}dcls}) \\
\land\quad (\text{program-body}(\text{cons}(\mathit{name}, \\
\qquad\qquad\qquad \text{cons}(\mathit{formal\text{-}vars},\ \text{cons}(\mathit{temp\text{-}var\text{-}dcls},\ \mathit{program\text{-}body})))) \\
\qquad = \quad \mathit{program\text{-}body})
\end{array}
$$

THEOREM: program-body-assoc-comp-programs-1

$$
\begin{array}{l}
\text{definedp}(\mathit{name},\ \mathit{programs}) \\
\to\quad (\text{program-body}(\text{assoc}(\mathit{name},\ \text{comp-programs-1}(\mathit{programs}))) \\
\qquad = \quad \text{comp-body}(\text{program-body}(\text{assoc}(\mathit{name},\ \mathit{programs}))))
\end{array}
$$

THEOREM: program-body-assoc-comp-programs

$$
\begin{array}{l}
\text{definedp}(\mathit{name},\ \mathit{programs}) \\
\to\quad (\text{program-body}(\text{assoc}(\mathit{name},\ \text{comp-programs}(\mathit{programs}))) \\
\qquad = \quad \textbf{if}\ \mathit{name} = \text{name}(\text{car}(\mathit{programs})) \\
\qquad\qquad \textbf{then}\ \text{label-instrs}(\text{append}(\text{comp-body-1}(\textbf{t}, \\
\qquad\qquad\qquad\qquad\qquad \text{program-body}(\text{car}(\mathit{programs})), \\
\qquad\qquad\qquad\qquad\qquad \mathtt{0}), \\
\qquad\qquad\qquad\qquad \text{list}(\text{identity}(\text{list}(\mathtt{'set\text{-}global}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{area-name}(\textsc{lr-answer-addr}))), \\
\qquad\qquad\qquad\qquad\qquad \mathtt{'(ret)})), \\
\qquad\qquad\qquad \mathtt{0}) \\
\qquad\qquad \textbf{else}\ \text{comp-body}(\text{program-body}(\text{assoc}(\mathit{name},\ \text{cdr}(\mathit{programs})))) \ \textbf{endif})
\end{array}
$$

THEOREM: definedp-area-name-member-p-current-program

$$
\begin{array}{l}
\text{definedp}(\text{area-name}(\text{p-pc}(l)),\ \text{p-prog-segment}(l)) \\
\to\quad (\text{p-current-program}(l) \in \text{p-prog-segment}(l))
\end{array}
$$

DEFINITION:

$$
\begin{array}{l}
\text{induct-hint-6}(n,\ \mathit{body}) \\
= \quad \textbf{if}\ n < \text{length}(\mathit{body})\ \ \textbf{then}\ \text{induct-hint-6}(1 + n,\ \mathit{body}) \\
\qquad \textbf{else t endif}
\end{array}
$$

THEOREM: lr-p-c-size-list-0

$$
\begin{array}{l}
\text{listp}(\mathit{body}) \\
\to\quad ((\text{lr-p-c-size-list}(n,\ \mathit{body}) = \mathtt{0}) = ((n \simeq \mathtt{0}) \lor (\text{cdr}(\mathit{body}) \simeq \textbf{nil})))
\end{array}
$$

EVENT: Disable lr-p-c-size-list-0.

THEOREM: lessp-lr-p-c-size-list-lessp-sub1-length

$$(\text{lr-p-c-size-list}(n,\ \mathit{body}) \neq \mathtt{0})$$

$\rightarrow$ ((lr-p-c-size-list (n, *body*) $-$ 1)
$<$ lr-p-c-size-list (length (*body*) $-$ 1, *body*))

THEOREM: lr-p-pc-1-body-0
lr-p-pc-1 (0, *pos*) $=$ 0

THEOREM: lessp-lr-p-pc-1-lr-p-c-size-helper-1
(listp (*body*)
$\wedge$ ($n \not\simeq$ 0)
$\wedge$ (lr-p-pc-1 (get (n, *body*), *pos*) $<$ lr-p-c-size (**t**, get (n, *body*)))
$\wedge$ (lr-p-pc-1 (get (n, *body*), *pos*) $\neq$ 0))
$\rightarrow$ (((lr-p-c-size-list (n $-$ 1, *body*) $+$ lr-p-pc-1 (get (n, *body*), *pos*)) $-$ 1)
$<$ lr-p-c-size-list (length (*body*) $-$ 1, *body*))

THEOREM: lessp-lr-p-pc-1-lr-p-c-size
lr-p-pc-1 (*body*, *pos*) $<$ lr-p-c-size (**t**, *body*)

EVENT: Disable lessp-lr-p-pc-1-lr-p-c-size-helper-1.

THEOREM: not-lessp-p-max-temp-stk-size-lr-push-tstk
(p-psw (lr-push-tstk (l, *anything*)) $=$ `'run`)
$\rightarrow$ (p-max-temp-stk-size (l)
$\not<$ length (p-temp-stk (lr-push-tstk (l, *anything*))))

THEOREM: proper-p-temp-stkp-lr->p-lr-push-tstk
proper-p-temp-stkp (*temp-stkp*, lr->p (lr-push-tstk (l, *anything*)))
$=$ proper-p-temp-stkp (*temp-stkp*, lr->p (l))

THEOREM: proper-p-alistp-p-objectp
(proper-p-alistp (*bindings*, l) $\wedge$ definedp (*name*, *bindings*))
$\rightarrow$ p-objectp (cdr (assoc (*name*, *bindings*)), l)

THEOREM: formal-vars-assoc-comp-programs-1
definedp (*name*, *programs*)
$\rightarrow$ (formal-vars (assoc (*name*, comp-programs-1 (*programs*)))
$=$ formal-vars (assoc (*name*, *programs*)))

THEOREM: formal-vars-assoc-comp-programs
definedp (*name*, *programs*)
$\rightarrow$ (formal-vars (assoc (*name*, comp-programs (*programs*)))
$=$ formal-vars (assoc (*name*, *programs*)))

THEOREM: temp-var-dcls-assoc-comp-programs-1
definedp (*name*, *programs*)
$\rightarrow$ (temp-var-dcls (assoc (*name*, comp-programs-1 (*programs*)))
$=$ temp-var-dcls (assoc (*name*, *programs*)))

THEOREM: temp-var-dcls-assoc-comp-programs
definedp ($name$, $programs$)
$\rightarrow$ (temp-var-dcls (assoc ($name$, comp-programs ($programs$)))
  $=$ temp-var-dcls (assoc ($name$, $programs$)))

THEOREM: lr-programs-properp-definedp-car-untag-p-pc
lr-programs-properp (l, $table$)
$\rightarrow$ definedp (car (untag (p-pc (l))), p-prog-segment (l))

THEOREM: p-objectp-cdr-assoc-litatom-proper-p-alistp
(proper-p-alistp ($bindings$, lp)
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, $table$)
$\land$ (strip-cars ($bindings$)
  $=$ append (formal-vars (assoc (car (untag (p-pc (l))),
      comp-programs-1 (p-prog-segment (l)))),
    strip-cars (temp-var-dcls (assoc (car (untag (p-pc (l))),
      comp-programs-1 (p-prog-segment (l)))))))
$\land$ litatom (lr-expr (l)))
$\rightarrow$ p-objectp (cdr (assoc (lr-expr (l), $bindings$)), lp)

THEOREM: proper-p-temp-stkp-lr-push-tstk-assoc-bindings
(lr-programs-properp (l, $table$)
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ proper-p-alistp (bindings (car (p-ctrl-stk (l))), lr->p (l))
$\land$ litatom (lr-expr (l))
$\land$ (strip-cars (bindings (car (p-ctrl-stk (l))))
  $=$ append (formal-vars (assoc (car (untag (p-pc (l))),
      comp-programs-1 (p-prog-segment (l)))),
    strip-cars (temp-var-dcls (assoc (car (untag (p-pc (l))),
      comp-programs-1 (p-prog-segment (l))))))))
$\rightarrow$ (proper-p-temp-stkp (p-temp-stk (lr-push-tstk (l,
      cdr (assoc (lr-expr (l),
        bindings (car (p-ctrl-stk (l))))))),
    lr->p (l))
  $=$ proper-p-temp-stkp (p-temp-stk (l), lr->p (l)))

THEOREM: lr-p-pc-lr-push-tstk
lr-p-pc (lr-push-tstk (l, $anything$)) $=$ lr-p-pc (l)

THEOREM: proper-p-statep-lr->p-lr-push-tstk
(proper-p-statep (lr->p (l))
$\land$ lr-programs-properp (l, $table$)
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ (p-psw (lr-push-tstk (l,

$$
\begin{array}{lll}
 & & \quad\quad\quad\quad\text{cdr}\,(\text{assoc}\,(\text{lr-expr}\,(l), \\
 & & \quad\quad\quad\quad\quad\quad\quad\quad\text{bindings}\,(\text{car}\,(\text{p-ctrl-stk}\,(l)))))) \\
 & \quad = \quad \texttt{'run}) & \\
\wedge & \text{litatom}\,(\text{lr-expr}\,(l))) & \\
\rightarrow & \text{proper-p-statep}\,(\text{lr->p}\,(\text{lr-push-tstk}\,(l, & \\
 & \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\text{cdr}\,(\text{assoc}\,(\text{lr-expr}\,(l), & \\
 & \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\text{bindings}\,(\text{car}\,(\text{p-ctrl-stk}\,(l)))))))) &
\end{array}
$$

THEOREM: good-posp1-cons-lessp-4-if-lr-proper-exprp

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{car}\,(\text{cur-expr}\,(pos,\ body)) = \texttt{'if})} \\
\wedge & \text{good-posp1}\,(pos,\ body) \\
\wedge & \text{lr-proper-exprp}\,(\texttt{t},\ body,\ pnames,\ formals,\ temps,\ table)) \\
\rightarrow & (\text{good-posp1}\,(\text{dv}\,(pos,\ \texttt{1}),\ body) \\
 & \quad\wedge\quad \text{good-posp1}\,(\text{dv}\,(pos,\ \texttt{2}),\ body) \\
 & \quad\wedge\quad \text{good-posp1}\,(\text{dv}\,(pos,\ \texttt{3}),\ body))
\end{array}
$$

THEOREM: good-posp-cons-lessp-4-if-lr-programs-properp

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{car}\,(\text{lr-expr}\,(l)) = \texttt{'if})} \\
\wedge & \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
\wedge & \text{lr-programs-properp}\,(l,\ table)) \\
\rightarrow & (\text{good-posp1}\,(\text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \texttt{1}),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
 & \quad\wedge\quad \text{good-posp1}\,(\text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \texttt{2}), \\
 & \quad\quad\quad\quad\quad\quad\quad \text{program-body}\,(\text{p-current-program}\,(l))) \\
 & \quad\wedge\quad \text{good-posp1}\,(\text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \texttt{3}), \\
 & \quad\quad\quad\quad\quad\quad\quad \text{program-body}\,(\text{p-current-program}\,(l))))
\end{array}
$$

THEOREM: proper-p-statep-lr->p-lr-set-pos

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-programs-properp}\,(l,\ table) \wedge \text{proper-p-statep}\,(\text{lr->p}\,(l)))} \\
\rightarrow & \text{proper-p-statep}\,(\text{lr->p}\,(\text{lr-set-pos}\,(l,\ pos)))
\end{array}
$$

THEOREM: lr-p-pc-lr-pop-tstk

$$\text{lr-p-pc}\,(\text{lr-pop-tstk}\,(l)) = \text{lr-p-pc}\,(l)$$

THEOREM: proper-p-statep-lr->p-lr-pop-tstk

$$\text{proper-p-statep}\,(\text{lr->p}\,(l)) \rightarrow \text{proper-p-statep}\,(\text{lr->p}\,(\text{lr-pop-tstk}\,(l)))$$

THEOREM: good-posp-dv-1-temps-lr-expr

$$
\begin{array}{ll}
\multicolumn{2}{l}{(((\text{car}\,(\text{lr-expr}\,(l)) = \textsc{s-temp-eval}) \vee (\text{car}\,(\text{lr-expr}\,(l)) = \textsc{s-temp-test}))} \\
\wedge & \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l)))) \\
\rightarrow & \text{good-posp1}\,(\text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \texttt{1}),\ \text{program-body}\,(\text{p-current-program}\,(l)))
\end{array}
$$

THEOREM: proper-p-alistp-put-assoc

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{proper-p-alistp}\,(bindings,\ l) \wedge \text{p-objectp}\,(object,\ l))} \\
\rightarrow & \text{proper-p-alistp}\,(\text{put-assoc}\,(object,\ var\text{-}name,\ bindings),\ l)
\end{array}
$$

THEOREM: listp-p-temp-stk-lr-push-tstk
(p-psw (lr-push-tstk (l, *object*)) = `'run`)
$\rightarrow$ listp (p-temp-stk (lr-push-tstk (l, *object*)))

THEOREM: lr-p-pc-lr-set-temp
lr-p-pc (lr-set-temp (l, *value*, *var-name*)) = lr-p-pc (l)

THEOREM: proper-p-statep-lr-set-temp
(proper-p-statep (lr->p (l)) $\wedge$ listp (p-temp-stk (l)))
$\rightarrow$ proper-p-statep (lr->p (lr-set-temp (l, car (p-temp-stk (l)), *var-name*)))

THEOREM: p-objectp-cdr-assoc-bindings-proper-p-alistp
(proper-p-alistp (*bindings*, l) $\wedge$ definedp (*object*, *bindings*))
$\rightarrow$ p-objectp (cdr (assoc (*object*, *bindings*)), l)

THEOREM: definedp-caddr-lr-expr-bindings-ctrl-stk
(lr-programs-properp-1 (*progs*, *program-names*, *table*)
$\wedge$ definedp (*name*, *progs*)
$\wedge$ ((car (cur-expr (*pos*, program-body (assoc (*name*, *progs*))))
= S-TEMP-FETCH)
$\vee$ (car (cur-expr (*pos*, program-body (assoc (*name*, *progs*))))
= S-TEMP-TEST))
$\wedge$ good-posp1 (*pos*, program-body (assoc (*name*, *progs*)))
$\wedge$ (strip-cars (*bindings*)
= append (formal-vars (assoc (*name*, comp-programs (*progs*))),
strip-cars (temp-var-dcls (assoc (*name*,
comp-programs (*progs*)))))))
$\rightarrow$ definedp (caddr (cur-expr (*pos*, program-body (assoc (*name*, *progs*)))),
bindings)

THEOREM: proper-p-temp-stkp-p-temp-stk-lr-do-temp-fetch
(proper-p-framep (top (p-ctrl-stk (*l1*)), area-name (p-pc (*l1*)), *l2*)
$\wedge$ lr-programs-properp (*l1*, *table*)
$\wedge$ ((car (lr-expr (*l1*)) = S-TEMP-FETCH)
$\vee$ (car (lr-expr (*l1*)) = S-TEMP-TEST))
$\wedge$ good-posp1 (offset (p-pc (*l1*)), program-body (p-current-program (*l1*)))
$\wedge$ same-signature (p-data-segment (*l1*), p-data-segment (*l2*))
$\wedge$ (p-prog-segment (lr->p (*l1*)) = p-prog-segment (*l2*))
$\wedge$ (p-word-size (*l1*) = p-word-size (*l2*)))
$\rightarrow$ (proper-p-temp-stkp (p-temp-stk (lr-do-temp-fetch (*l1*)), *l2*)
= proper-p-temp-stkp (p-temp-stk (*l1*), *l2*))

THEOREM: length-lr-do-temp-fetch
(p-psw (lr-do-temp-fetch (l)) = `'run`)
$\rightarrow$ (p-max-temp-stk-size (l) $\not<$ length (p-temp-stk (lr-do-temp-fetch (l))))

THEOREM: lr-p-pc-lr-do-temp-fetch
$\text{lr-p-pc}\,(\text{lr-do-temp-fetch}\,(l)) = \text{lr-p-pc}\,(l)$

THEOREM: proper-p-statep-lr-do-temp-fetch
$((\text{p-psw}\,(\text{lr-do-temp-fetch}\,(l)) = \texttt{'run})$
$\land\ \ \text{lr-programs-properp}\,(l,\ \mathit{table})$
$\land\ \ ((\text{car}\,(\text{lr-expr}\,(l)) = \text{S-TEMP-FETCH})$
$\qquad \lor\ \ (\text{car}\,(\text{lr-expr}\,(l)) = \text{S-TEMP-TEST}))$
$\land\ \ \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l)))$
$\land\ \ \text{proper-p-statep}\,(\text{lr->p}\,(l)))$
$\to\ \ \text{proper-p-statep}\,(\text{lr->p}\,(\text{lr-do-temp-fetch}\,(l)))$

THEOREM: length-lr-push-tstk
$(\text{p-psw}\,(\text{lr-push-tstk}\,(l,\ \mathit{object})) = \texttt{'run})$
$\to\ \ (\text{p-max-temp-stk-size}\,(l)$
$\qquad \not< \ \ \text{length}\,(\text{p-temp-stk}\,(\text{lr-push-tstk}\,(l,\ \mathit{object}))))$

THEOREM: listp-p-temp-stk-lr-do-temp-fetch
$(\text{p-psw}\,(\text{lr-do-temp-fetch}\,(l)) = \texttt{'run})$
$\to\ \ \text{listp}\,(\text{p-temp-stk}\,(\text{lr-do-temp-fetch}\,(l)))$

THEOREM: proper-p-prog-segmentp-append
$\text{plistp}\,(\mathit{segment1})$
$\to\ \ (\text{proper-p-prog-segmentp}\,(\text{append}\,(\mathit{segment1},\ \mathit{segment2}),\ p)$
$\qquad =\ \ (\text{proper-p-prog-segmentp}\,(\mathit{segment1},\ p)$
$\qquad\qquad \land\ \ \text{proper-p-prog-segmentp}\,(\mathit{segment2},\ p)))$

THEOREM: lr-programs-properp-expr-quote-type-addr
$(\text{lr-programs-properp}\,(l,\ \mathit{table})$
$\land\ \ \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l)))$
$\land\ \ \text{listp}\,(\text{lr-expr}\,(l))$
$\land\ \ (\text{car}\,(\text{lr-expr}\,(l)) = \texttt{'quote}))$
$\to\ \ (\text{type}\,(\text{cadr}\,(\text{lr-expr}\,(l))) = \texttt{'addr})$

THEOREM: proper-p-instructionp-push-constant-opener
$\text{proper-p-instructionp}\,(\text{list}\,(\texttt{'push-constant},\ \mathit{object}),\ \mathit{name},\ p)$
$=\ \ \text{proper-p-push-constant-instructionp}\,(\text{list}\,(\texttt{'push-constant},\ \mathit{object}),$
$\qquad \mathit{name},$
$\qquad p)$

THEOREM: proper-labeled-p-instructionsp-find-labelp-non-litatom
$(\text{proper-labeled-p-instructionsp}\,(\mathit{body},\ \mathit{name},\ p) \land (\neg\ \text{litatom}\,(\mathit{label})))$
$\to\ \ (\text{find-labelp}\,(\mathit{label},\ \mathit{body}) = \mathbf{f})$

THEOREM: lessp-4-not-zerop-not-1-not-2-3
$((n \not\simeq \texttt{0}) \land (n \neq \texttt{1}) \land (n \neq \texttt{2}) \land (n < \texttt{4})) \to (n = \texttt{3})$

THEOREM: lessp-4-not-zerop-not-1-not-2-3-get-car-pos

$$((\text{car}(pos) \not\simeq 0) \land (\text{car}(pos) \neq 1) \land (\text{car}(pos) \neq 2) \land (\text{car}(pos) < 4)) \rightarrow (\text{get}(\text{car}(pos), body) = \text{cadddr}(body))$$

EVENT: Disable lessp-4-not-zerop-not-1-not-2-3-get-car-pos.

THEOREM: lessp-index-lessp-lr-p-c-size-list

$$\text{lr-p-c-size-list}(\text{length}(\text{cdr}(body)), body) \not< \text{lr-p-c-size-list}(n, body)$$

THEOREM: lessp-plus-lr-p-c-size-lr-p-pc-1-helper

$$\begin{aligned}&(\text{listp}(body)\\ &\land\ (n \not\simeq 0)\\ &\land\ (\text{lr-p-c-size}(\mathbf{t}, \text{get}(n, body)) \not< x)\\ &\land\ ((n - 1) < \text{length}(\text{cdr}(body)))\\ &\land\ (len = \text{length}(\text{cdr}(body))))\\ &\rightarrow\ (((\text{lr-p-c-size-list}(len, body) + 1)\\ &\qquad < (\text{lr-p-c-size-list}(n - 1, body) + x))\\ &\qquad = \mathbf{f})\end{aligned}$$

THEOREM: lessp-plus-lr-p-c-size-lr-p-pc-1

$$\begin{aligned}&(\text{good-posp1}(pos, body) \land \text{lr-proper-exprp}(\mathbf{t}, body, pnames, formals, temps, table))\\ &\rightarrow\ (\text{lr-p-c-size}(\mathbf{t}, body)\\ &\qquad \not< (\text{lr-p-pc-1}(body, pos) + \text{lr-p-c-size}(\mathbf{t}, \text{cur-expr}(pos, body))))\end{aligned}$$

DEFINITION:

induct-hint-7 (*pos*, *expr*, *n*)

= **if** *pos* ≃ **nil** **then t**
  elseif *expr* ≃ **nil** **then t**
  elseif car (*expr*) = 'if
  then let *then-n* **be** *n* + 3 + lr-p-c-size (**t**, cadr (*expr*))
    in
    case on car (*pos*):
    case = 1
    then induct-hint-7 (cdr (*pos*), cadr (*expr*), *n*)
    case = 2
     then induct-hint-7 (cdr (*pos*), caddr (*expr*), *then-n*)
    otherwise induct-hint-7 (cdr (*pos*),
      cadddr (*expr*),
      1
      + *then-n*
      + lr-p-c-size (**t**, caddr (*expr*))) **endcase endlet**

elseif car ($expr$) = S-TEMP-FETCH **then t**
elseif car ($expr$) = S-TEMP-EVAL
then induct-hint-7 (cdr (pos), cadr ($expr$), n)
elseif car ($expr$) = S-TEMP-TEST
then induct-hint-7 (cdr (pos), cadr ($expr$), n + 4)
elseif car ($expr$) = `'quote` **then t**
else induct-hint-7 (cdr (pos),
get (car (pos), $expr$),
n + lr-p-c-size-list (car (pos) − 1, $expr$)) **endif**

THEOREM: lr-p-c-size-s-temp-test-eval-cadr-not-lessp-fact
(listp ($expr$)
∧ ((car ($expr$) = S-TEMP-EVAL) ∨ (car ($expr$) = S-TEMP-TEST)))
→ (lr-p-c-size (**t**, cadr ($expr$)) < lr-p-c-size (**t**, $expr$))

THEOREM: length-comp-temp-test
(listp ($body$) ∧ (car ($body$) = S-TEMP-TEST))
→ (length (comp-temp-test (any-$body$, comp-body-1 (**t**, cadr ($body$), n), any-n))
= lr-p-c-size (**t**, $body$))

THEOREM: plistp-comp-temp-test
plistp (comp-temp-test ($body$, $instrs$, n))

THEOREM: length-comp-if-alt
(listp ($body$) ∧ (car ($body$) = `'if`))
→ (length (comp-if (comp-body-1 (**t**, cadr ($body$), $n1$),
comp-body-1 (**t**, caddr ($body$), $n2$),
comp-body-1 (**t**, cadddr ($body$), $n3$),
any-n))
= lr-p-c-size (**t**, $body$))

THEOREM: plistp-comp-if
(plistp ($else$-$instrs$) ∧ listp ($else$-$instrs$))
→ plistp (comp-if ($test$-$intrs$, $then$-$instrs$, $else$-$instrs$, n))

THEOREM: plistp-comp-body-1
plistp (comp-body-1 ($flag$, $body$, n))

THEOREM: lr-p-c-size-list-funcall-not-lessp-fact
(listp ($expr$)
∧ (car ($expr$) ≠ S-TEMP-FETCH)
∧ (car ($expr$) ≠ S-TEMP-EVAL)
∧ (car ($expr$) ≠ S-TEMP-TEST)
∧ (car ($expr$) ≠ `'quote`)
∧ (car ($expr$) ≠ `'if`))
→ (lr-p-c-size-list (length ($expr$) − 1, $expr$) < lr-p-c-size (**t**, $expr$))

THEOREM: lr-p-c-size-nlistp-body
$(\neg\ \text{listp}\,(\mathit{body})) \rightarrow (\text{lr-p-c-size}\,(\mathbf{t},\ \mathit{body}) = \mathtt{1})$

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-1

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{good-posp1}\,(\mathit{pos},\ \text{cadr}\,(\mathit{body}))} \\
\wedge & (\text{car}\,(\mathit{body}) = \mathtt{'if}) \\
\wedge & \text{listp}\,(\mathit{body}) \\
\wedge & \text{lr-proper-exprp}\,(\mathbf{t},\ \mathit{body},\ \mathit{pnames},\ \mathit{formals},\ \mathit{temps},\ \mathit{table})) \\
\rightarrow & (\text{firstn}\,(\text{lr-p-c-size}\,(\mathbf{t},\ \text{cur-expr}\,(\mathit{pos},\ \text{cadr}\,(\mathit{body}))), \\
& \qquad\quad \text{restn}\,(\text{lr-p-pc-1}\,(\text{cadr}\,(\mathit{body}),\ \mathit{pos}), \\
& \qquad\qquad\quad \text{comp-if}\,(\text{comp-body-1}\,(\mathbf{t},\ \text{cadr}\,(\mathit{body}),\ n), \\
& \qquad\qquad\qquad\qquad \mathit{then\text{-}instrs}, \\
& \qquad\qquad\qquad\qquad \mathit{else\text{-}instrs}, \\
& \qquad\qquad\qquad\qquad n))) \\
& \quad = \quad \text{firstn}\,(\text{lr-p-c-size}\,(\mathbf{t},\ \text{cur-expr}\,(\mathit{pos},\ \text{cadr}\,(\mathit{body}))), \\
& \qquad\qquad\quad \text{restn}\,(\text{lr-p-pc-1}\,(\text{cadr}\,(\mathit{body}),\ \mathit{pos}), \\
& \qquad\qquad\qquad \text{comp-body-1}\,(\mathbf{t},\ \text{cadr}\,(\mathit{body}),\ n))))
\end{array}
$$

THEOREM: firstn-restn-plus-comp-if-1

$$
\begin{array}{ll}
\multicolumn{2}{l}{((j = \text{length}\,(\mathit{test}))} \\
\wedge & \text{listp}\,(\mathit{then}) \\
\wedge & (m < \text{length}\,(\mathit{then})) \\
\wedge & (m \in \mathbf{N}) \\
\wedge & \text{listp}\,(\mathit{test}) \\
\wedge & (\text{length}\,(\mathit{then}) \not< (k + m))) \\
\rightarrow & (\text{firstn}\,(k,\ \text{restn}\,(\mathtt{3} + j + m,\ \text{comp-if}\,(\mathit{test},\ \mathit{then},\ \mathit{else},\ n))) \\
& \quad = \quad \text{firstn}\,(k,\ \text{restn}\,(m,\ \mathit{then})))
\end{array}
$$

THEOREM: firstn-unlabel-instrs-comp-body-1-lr-p-pc-1-helper-2

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{good-posp1}\,(\mathit{pos},\ \text{caddr}\,(\mathit{body}))} \\
\wedge & (\text{car}\,(\mathit{body}) = \mathtt{'if}) \\
\wedge & \text{listp}\,(\mathit{body}) \\
\wedge & \text{lr-proper-exprp}\,(\mathbf{t},\ \mathit{body},\ \mathit{pnames},\ \mathit{formals},\ \mathit{temps},\ \mathit{table})) \\
\rightarrow & (\text{lr-p-c-size}\,(\mathbf{t},\ \text{caddr}\,(\mathit{body})) \\
& \quad \not< \quad (\text{lr-p-pc-1}\,(\text{caddr}\,(\mathit{body}),\ \mathit{pos}) \\
& \qquad\quad + \quad \text{lr-p-c-size}\,(\mathbf{t},\ \text{cur-expr}\,(\mathit{pos},\ \text{caddr}\,(\mathit{body})))))
\end{array}
$$

THEOREM: firstn-restn-plus-comp-if-2

$$
\begin{array}{ll}
\multicolumn{2}{l}{((j = \text{length}\,(\mathit{test}))} \\
\wedge & (i = \text{length}\,(\mathit{then})) \\
\wedge & \text{listp}\,(\mathit{then}) \\
\wedge & (m < \text{length}\,(\mathit{else})) \\
\wedge & (m \in \mathbf{N}) \\
\wedge & \text{listp}\,(\mathit{test}) \\
\wedge & \text{listp}\,(\mathit{else})
\end{array}
$$

$\wedge$ (length (*else*) $\not<$ $(m + k)$))
$\rightarrow$ (firstn (k, restn ($j + i + m + \mathtt{4}$, comp-if (*test*, *then*, *else*, n)))
= firstn (k, restn (m, *else*)))

THEOREM: firstn-unlabel-instrs-comp-body-1-lr-p-pc-1-helper-3
(good-posp1 (*pos*, cadddr (*body*))
$\wedge$ (car (*body*) = `'if`)
$\wedge$ listp (*body*)
$\wedge$ lr-proper-exprp (**t**, *body*, *pnames*, *formals*, *temps*, *table*))
$\rightarrow$ (lr-p-c-size (**t**, cadddr (*body*))
$\not<$ (lr-p-pc-1 (cadddr (*body*), *pos*)
+ lr-p-c-size (**t**, cur-expr (*pos*, cadddr (*body*)))))

THEOREM: plus-constant-fact-helper-1
$(\mathtt{1} + n + \mathtt{3} + x + y) = (n + \mathtt{4} + x + y)$

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-4
(good-posp1 (*pos*, cadr (*body*))
$\wedge$ listp (*body*)
$\wedge$ (car (*body*) = S-TEMP-TEST)
$\wedge$ lr-proper-exprp (**t**, *body*, *pnames*, *formals*, *temps*, *table*))
$\rightarrow$ (firstn (lr-p-c-size (**t**, cur-expr (*pos*, cadr (*body*))),
restn (lr-p-pc-1 (cadr (*body*), *pos*) + `4`,
comp-temp-test (*body-1*, comp-body-1 (**t**, cadr (*body*), n), m)))
= firstn (lr-p-c-size (**t**, cur-expr (*pos*, cadr (*body*))),
restn (lr-p-pc-1 (cadr (*body*), *pos*),
comp-body-1 (**t**, cadr (*body*), n))))

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-5
$(\mathtt{4} + x) = (x + \mathtt{4})$

THEOREM: good-posp1-lr-proper-exprp-get-cadddr
(listp (*pos*)
$\wedge$ listp (*body*)
$\wedge$ (car (*body*) = `'if`)
$\wedge$ (car (*pos*) $\neq$ `1`)
$\wedge$ (car (*pos*) $\neq$ `2`)
$\wedge$ (car (*pos*) $\neq$ `0`)
$\wedge$ (car (*pos*) $\in \mathbf{N}$)
$\wedge$ lr-proper-exprp (**t**, *body*, *pnames*, *formals*, *temps*, *table*)
$\wedge$ ((((car (*pos*) $-$ 1) $-$ 1) $-$ 1) $<$ length (cdddr (*body*))))
$\rightarrow$ (get (car (*pos*), *body*) = cadddr (*body*))

THEOREM: lr-proper-exprp-cadr-temps
(lr-proper-exprp (**t**, *expr*, *pnames*, *formals*, *temps*, *table*)
$\wedge$ ((car (*expr*) = S-TEMP-EVAL) $\vee$ (car (*expr*) = S-TEMP-TEST)))
$\rightarrow$ lr-proper-exprp (**t**, cadr (*expr*), *pnames*, *formals*, *temps*, *table*)

THEOREM: lessp-plus-lr-p-c-size-lr-p-pc-1-temps

$$
\begin{array}{l}
(\text{good-posp1}\,(pos,\ \text{cadr}\,(body))\\
\ \wedge\quad \text{listp}\,(body)\\
\ \wedge\quad ((\text{car}\,(body) = \text{S-TEMP-EVAL}) \vee (\text{car}\,(body) = \text{S-TEMP-TEST}))\\
\ \wedge\quad \text{lr-proper-exprp}\,(\mathbf{t},\ body,\ pnames,\ formals,\ temps,\ table))\\
\rightarrow\quad (\text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(body))\\
\qquad\quad \not< \quad (\text{lr-p-pc-1}\,(\text{cadr}\,(body),\ pos)\\
\qquad\qquad\quad + \quad \text{lr-p-c-size}\,(\mathbf{t},\ \text{cur-expr}\,(pos,\ \text{cadr}\,(body)))))
\end{array}
$$

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-6

$$
\begin{array}{l}
(\text{listp}\,(body)\\
\ \wedge\quad (\text{car}\,(body) \neq \texttt{'if})\\
\ \wedge\quad (\text{car}\,(body) \neq \text{S-TEMP-FETCH})\\
\ \wedge\quad (\text{car}\,(body) \neq \text{S-TEMP-EVAL})\\
\ \wedge\quad (\text{car}\,(body) \neq \text{S-TEMP-TEST})\\
\ \wedge\quad (\text{car}\,(body) \neq \texttt{'quote})\\
\ \wedge\quad \text{lr-proper-exprp}\,(\mathbf{t},\ body,\ pnames,\ formals,\ temps,\ table)\\
\ \wedge\quad (n \not\simeq \texttt{0}))\\
\rightarrow\quad (\text{lr-p-c-size-list}\,(\text{length}\,(body) - 1,\ body)\\
\qquad\quad \not< \quad (\text{lr-p-c-size-list}\,(n - 1,\ body) + \text{lr-p-pc-1}\,(\text{get}\,(n,\ body),\ pos)))
\end{array}
$$

DEFINITION:

$$
\begin{array}{l}
\text{induct-hint-10}\,(n,\ l,\ x)\\
= \quad \textbf{if}\ \neg\ \text{listp}\,(l)\ \ \textbf{then}\ \mathbf{t}\\
\qquad \textbf{elseif}\ n \simeq \texttt{0}\ \ \textbf{then}\ \mathbf{t}\\
\qquad \textbf{elseif}\ \text{listp}\,(\text{cdr}\,(l))\\
\qquad \textbf{then}\ \text{induct-hint-10}\,(n - 1,\ \text{cdr}\,(l),\ x + \text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(l)))\\
\qquad \textbf{else}\ \mathbf{t}\ \textbf{endif}
\end{array}
$$

THEOREM: lr-p-c-size-list-car-opener

$$
\begin{array}{l}
((n \not\simeq \texttt{0}) \wedge (n < \text{length}\,(body)))\\
\rightarrow\quad (\text{lr-p-c-size-list}\,(n,\ body)\\
\qquad\quad = \quad (\text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(body))\\
\qquad\qquad\quad + \quad \text{lr-p-c-size-list}\,(n - 1,\ \text{cdr}\,(body))))
\end{array}
$$

THEOREM: restn-comp-body-1-list-fact

$$
\begin{array}{l}
((\text{lr-p-c-size}\,(\mathbf{t},\ \text{get}\,(m,\ \text{cdr}\,(body))) \not< j)\\
\ \wedge\quad (m < \text{length}\,(\text{cdr}\,(body)))\\
\ \wedge\quad (m \in \mathbf{N})\\
\ \wedge\quad (n \in \mathbf{N})\\
\ \wedge\quad (j \in \mathbf{N}))\\
\rightarrow\quad (\text{restn}\,(\text{lr-p-c-size-list}\,(m,\ body) + j,\\
\qquad\qquad\quad \text{comp-body-1}\,(\texttt{'list},\ \text{cdr}\,(body),\ n))\\
\qquad\quad = \quad \text{restn}\,(j,\\
\qquad\qquad\qquad\quad \text{comp-body-1}\,(\texttt{'list},
\end{array}
$$

$$\begin{array}{l}
\qquad\qquad\qquad \text{restn}\,(m,\ \text{cdr}\,(\mathit{body})),\\
\qquad\qquad\qquad n + \text{lr-p-c-size-list}\,(m,\ \mathit{body}))))
\end{array}$$

EVENT: Disable lr-p-c-size-list-car-opener.

THEOREM: firstn-restn-small-enough-cdr-comp-body-1-list

$$\begin{array}{l}
(\text{listp}\,(\mathit{body}) \land (\text{lr-p-c-size}\,(\mathbf{t},\ \text{car}\,(\mathit{body})) \not< (j + k)))\\
\rightarrow\ \ (\text{firstn}\,(j,\ \text{restn}\,(k,\ \text{comp-body-1}\,(\texttt{'list},\ \mathit{body},\ n)))\\
\qquad =\ \ \text{firstn}\,(j,\ \text{restn}\,(k,\ \text{comp-body-1}\,(\mathbf{t},\ \text{car}\,(\mathit{body}),\ n))))
\end{array}$$

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-7

$$\begin{array}{l}
(\text{listp}\,(\mathit{body})\\
\ \land\ \ (\text{car}\,(\mathit{body}) \neq \texttt{'if})\\
\ \land\ \ (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-FETCH})\\
\ \land\ \ (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-EVAL})\\
\ \land\ \ (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-TEST})\\
\ \land\ \ (\text{car}\,(\mathit{body}) \neq \texttt{'quote})\\
\ \land\ \ (n \in \mathbf{N})\\
\ \land\ \ \text{lr-proper-exprp}\,(\mathbf{t},\ \mathit{body},\ \mathit{pnames},\ \mathit{formals},\ \mathit{temps},\ \mathit{table})\\
\ \land\ \ (m \not\simeq \texttt{0})\\
\ \land\ \ (m < \text{length}\,(\mathit{body}))\\
\ \land\ \ \text{good-posp1}\,(\mathit{pos},\ \text{get}\,(m,\ \mathit{body})))\\
\rightarrow\ \ (\text{firstn}\,(\text{lr-p-c-size}\,(\mathbf{t},\ \text{cur-expr}\,(\mathit{pos},\ \text{get}\,(m,\ \mathit{body}))),\\
\qquad\qquad \text{restn}\,(\text{lr-p-c-size-list}\,(m - 1,\ \mathit{body})\\
\qquad\qquad\qquad +\ \ \text{lr-p-pc-1}\,(\text{get}\,(m,\ \mathit{body}),\ \mathit{pos}),\\
\qquad\qquad\qquad \text{comp-body-1}\,(\texttt{'list},\ \text{cdr}\,(\mathit{body}),\ n)))\\
\qquad =\ \ \text{firstn}\,(\text{lr-p-c-size}\,(\mathbf{t},\ \text{cur-expr}\,(\mathit{pos},\ \text{get}\,(m,\ \mathit{body}))),\\
\qquad\qquad\quad \text{restn}\,(\text{lr-p-pc-1}\,(\text{get}\,(m,\ \mathit{body}),\ \mathit{pos}),\\
\qquad\qquad\qquad \text{comp-body-1}\,(\mathbf{t},\\
\qquad\qquad\qquad\qquad \text{get}\,(m,\ \mathit{body}),\\
\qquad\qquad\qquad\qquad n + \text{lr-p-c-size-list}\,(m - 1,\ \mathit{body})))))
\end{array}$$

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-8

$$\begin{array}{l}
(\text{listp}\,(\mathit{body})\\
\ \land\ \ (\text{car}\,(\mathit{body}) \neq \texttt{'if})\\
\ \land\ \ (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-FETCH})\\
\ \land\ \ (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-EVAL})\\
\ \land\ \ (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-TEST})\\
\ \land\ \ (\text{car}\,(\mathit{body}) \neq \texttt{'quote})\\
\ \land\ \ \text{lr-proper-exprp}\,(\mathbf{t},\ \mathit{body},\ \mathit{pnames},\ \mathit{formals},\ \mathit{temps},\ \mathit{table})\\
\ \land\ \ (n \not\simeq \texttt{0})\\
\ \land\ \ (n < \text{length}\,(\mathit{body}))\\
\ \land\ \ \text{good-posp1}\,(\mathit{pos},\ \text{get}\,(n,\ \mathit{body})))\\
\rightarrow\ \ ((\text{lr-p-c-size-list}\,(\text{length}\,(\mathit{body}) - 1,\ \mathit{body})
\end{array}$$

$-$ (lr-p-c-size-list ($n - 1$, *body*) + lr-p-pc-1 (get (n, *body*), *pos*)))
$\not<$ lr-p-c-size (**t**, cur-expr (*pos*, get (n, *body*))))

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1
(good-posp1 (*pos*, *body*)
$\land$ lr-proper-exprp (**t**, *body*, *pnames*, *formals*, *temps*, *table*)
$\land$ ($n \in \mathbf{N}$))
$\rightarrow$ (firstn (lr-p-c-size (**t**, cur-expr (*pos*, *body*)),
restn (lr-p-pc-1 (*body*, *pos*), comp-body-1 (**t**, *body*, n)))
$=$ comp-body-1 (**t**, cur-expr (*pos*, *body*), n + lr-p-pc-1 (*body*, *pos*)))

EVENT: Disable firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-5.

THEOREM: not-lessp-lr-p-c-size-flag-t-1
lr-p-c-size (**t**, *body1*) $\not<$ `1`

THEOREM: not-lessp-x-x
$(x < x) =$ **f**

THEOREM: get-plus
get ($x + y$, *list*) = get (y, restn (x, *list*))

EVENT: Disable get-plus.

THEOREM: get-firstn-different-lists
(($k < n$) $\land$ (firstn (n, *list1*) = firstn (n, *list2*)))
$\rightarrow$ (get (k, *list1*) = get (k, *list2*))

THEOREM: unlabel-list-label
unlabel (list (`'dl`, *lab*, *comment*, *instr*)) = *instr*

THEOREM: legal-labelp-label-make-label
legal-labelp (list (`'dl`, lr-make-label (n), *comment*, *instr*))

THEOREM: lr-make-label-not-numberp
($n \notin \mathbf{N}$) $\rightarrow$ (lr-make-label (n) = lr-make-label (`0`))

DEFINITION:
induct-hint-9 (m, *instrs*, n)
= **if** listp (*instrs*) **then** induct-hint-9 ($m - 1$, cdr (*instrs*), $1 + n$)
else t endif

THEOREM: get-label-instrs
($m <$ length (*instrs*))
$\rightarrow$ (get (m, label-instrs (*instrs*, n))
$=$ list (`'dl`, lr-make-label ($n + m$), **nil**, get (m, *instrs*)))

Event: Disable lr-make-label-not-numberp.

Theorem: get-append
get $(n,$ append $(x,\ y))$
$=$ **if** $n <$ length (x) **then** get $(n,\ x)$
else get $(n -$ length $(x),\ y)$ **endif**

Event: Disable get-append.

Theorem: get-lr-p-c-size-lessp-lr-p-c-size-comp-body-1
(good-posp1 (*pos*, *body*)
$\wedge$ lr-proper-exprp (**t**, *body*, *pnames*, *formals*, *temps*, *table*)
$\wedge$ $(n \in \mathbf{N})$
$\wedge$ ($m <$ lr-p-c-size (**t**, cur-expr (*pos*, *body*))))
$\rightarrow$ (get (lr-p-pc-1 (*body*, *pos*) $+\ m$, comp-body-1 (**t**, *body*, n))
$=$ get (m,
comp-body-1 (**t**,
cur-expr (*pos*, *body*),
$n\ +$ lr-p-pc-1 (*body*, *pos*))))

Theorem: get-lr-p-pc-1-comp-body-1-cur-expr-comp-body
(good-posp1 (offset (p-pc (l)), program-body (*prog*))
$\wedge$ listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) = `'quote`)
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (*prog* = p-current-program (l)))
$\rightarrow$ (get (lr-p-pc-1 (program-body (*prog*), offset (p-pc (l))),
comp-body (program-body (*prog*)))
$=$ list (`'dl`,
lr-make-label (lr-p-pc-1 (program-body (*prog*),
offset (p-pc (l)))),
nil,
list (`'push-constant`, cadr (lr-expr (l)))))

Theorem: get-lr-p-pc-1-comp-body-1-quote
(good-posp1 (offset (p-pc (l)), program-body (car (p-prog-segment (l))))
$\wedge$ listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) = `'quote`)
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (area-name (p-pc (l)) = caar (p-prog-segment (l))))
$\rightarrow$ (get (lr-p-pc-1 (program-body (car (p-prog-segment (l))), offset (p-pc (l))),
comp-body-1 (**t**, program-body (car (p-prog-segment (l))), `0`))
$=$ list (`'push-constant`, cadr (lr-expr (l))))

THEOREM: proper-p-temp-stkp-p-temp-stk-lr-push-tstk-quote
(lr-programs-properp (l, *table*)
 $\wedge$ proper-p-prog-segmentp (comp-programs (p-prog-segment (l)), lr->p (l))
 $\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 $\wedge$ listp (lr-expr (l))
 $\wedge$ (car (lr-expr (l)) = `'quote`)
 $\wedge$ proper-p-temp-stkp (p-temp-stk (l), lr->p (l)))
$\rightarrow$ proper-p-temp-stkp (p-temp-stk (lr-push-tstk (l, cadr (lr-expr (l)))),
lr->p (l))

THEOREM: proper-p-statep-lr-push-tstk-quote
(proper-p-statep (lr->p (l))
 $\wedge$ listp (lr-expr (l))
 $\wedge$ (car (lr-expr (l)) = `'quote`)
 $\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 $\wedge$ lr-programs-properp (l, *table*)
 $\wedge$ (p-psw (lr-push-tstk (l, cadr (lr-expr (l)))) = `'run`))
$\rightarrow$ proper-p-statep (lr->p (lr-push-tstk (l, cadr (lr-expr (l)))))

THEOREM: good-posp-dv-1-funcall-lr-expr
(listp (lr-expr (l))
 $\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
 $\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-EVAL)
 $\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-TEST)
 $\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-FETCH)
 $\wedge$ (car (lr-expr (l)) $\neq$ `'quote`)
 $\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l))))
$\rightarrow$ good-posp (`'list`,
dv (offset (p-pc (l)), `1`),
program-body (p-current-program (l)))

THEOREM: plistp-pairlist
plistp (pairlist (x, y))

THEOREM: all-p-objectps-append
plistp (*lst1*)
$\rightarrow$ (all-p-objectps (append (*lst1*, *lst2*), p)
= (all-p-objectps (*lst1*, p) $\wedge$ all-p-objectps (*lst2*, p)))

THEOREM: all-p-objectps-reverse
plistp (*lst*) $\rightarrow$ (all-p-objectps (reverse (*lst*), p) = all-p-objectps (*lst*, p))

THEOREM: plistp-first-n
plistp (first-n (n, *list*))

THEOREM: proper-p-temp-stkp-all-p-objectps
proper-p-temp-stkp ($temp\text{-}stk$, p) $\rightarrow$ all-p-objectps ($temp\text{-}stk$, p)

THEOREM: all-p-objectps-first-n
((length (lst) $\not<$ n) $\wedge$ all-p-objectps (lst, p))
$\rightarrow$ all-p-objectps (first-n (n, lst), p)

THEOREM: strip-cars-append
strip-cars (append (x, y)) $=$ append (strip-cars (x), strip-cars (y))

EVENT: Disable strip-cars-append.

THEOREM: strip-cars-pairlist
strip-cars (pairlist (x, y)) $=$ plist (x)

EVENT: Disable strip-cars-pairlist.

THEOREM: strip-cars-pair-temps-with-initial-values
strip-cars (pair-temps-with-initial-values ($temp\text{-}var\text{-}decls$))
$=$ strip-cars ($temp\text{-}var\text{-}decls$)

THEOREM: length-popn
(length ($list$) $\not<$ n) $\rightarrow$ (length (popn (n, $list$)) $=$ (length ($list$) $-$ n))

THEOREM: proper-p-temp-stkp-popn
((length ($temp\text{-}stk$) $\not<$ n) $\wedge$ proper-p-temp-stkp ($temp\text{-}stk$, p))
$\rightarrow$ proper-p-temp-stkp (popn (n, $temp\text{-}stk$), p)

THEOREM: proper-p-prog-segmentp-length-program-body
(proper-p-prog-segmentp ($prog\text{-}segment$, p) $\wedge$ definedp ($name$, $prog\text{-}segment$))
$\rightarrow$ listp (program-body (assoc ($name$, $prog\text{-}segment$)))

THEOREM: ret-pc-make-p-call-frame
ret-pc (make-p-call-frame ($f\text{-}vars$, $temp\text{-}stk$, $temp\text{-}var\text{-}dcls$, $ret\text{-}pc$)) $=$ $ret\text{-}pc$

THEOREM: bindings-make-p-call-frame
bindings (make-p-call-frame ($f\text{-}vars$, $temp\text{-}stk$, $temp\text{-}var\text{-}dcls$, $ret\text{-}pc$))
$=$ append (pair-formal-vars-with-actuals ($f\text{-}vars$, $temp\text{-}stk$),
pair-temps-with-initial-values ($temp\text{-}var\text{-}dcls$))

THEOREM: cddr-nil-make-p-call-frame
cddr (make-p-call-frame ($f\text{-}vars$, $temp\text{-}stk$, $temp\text{-}var\text{-}dcls$, $ret\text{-}pc$)) $=$ **nil**

THEOREM: listp-cdr-make-p-call-frame
listp (cdr (make-p-call-frame ($f\text{-}vars$, $temp\text{-}stk$, $temp\text{-}var\text{-}dcls$, $ret\text{-}pc$)))

THEOREM: length-pairlist
length (pairlist (x, y)) = length (x)

THEOREM: length-pair-temps-with-initial-values
length (pair-temps-with-initial-values (*temp-var-dcls*))
= length (*temp-var-dcls*)

THEOREM: not-proper-p-statep-not-listp-p-ctrl-stk
($\neg$ listp (p-ctrl-stk (l))) $\rightarrow$ ($\neg$ proper-p-statep (lr->p (l)))

THEOREM: proper-p-statep-bad-type-1
((fetch (car (p-temp-stk (l)), p-data-segment (l))
  $\neq$ list (type (fetch (car (p-temp-stk (l)), p-data-segment (l))),
        untag (fetch (car (p-temp-stk (l)), p-data-segment (l)))))
 $\wedge$ adpp (untag (car (p-temp-stk (l))), p-data-segment (l)))
$\rightarrow$ ($\neg$ proper-p-statep (lr->p (l)))

THEOREM: p-good-resultp-run-car
(proper-p-statep (lr->p (l))
 $\wedge$ (p-psw (l) = **'run**)
 $\wedge$ lr-programs-properp (l, *table*)
 $\wedge$ (unlabel (get (offset (*pc*),
                program-body (assoc (area-name (*pc*),
                        p-prog-segment (lr->p (l))))))
   = `'(call car)`))
$\rightarrow$ p-good-resultp (p (p-set-pc (lr->p (l), *pc*),
          p-car-clock (p-set-pc (lr->p (l), *pc*))),
         p-data-segment (l),
         if fetch (top (p-temp-stk (l)), p-data-segment (l))
           = tag (**'nat**, LR-CONS-TAG)
         then cons (fetch (add-addr (top (p-temp-stk (l)),
                          LR-CAR-OFFSET),
                      p-data-segment (l)),
                  cdr (p-temp-stk (l)))
         else cons (LR-0-ADDR, cdr (p-temp-stk (l))) **endif**,
         p-ctrl-stk (l),
         add-addr (*pc*, 1))

THEOREM: p-good-resultp-run-cdr
(proper-p-statep (lr->p (l))
 $\wedge$ (p-psw (l) = **'run**)
 $\wedge$ lr-programs-properp (l, *table*)
 $\wedge$ (unlabel (get (offset (*pc*),
                program-body (assoc (area-name (*pc*),
                        p-prog-segment (lr->p (l))))))

```
       =   '(call cdr)))
→   p-good-resultp (p (p-set-pc (lr->p (l), pc),
                       p-cdr-clock (p-set-pc (lr->p (l), pc))),
                    p-data-segment (l),
                    if fetch (top (p-temp-stk (l)), p-data-segment (l))
                       =   tag ('nat, LR-CONS-TAG)
                    then cons (fetch (add-addr (top (p-temp-stk (l)),
                                                LR-CDR-OFFSET),
                                      p-data-segment (l)),
                               cdr (p-temp-stk (l)))
                    else cons (LR-0-ADDR, cdr (p-temp-stk (l))) endif,
                    p-ctrl-stk (l),
                    add-addr (pc, 1))
```

THEOREM: p-good-resultp-run-listp

```
(proper-p-statep (lr->p (l))
 ∧   (p-psw (l) = 'run)
 ∧   lr-programs-properp (l, table)
 ∧   (unlabel (get (offset (pc),
                    program-body (assoc (area-name (pc),
                                         p-prog-segment (lr->p (l))))))
       =   '(call listp)))
→   p-good-resultp (p (p-set-pc (lr->p (l), pc),
                       p-listp-clock (p-set-pc (lr->p (l), pc))),
                    p-data-segment (l),
                    if fetch (car (p-temp-stk (l)), p-data-segment (l))
                       =   tag ('nat, LR-CONS-TAG)
                    then cons (LR-T-ADDR, cdr (p-temp-stk (l)))
                    else cons (LR-F-ADDR, cdr (p-temp-stk (l))) endif,
                    p-ctrl-stk (l),
                    add-addr (pc, 1))
```

THEOREM: p-good-resultp-run-nlistp

```
(proper-p-statep (lr->p (l))
 ∧   (p-psw (l) = 'run)
 ∧   lr-programs-properp (l, table)
 ∧   (unlabel (get (offset (pc),
                    program-body (assoc (area-name (pc),
                                         p-prog-segment (lr->p (l))))))
       =   '(call nlistp)))
→   p-good-resultp (p (p-set-pc (lr->p (l), pc),
                       p-nlistp-clock (p-set-pc (lr->p (l), pc))),
                    p-data-segment (l),
                    if fetch (car (p-temp-stk (l)), p-data-segment (l))
```

```
                       =   tag ('nat, LR-CONS-TAG)
                   then cons (LR-F-ADDR, cdr (p-temp-stk (l)))
                   else cons (LR-T-ADDR, cdr (p-temp-stk (l))) endif,
                   p-ctrl-stk (l),
                   add-addr (pc, 1))
```

THEOREM: p-good-resultp-run-truep

```
 (proper-p-statep (lr->p (l))
  ∧   (p-psw (l) = 'run)
  ∧   lr-programs-properp (l, table)
  ∧   (unlabel (get (offset (pc),
                 program-body (assoc (area-name (pc),
                                      p-prog-segment (lr->p (l))))))
        =  '(call truep)))
 →   p-good-resultp (p (p-set-pc (lr->p (l), pc),
                       p-truep-clock (p-set-pc (lr->p (l), pc))),
                   p-data-segment (l),
                   if fetch (car (p-temp-stk (l)), p-data-segment (l))
                       =   tag ('nat, LR-TRUE-TAG)
                   then cons (LR-T-ADDR, cdr (p-temp-stk (l)))
                   else cons (LR-F-ADDR, cdr (p-temp-stk (l))) endif,
                   p-ctrl-stk (l),
                   add-addr (pc, 1))
```

EVENT: Disable proper-p-statep-bad-type-1.

THEOREM: p-good-resultp-run-cons

```
 (proper-p-statep (lr->p (l))
  ∧   (p-psw (l) = 'run)
  ∧   lr-programs-properp (l, table)
  ∧   (unlabel (get (offset (pc),
                 program-body (assoc (area-name (pc),
                                      p-prog-segment (lr->p (l))))))
        =  '(call cons)))
 →   p-good-resultp (p (p-set-pc (lr->p (l), pc),
                       p-cons-clock (p-set-pc (lr->p (l), pc))),
                   deposit (fetch (add-addr (fetch (LR-FP-ADDR,
                                                    p-data-segment (l)),
                                             LR-REF-COUNT-OFFSET),
                                   p-data-segment (l)),
                            LR-FP-ADDR,
                            deposit-a-list (list (tag ('nat, LR-CONS-TAG),
                                                  tag ('nat, 1),
                                                  top1 (p-temp-stk (l)),
```

top (p-temp-stk (l))),
fetch (LR-FP-ADDR,
p-data-segment (l)),
p-data-segment (l))),
cons (fetch (LR-FP-ADDR, p-data-segment (l)),
cddr (p-temp-stk (l))),
p-ctrl-stk (l),
add-addr (pc, `1`))

THEOREM: p-objectp-bad-type
($object \neq$ list (type ($object$), untag ($object$))) $\rightarrow$ ($\neg$ p-objectp ($object$, p))

THEOREM: proper-p-statep-bad-type-2
((car (p-temp-stk (l))
$\neq$ list (type (car (p-temp-stk (l))), untag (car (p-temp-stk (l)))))
$\wedge$ listp (p-temp-stk (l)))
$\rightarrow$ ($\neg$ proper-p-statep (lr->p (l)))

THEOREM: p-good-resultp-run-falsep
(proper-p-statep (lr->p (l))
$\wedge$ (p-psw (l) = **`'run`**)
$\wedge$ lr-programs-properp (l, $table$)
$\wedge$ (unlabel (get (offset (pc),
program-body (assoc (area-name (pc),
p-prog-segment (lr->p (l))))))
= **`'(call falsep)`**))
$\rightarrow$ p-good-resultp (p (p-set-pc (lr->p (l), pc),
p-falsep-clock (p-set-pc (lr->p (l), pc))),
p-data-segment (l),
if car (p-temp-stk (l)) = LR-F-ADDR
then cons (LR-T-ADDR, cdr (p-temp-stk (l)))
else cons (LR-F-ADDR, cdr (p-temp-stk (l))) **endif**,
p-ctrl-stk (l),
add-addr (pc, `1`))

EVENT: Disable proper-p-statep-bad-type-2.

THEOREM: p-good-resultp-run-false
(proper-p-statep (lr->p (l))
$\wedge$ (p-psw (l) = **`'run`**)
$\wedge$ lr-programs-properp (l, $table$)
$\wedge$ (unlabel (get (offset (pc),
program-body (assoc (area-name (pc),
p-prog-segment (lr->p (l))))))

```
      =   '(call false)))
→   p-good-resultp (p (p-set-pc (lr->p (l), pc),
                          p-false-clock (p-set-pc (lr->p (l), pc))),
                       p-data-segment (l),
                       cons (LR-F-ADDR, p-temp-stk (l)),
                       p-ctrl-stk (l),
                       add-addr (pc, 1))
```

THEOREM: p-good-resultp-run-true

```
 (proper-p-statep (lr->p (l))
  ∧   (p-psw (l) = 'run)
  ∧   lr-programs-properp (l, table)
  ∧   (unlabel (get (offset (pc),
                     program-body (assoc (area-name (pc),
                                          p-prog-segment (lr->p (l))))))
      =   '(call true)))
→   p-good-resultp (p (p-set-pc (lr->p (l), pc),
                          p-true-clock (p-set-pc (lr->p (l), pc))),
                       p-data-segment (l),
                       cons (LR-T-ADDR, p-temp-stk (l)),
                       p-ctrl-stk (l),
                       add-addr (pc, 1))
```

THEOREM: p-temp-stk-p-ctrl-stk-p-data-segment-run-car

```
 (proper-p-statep (lr->p (l))
  ∧   (p-psw (l) = 'run)
  ∧   (p-psw (p (p-set-pc (lr->p (l), pc), p-car-clock (p-set-pc (lr->p (l), pc))))
      =   'run)
  ∧   lr-programs-properp (l, table)
  ∧   (unlabel (get (offset (pc),
                     program-body (assoc (area-name (pc),
                                          p-prog-segment (lr->p (l))))))
      =   '(call car)))
→   ((p-temp-stk (p (p-set-pc (lr->p (l), pc),
                       p-car-clock (p-set-pc (lr->p (l), pc))))
       =   if fetch (top (p-temp-stk (l)), p-data-segment (l))
              =   tag ('nat, LR-CONS-TAG)
           then cons (fetch (add-addr (top (p-temp-stk (l)), LR-CAR-OFFSET),
                             p-data-segment (l)),
                      cdr (p-temp-stk (l)))
           else cons (LR-0-ADDR, cdr (p-temp-stk (l))) endif)
     ∧   (p-ctrl-stk (p (p-set-pc (lr->p (l), pc),
                           p-car-clock (p-set-pc (lr->p (l), pc))))
          =   p-ctrl-stk (l))
```

∧ (p-data-segment (p (p-set-pc (lr->p (*l*), *pc*),
p-car-clock (p-set-pc (lr->p (*l*), *pc*))))
= p-data-segment (*l*)))

THEOREM: p-temp-stk-p-ctrl-stk-p-data-segment-run-cdr
(proper-p-statep (lr->p (*l*))
∧ (p-psw (*l*) = `'run`)
∧ (p-psw (p (p-set-pc (lr->p (*l*), *pc*), p-cdr-clock (p-set-pc (lr->p (*l*), *pc*))))
= `'run`)
∧ lr-programs-properp (*l*, *table*)
∧ (unlabel (get (offset (*pc*),
program-body (assoc (area-name (*pc*),
p-prog-segment (lr->p (*l*))))))
= `'(call cdr)`))
→ ((p-temp-stk (p (p-set-pc (lr->p (*l*), *pc*),
p-cdr-clock (p-set-pc (lr->p (*l*), *pc*))))
= **if** fetch (top (p-temp-stk (*l*)), p-data-segment (*l*))
= tag (`'nat`, LR-CONS-TAG)
then cons (fetch (add-addr (top (p-temp-stk (*l*)), LR-CDR-OFFSET),
p-data-segment (*l*)),
cdr (p-temp-stk (*l*)))
else cons (LR-0-ADDR, cdr (p-temp-stk (*l*))) **endif**)
∧ (p-ctrl-stk (p (p-set-pc (lr->p (*l*), *pc*),
p-cdr-clock (p-set-pc (lr->p (*l*), *pc*))))
= p-ctrl-stk (*l*))
∧ (p-data-segment (p (p-set-pc (lr->p (*l*), *pc*),
p-cdr-clock (p-set-pc (lr->p (*l*), *pc*))))
= p-data-segment (*l*)))

THEOREM: p-temp-stk-p-ctrl-stk-p-data-segment-run-cons
let *fp-addr* **be** fetch (LR-FP-ADDR, p-data-segment (*l*))
in
(proper-p-statep (lr->p (*l*))
∧ (p-psw (*l*) = `'run`)
∧ (p-psw (p (p-set-pc (lr->p (*l*), *pc*),
p-cons-clock (p-set-pc (lr->p (*l*), *pc*))))
= `'run`)
∧ lr-programs-properp (*l*, *table*)
∧ (unlabel (get (offset (*pc*),
program-body (assoc (area-name (*pc*),
p-prog-segment (lr->p (*l*))))))
= `'(call cons)`))
→ ((p-temp-stk (p (p-set-pc (lr->p (*l*), *pc*),
p-cons-clock (p-set-pc (lr->p (*l*), *pc*))))

```
      =  cons (fetch (LR-FP-ADDR, p-data-segment (l)),
                 cddr (p-temp-stk (l))))
    ∧  (p-ctrl-stk (p (p-set-pc (lr->p (l), pc),
                        p-cons-clock (p-set-pc (lr->p (l), pc))))
          =  p-ctrl-stk (l))
    ∧  (p-data-segment (p (p-set-pc (lr->p (l), pc),
                            p-cons-clock (p-set-pc (lr->p (l), pc))))
          =  deposit (fetch (add-addr (fetch (LR-FP-ADDR,
                                              p-data-segment (l)),
                                       LR-REF-COUNT-OFFSET),
                             p-data-segment (l)),
                      LR-FP-ADDR,
                      deposit-a-list (list (tag ('nat,
                                                 LR-CONS-TAG),
                                            tag ('nat, 1),
                                            top1 (p-temp-stk (l)),
                                            top (p-temp-stk (l))),
                                      fetch (LR-FP-ADDR,
                                             p-data-segment (l)),
                                      p-data-segment (l))))) endlet
```

THEOREM: p-temp-stk-p-ctrl-stk-p-data-segment-run-false

```
 (proper-p-statep (lr->p (l))
  ∧  (p-psw (l) = 'run)
  ∧  (p-psw (p (p-set-pc (lr->p (l), pc), p-false-clock (p-set-pc (lr->p (l), pc))))
        =  'run)
  ∧  lr-programs-properp (l, table)
  ∧  (unlabel (get (offset (pc),
                    program-body (assoc (area-name (pc),
                                         p-prog-segment (lr->p (l))))))
        =  '(call false)))
 →  ((p-temp-stk (p (p-set-pc (lr->p (l), pc),
                     p-false-clock (p-set-pc (lr->p (l), pc))))
       =  cons (LR-F-ADDR, p-temp-stk (l)))
     ∧  (p-ctrl-stk (p (p-set-pc (lr->p (l), pc),
                        p-false-clock (p-set-pc (lr->p (l), pc))))
          =  p-ctrl-stk (l))
     ∧  (p-data-segment (p (p-set-pc (lr->p (l), pc),
                            p-false-clock (p-set-pc (lr->p (l), pc))))
          =  p-data-segment (l)))
```

THEOREM: p-temp-stk-p-ctrl-stk-p-data-segment-run-falsep

```
 (proper-p-statep (lr->p (l))
  ∧  (p-psw (l) = 'run)
```

$\land$ (p-psw (p (p-set-pc (lr->p (l), pc), p-falsep-clock (p-set-pc (lr->p (l), pc))))
= `'run`)
$\land$ lr-programs-properp (l, *table*)
$\land$ (unlabel (get (offset (pc),
program-body (assoc (area-name (pc),
p-prog-segment (lr->p (l))))))
= `'(call falsep)`))
$\rightarrow$ ((p-temp-stk (p (p-set-pc (lr->p (l), pc),
p-falsep-clock (p-set-pc (lr->p (l), pc))))
= **if** car (p-temp-stk (l)) = LR-F-ADDR
then cons (LR-T-ADDR, cdr (p-temp-stk (l)))
else cons (LR-F-ADDR, cdr (p-temp-stk (l))) **endif**)
$\land$ (p-ctrl-stk (p (p-set-pc (lr->p (l), pc),
p-falsep-clock (p-set-pc (lr->p (l), pc))))
= p-ctrl-stk (l))
$\land$ (p-data-segment (p (p-set-pc (lr->p (l), pc),
p-falsep-clock (p-set-pc (lr->p (l), pc))))
= p-data-segment (l)))

THEOREM: p-temp-stk-p-ctrl-stk-p-data-segment-run-listp
(proper-p-statep (lr->p (l))
$\land$ (p-psw (l) = `'run`)
$\land$ (p-psw (p (p-set-pc (lr->p (l), pc), p-listp-clock (p-set-pc (lr->p (l), pc))))
= `'run`)
$\land$ lr-programs-properp (l, *table*)
$\land$ (unlabel (get (offset (pc),
program-body (assoc (area-name (pc),
p-prog-segment (lr->p (l))))))
= `'(call listp)`))
$\rightarrow$ ((p-temp-stk (p (p-set-pc (lr->p (l), pc),
p-listp-clock (p-set-pc (lr->p (l), pc))))
= **if** fetch (car (p-temp-stk (l)), p-data-segment (l))
= tag (`'nat`, LR-CONS-TAG)
then cons (LR-T-ADDR, cdr (p-temp-stk (l)))
else cons (LR-F-ADDR, cdr (p-temp-stk (l))) **endif**)
$\land$ (p-ctrl-stk (p (p-set-pc (lr->p (l), pc),
p-listp-clock (p-set-pc (lr->p (l), pc))))
= p-ctrl-stk (l))
$\land$ (p-data-segment (p (p-set-pc (lr->p (l), pc),
p-listp-clock (p-set-pc (lr->p (l), pc))))
= p-data-segment (l)))

THEOREM: p-temp-stk-p-ctrl-stk-p-data-segment-run-nlistp
(proper-p-statep (lr->p (l))

∧ (p-psw (*l*) = **'run**)
∧ (p-psw (p (p-set-pc (lr->p (*l*), *pc*), p-nlistp-clock (p-set-pc (lr->p (*l*), *pc*))))
= **'run**)
∧ lr-programs-properp (*l*, *table*)
∧ (unlabel (get (offset (*pc*),
program-body (assoc (area-name (*pc*),
p-prog-segment (lr->p (*l*))))))
= **'(call nlistp)**))
→ ((p-temp-stk (p (p-set-pc (lr->p (*l*), *pc*),
p-nlistp-clock (p-set-pc (lr->p (*l*), *pc*))))
= **if** fetch (car (p-temp-stk (*l*)), p-data-segment (*l*))
= tag (**'nat**, LR-CONS-TAG)
then cons (LR-F-ADDR, cdr (p-temp-stk (*l*)))
else cons (LR-T-ADDR, cdr (p-temp-stk (*l*))) **endif**)
∧ (p-ctrl-stk (p (p-set-pc (lr->p (*l*), *pc*),
p-nlistp-clock (p-set-pc (lr->p (*l*), *pc*))))
= p-ctrl-stk (*l*))
∧ (p-data-segment (p (p-set-pc (lr->p (*l*), *pc*),
p-nlistp-clock (p-set-pc (lr->p (*l*), *pc*))))
= p-data-segment (*l*)))

THEOREM: p-temp-stk-p-ctrl-stk-p-data-segment-run-true
(proper-p-statep (lr->p (*l*))
∧ (p-psw (*l*) = **'run**)
∧ (p-psw (p (p-set-pc (lr->p (*l*), *pc*), p-true-clock (p-set-pc (lr->p (*l*), *pc*))))
= **'run**)
∧ lr-programs-properp (*l*, *table*)
∧ (unlabel (get (offset (*pc*),
program-body (assoc (area-name (*pc*),
p-prog-segment (lr->p (*l*))))))
= **'(call true)**))
→ ((p-temp-stk (p (p-set-pc (lr->p (*l*), *pc*),
p-true-clock (p-set-pc (lr->p (*l*), *pc*))))
= cons (LR-T-ADDR, p-temp-stk (*l*)))
∧ (p-ctrl-stk (p (p-set-pc (lr->p (*l*), *pc*),
p-true-clock (p-set-pc (lr->p (*l*), *pc*))))
= p-ctrl-stk (*l*))
∧ (p-data-segment (p (p-set-pc (lr->p (*l*), *pc*),
p-true-clock (p-set-pc (lr->p (*l*), *pc*))))
= p-data-segment (*l*)))

THEOREM: p-temp-stk-p-ctrl-stk-p-data-segment-run-truep
(proper-p-statep (lr->p (*l*))
∧ (p-psw (*l*) = **'run**)

$\wedge$ (p-psw (p (p-set-pc (lr->p (*l*), *pc*), p-truep-clock (p-set-pc (lr->p (*l*), *pc*))))
  = `'run`)
$\wedge$ lr-programs-properp (*l*, *table*)
$\wedge$ (unlabel (get (offset (*pc*),
    program-body (assoc (area-name (*pc*),
      p-prog-segment (lr->p (*l*))))))
  = `'(call truep)`))
$\rightarrow$ ((p-temp-stk (p (p-set-pc (lr->p (*l*), *pc*),
    p-truep-clock (p-set-pc (lr->p (*l*), *pc*))))
  = **if** fetch (car (p-temp-stk (*l*)), p-data-segment (*l*))
      = tag (`'nat`, LR-TRUE-TAG)
    then cons (LR-T-ADDR, cdr (p-temp-stk (*l*)))
    else cons (LR-F-ADDR, cdr (p-temp-stk (*l*))) **endif**)
  $\wedge$ (p-ctrl-stk (p (p-set-pc (lr->p (*l*), *pc*),
      p-truep-clock (p-set-pc (lr->p (*l*), *pc*))))
    = p-ctrl-stk (*l*))
  $\wedge$ (p-data-segment (p (p-set-pc (lr->p (*l*), *pc*),
      p-truep-clock (p-set-pc (lr->p (*l*), *pc*))))
    = p-data-segment (*l*)))

THEOREM: get-last-funcall-cur-expr
(listp (*expr*)
$\wedge$ (car (*expr*) $\neq$ `'if`)
$\wedge$ (car (*expr*) $\neq$ S-TEMP-EVAL)
$\wedge$ (car (*expr*) $\neq$ S-TEMP-TEST)
$\wedge$ (car (*expr*) $\neq$ S-TEMP-FETCH)
$\wedge$ (car (*expr*) $\neq$ `'quote`))
$\rightarrow$ (get (lr-p-c-size-list (length (*expr*) $-$ 1, *expr*), comp-body-1 (**t**, *expr*, *n*))
  = **if** definedp (car (*expr*), P-RUNTIME-SUPPORT-PROGRAMS)
    then list (`'call`, car (*expr*))
    else list (`'call`, user-fname (car (*expr*))) **endif**)

THEOREM: not-listp-p-prog-segment-lr-expr
($\neg$ listp (p-prog-segment (*l*))) $\rightarrow$ ($\neg$ listp (lr-expr (*l*)))

THEOREM: get-offset-return-pc-program-body-assoc-comp-programs
(good-posp1 (offset (p-pc (*l*)),
    program-body (assoc (area-name (p-pc (*l*)), p-prog-segment (*l*))))
$\wedge$ lr-programs-properp (*l*, *table*)
$\wedge$ (car (lr-expr (*l*)) $\neq$ `'if`)
$\wedge$ (car (lr-expr (*l*)) $\neq$ S-TEMP-EVAL)
$\wedge$ (car (lr-expr (*l*)) $\neq$ S-TEMP-TEST)
$\wedge$ (car (lr-expr (*l*)) $\neq$ S-TEMP-FETCH)
$\wedge$ (car (lr-expr (*l*)) $\neq$ `'quote`)
$\wedge$ listp (lr-expr (*l*)))

$\rightarrow$ (get (offset (lr-return-pc (l)),
program-body (assoc (area-name (p-pc (l)),
comp-programs (p-prog-segment (l)))))
= list (`'dl`,
lr-make-label (offset (lr-return-pc (l))),
nil,
if definedp (car (lr-expr (l)), P-RUNTIME-SUPPORT-PROGRAMS)
then list (`'call`, car (lr-expr (l)))
else list (`'call`, user-fname (car (lr-expr (l)))) **endif**))

THEOREM: listp-p-temp-stk-proper-ctrl-stk-p-run-subr
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (*new-l*, *table*)
$\land$ lr-programs-properp (l, *table*)
$\land$ listp (lr-expr (l))
$\land$ proper-p-statep (lr->p (*new-l*))
$\land$ (p-psw (*new-l*) = `'run`)
$\land$ (p-psw (p-run-subr (car (lr-expr (l)),
p-set-pc (lr->p (*new-l*), lr-return-pc (l))))
= `'run`)
$\land$ (p-prog-segment (l) = p-prog-segment (*new-l*)))
$\rightarrow$ (listp (p-temp-stk (p-run-subr (car (lr-expr (l)),
p-set-pc (lr->p (*new-l*), lr-return-pc (l)))))
$\land$ (p-ctrl-stk (p-run-subr (car (lr-expr (l)),
p-set-pc (lr->p (*new-l*), lr-return-pc (l))))
= p-ctrl-stk (*new-l*)))

THEOREM: listp-p-temp-stk-proper-ctrl-stk-lr-apply-subr
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ lr-programs-properp (*new-l*, *table*)
$\land$ listp (lr-expr (l))
$\land$ subrp (car (lr-expr (l)))
$\land$ (p-psw (lr-apply-subr (l, *new-l*)) = `'run`)
$\land$ proper-p-statep (lr->p (*new-l*))
$\land$ (p-psw (*new-l*) = `'run`)
$\land$ (p-prog-segment (l) = p-prog-segment (*new-l*)))
$\rightarrow$ (listp (p-temp-stk (lr-apply-subr (l, *new-l*)))
$\land$ (p-ctrl-stk (lr-apply-subr (l, *new-l*)) = p-ctrl-stk (*new-l*)))

THEOREM: cur-expr-nlistp-pos
(*pos* $\simeq$ **nil**) $\rightarrow$ (cur-expr (*pos*, *body*) = *body*)

THEOREM: proper-p-statep-p-run-subr
(proper-p-statep (p) $\land$ (p-psw (p-run-subr (*subr*, p)) = `'run`))
$\rightarrow$ proper-p-statep (p-run-subr (*subr*, p))

THEOREM: same-signature-commutative
same-signature $(x,\ y)$ = same-signature $(y,\ x)$

THEOREM: same-signature-p-run-subr
(proper-p-statep (p)
 $\wedge$ (p-psw (p-run-subr $(subr,\ p)$) = `'run`)
 $\wedge$ ($data\text{-}seg$ = p-data-segment (p)))
$\rightarrow$ same-signature ($data\text{-}seg$, p-data-segment (p-run-subr $(subr,\ p)$))

THEOREM: proper-p-framep-lr->p-similar-states
(proper-p-framep ($frame$, $name$, $p0$)
 $\wedge$ same-signature (p-data-segment ($p0$), p-data-segment ($p1$))
 $\wedge$ (p-prog-segment ($p0$) = p-prog-segment ($p1$))
 $\wedge$ (p-word-size ($p0$) = p-word-size ($p1$)))
$\rightarrow$ proper-p-framep ($frame$, $name$, $p1$)

THEOREM: car-untag-p-pc-lr-eval
car (untag (p-pc (lr-eval ($flag$, l, c)))) = car (untag (p-pc (l)))

THEOREM: lessp-cdr-untag-lr-return-pc-lr-p-c-size
(good-posp1 (offset (p-pc (l)),
 program-body (assoc (car (untag (p-pc (l))), p-prog-segment (l))))
 $\wedge$ lr-programs-properp (l, $table$)
 $\wedge$ (subrp (car (lr-expr (l))) $\vee$ litatom (car (lr-expr (l))))
 $\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
 $\wedge$ (car (lr-expr (l)) $\neq$ `'quote`)
 $\wedge$ ($name$ = area-name (p-pc (l))))
$\rightarrow$ (cdr (untag (lr-return-pc (l)))
 $<$ length (program-body (assoc ($name$,
 comp-programs (p-prog-segment (l))))))

THEOREM: proper-p-statep-lr-apply-subr-state
(proper-p-statep (lr->p ($new\text{-}l$))
 $\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 $\wedge$ lr-programs-properp (l, $table$)
 $\wedge$ listp (lr-expr (l))
 $\wedge$ subrp (car (lr-expr (l)))
 $\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
 $\wedge$ (p-prog-segment (l) = p-prog-segment ($new\text{-}l$))
 $\wedge$ (area-name (p-pc (l)) = area-name (p-pc ($new\text{-}l$))))
$\rightarrow$ proper-p-statep (p-set-pc (lr->p ($new\text{-}l$), lr-return-pc (l)))

THEOREM: same-signature-lr-apply-subr
(proper-p-statep (lr->p ($new\text{-}l$))
 $\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))

$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ listp (lr-expr (l))
$\wedge$ subrp (car (lr-expr (l)))
$\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
$\wedge$ (p-prog-segment (l) $=$ p-prog-segment (*new-l*))
$\wedge$ (area-name (p-pc (l)) $=$ area-name (p-pc (*new-l*)))
$\wedge$ (p-psw (lr-apply-subr (l, *new-l*)) $=$ `'run`)
$\wedge$ (*data-seg* $=$ p-data-segment (*new-l*)))
$\rightarrow$ same-signature (*data-seg*, p-data-segment (lr-apply-subr (l, *new-l*)))

THEOREM: p-current-program-lr-apply-subr
((area-name (p-pc (*new-l*)) $=$ area-name (p-pc (l)))
$\wedge$ (p-prog-segment (*new-l*) $=$ p-prog-segment (l)))
$\rightarrow$ (p-current-program (lr-apply-subr (l, *new-l*)) $=$ p-current-program (l))

THEOREM: p-current-program-lr-eval
p-current-program (lr-eval (*flag*, l, c)) $=$ p-current-program (l)

THEOREM: proper-p-framep-lr-apply-subr
(proper-p-statep (lr->p (*new-l*))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ listp (lr-expr (l))
$\wedge$ subrp (car (lr-expr (l)))
$\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
$\wedge$ (p-prog-segment (l) $=$ p-prog-segment (*new-l*))
$\wedge$ (area-name (p-pc (l)) $=$ area-name (p-pc (*new-l*)))
$\wedge$ (p-psw (lr-apply-subr (l, *new-l*)) $=$ `'run`)
$\wedge$ (*name* $=$ area-name (p-pc (*new-l*))))
$\rightarrow$ proper-p-framep (car (p-ctrl-stk (*new-l*)),
name,
lr->p (lr-apply-subr (l, *new-l*)))

THEOREM: proper-p-statep-lr->p-lessp-ctrl-stk-size
(proper-p-statep (lr->p (l)) $\wedge$ (*max* $=$ p-max-ctrl-stk-size (l)))
$\rightarrow$ ((*max* $<$ p-ctrl-stk-size (p-ctrl-stk (l))) $=$ **f**)

EVENT: Disable proper-p-statep-lr->p-lessp-ctrl-stk-size.

THEOREM: proper-p-statep-lr->p-numberp-max-ctrl-stk-size
proper-p-statep (lr->p (l)) $\rightarrow$ (p-max-ctrl-stk-size (l) $\in$ $\mathbf{N}$)

EVENT: Disable proper-p-statep-lr->p-numberp-max-ctrl-stk-size.

THEOREM: proper-p-statep-lr->p-numberp-max-temp-stk-size
proper-p-statep (lr->p (l)) $\rightarrow$ (p-max-temp-stk-size $(l) \in \mathbf{N}$)

EVENT: Disable proper-p-statep-lr->p-numberp-max-temp-stk-size.

THEOREM: proper-p-statep-lr->p-numberp-word-size
proper-p-statep (lr->p (l)) $\rightarrow$ (p-word-size $(l) \in \mathbf{N}$)

EVENT: Disable proper-p-statep-lr->p-numberp-word-size.

THEOREM: proper-p-statep-lr->p-lessp-max-ctrl-stk-size
proper-p-statep (lr->p (l))
$\rightarrow$ ((p-max-ctrl-stk-size (l) $<$ exp (2, p-word-size (l))) $=$ **t**)

EVENT: Disable proper-p-statep-lr->p-lessp-max-ctrl-stk-size.

THEOREM: proper-p-statep-lr->p-lessp-max-temp-stk-size
proper-p-statep (lr->p (l))
$\rightarrow$ ((p-max-temp-stk-size (l) $<$ exp (2, p-word-size (l))) $=$ **t**)

EVENT: Disable proper-p-statep-lr->p-lessp-max-temp-stk-size.

THEOREM: proper-p-statep-lr->p-equal-word-size-0
proper-p-statep (lr->p (l)) $\rightarrow$ (p-word-size $(l) \neq$ 0)

EVENT: Disable proper-p-statep-lr->p-equal-word-size-0.

THEOREM: proper-p-ctrl-stkp-lr-apply-subr
(proper-p-statep (lr->p (*new-l*))
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ listp (lr-expr (l))
$\land$ subrp (car (lr-expr (l)))
$\land$ (car (lr-expr (l)) $\neq$ `'if`)
$\land$ (p-prog-segment (l) $=$ p-prog-segment (*new-l*))
$\land$ (area-name (p-pc (l)) $=$ area-name (p-pc (*new-l*)))
$\land$ (p-psw (lr-apply-subr (l, *new-l*)) $=$ `'run`))
$\rightarrow$ proper-p-ctrl-stkp (cdr (p-ctrl-stk (*new-l*)),
area-name (ret-pc (car (p-ctrl-stk (*new-l*)))),
lr->p (lr-apply-subr (l, *new-l*)))

THEOREM: proper-p-prog-segmentp-lr-apply-subr
(proper-p-statep (lr->p (*new-l*))

$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ listp (lr-expr (l))
$\land$ subrp (car (lr-expr (l)))
$\land$ (car (lr-expr (l)) $\neq$ `'if`)
$\land$ (p-prog-segment (l) = p-prog-segment (*new-l*))
$\land$ (area-name (p-pc (l)) = area-name (p-pc (*new-l*)))
$\land$ (p-psw (lr-apply-subr (l, *new-l*)) = **`'run`**)
$\land$ (*progs* = p-prog-segment (*new-l*)))
$\rightarrow$ proper-p-prog-segmentp (comp-programs (*progs*),
lr->p (lr-apply-subr (l, *new-l*)))

THEOREM: proper-p-state-p-p-run-subr-opener-1
(proper-p-statep (p) $\land$ (p-psw (p-run-subr (*subr*, p)) = **`'run`**))
$\rightarrow$ proper-p-temp-stkp (p-temp-stk (p-run-subr (*subr*, p)), p-run-subr (*subr*, p))

THEOREM: proper-p-temp-stkp-lr-apply-subr
(proper-p-statep (lr->p (*new-l*))
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ listp (lr-expr (l))
$\land$ subrp (car (lr-expr (l)))
$\land$ (car (lr-expr (l)) $\neq$ `'if`)
$\land$ (p-prog-segment (l) = p-prog-segment (*new-l*))
$\land$ (area-name (p-pc (l)) = area-name (p-pc (*new-l*)))
$\land$ (p-psw (lr-apply-subr (l, *new-l*)) = **`'run`**))
$\rightarrow$ proper-p-temp-stkp (p-temp-stk (lr-apply-subr (l, *new-l*)),
lr->p (lr-apply-subr (l, *new-l*)))

THEOREM: proper-p-state-p-p-run-subr-opener-2
(proper-p-statep (p) $\land$ (p-psw (p-run-subr (*subr*, p)) = **`'run`**))
$\rightarrow$ (p-max-temp-stk-size (p) $\not<$ length (p-temp-stk (p-run-subr (*subr*, p))))

THEOREM: not-lessp-length-p-temp-stk-lr-apply-subr
(proper-p-statep (lr->p (*new-l*))
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ listp (lr-expr (l))
$\land$ subrp (car (lr-expr (l)))
$\land$ (car (lr-expr (l)) $\neq$ `'if`)
$\land$ (p-prog-segment (l) = p-prog-segment (*new-l*))
$\land$ (area-name (p-pc (l)) = area-name (p-pc (*new-l*)))
$\land$ (p-psw (lr-apply-subr (l, *new-l*)) = **`'run`**)
$\land$ (p-max-temp-stk-size (l) = p-max-temp-stk-size (*new-l*)))
$\rightarrow$ (p-max-temp-stk-size (l)
$\not<$ length (p-temp-stk (lr-apply-subr (l, *new-l*))))

THEOREM: proper-p-state-p-p-run-subr-opener-3

$$
\begin{array}{l}
(\text{proper-p-statep}\,(p) \wedge (\text{p-psw}\,(\text{p-run-subr}\,(subr,\ p)) = \texttt{'run}))\\
\rightarrow \quad \text{proper-p-data-segmentp}\,(\text{p-data-segment}\,(\text{p-run-subr}\,(subr,\ p)),\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{p-run-subr}\,(subr,\ p))
\end{array}
$$

THEOREM: proper-p-data-segmentp-lr-apply-subr

$$
\begin{array}{l}
(\text{proper-p-statep}\,(\text{lr->p}\,(new\text{-}l))\\
\ \wedge \quad \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l)))\\
\ \wedge \quad \text{lr-programs-properp}\,(l,\ table)\\
\ \wedge \quad \text{listp}\,(\text{lr-expr}\,(l))\\
\ \wedge \quad \text{subrp}\,(\text{car}\,(\text{lr-expr}\,(l)))\\
\ \wedge \quad (\text{car}\,(\text{lr-expr}\,(l)) \neq \texttt{'if})\\
\ \wedge \quad (\text{p-prog-segment}\,(l) = \text{p-prog-segment}\,(new\text{-}l))\\
\ \wedge \quad (\text{area-name}\,(\text{p-pc}\,(l)) = \text{area-name}\,(\text{p-pc}\,(new\text{-}l)))\\
\ \wedge \quad (\text{p-psw}\,(\text{lr-apply-subr}\,(l,\ new\text{-}l)) = \texttt{'run}))\\
\rightarrow \quad \text{proper-p-data-segmentp}\,(\text{p-data-segment}\,(\text{lr-apply-subr}\,(l,\ new\text{-}l)),\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{lr->p}\,(\text{lr-apply-subr}\,(l,\ new\text{-}l)))
\end{array}
$$

THEOREM: lr-programs-properp-lr-set-pos

$$
\begin{array}{l}
\text{lr-programs-properp}\,(\text{lr-set-pos}\,(l,\ pos),\ table)\\
= \quad \text{lr-programs-properp}\,(l,\ table)
\end{array}
$$

THEOREM: proper-p-statep-lr-apply-subr

$$
\begin{array}{l}
(\text{listp}\,(\text{lr-expr}\,(l))\\
\ \wedge \quad \text{subrp}\,(\text{car}\,(\text{lr-expr}\,(l)))\\
\ \wedge \quad (\text{car}\,(\text{lr-expr}\,(l)) \neq \texttt{'if})\\
\ \wedge \quad \text{proper-p-statep}\,(\text{lr->p}\,(l))\\
\ \wedge \quad \text{proper-p-statep}\,(\text{lr->p}\,(\text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ pos),\ c)))\\
\ \wedge \quad \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l)))\\
\ \wedge \quad \text{lr-programs-properp}\,(l,\ table)\\
\ \wedge \quad (\text{p-psw}\,(\text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ pos),\ c)) = \texttt{'run})\\
\ \wedge \quad (\text{p-psw}\,(\text{lr-apply-subr}\,(l,\ \text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ pos),\ c)))\\
\qquad = \quad \texttt{'run}))\\
\rightarrow \quad \text{proper-p-statep}\,(\text{lr->p}\,(\text{lr-apply-subr}\,(l,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{lr-eval}\,(\texttt{'list},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{lr-set-pos}\,(l,\ pos),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad c))))
\end{array}
$$

THEOREM: cdr-untag-lr-p-pc-lr-funcall

$$
\begin{array}{l}
(\text{p-psw}\,(\text{lr-funcall}\,(l,\ new\text{-}l)) = \texttt{'run})\\
\rightarrow \quad (\text{cdr}\,(\text{untag}\,(\text{lr-p-pc}\,(\text{lr-funcall}\,(l,\ new\text{-}l)))) = \texttt{0})
\end{array}
$$

THEOREM: listp-p-ctrl-stk-lr-funcall

$$
\text{listp}\,(\text{p-ctrl-stk}\,(new\text{-}l)) \rightarrow \text{listp}\,(\text{p-ctrl-stk}\,(\text{lr-funcall}\,(l,\ new\text{-}l)))
$$

THEOREM: proper-p-framep-top-p-ctrl-stk-lr-funcall
$$
\begin{aligned}
&(\text{p-psw}\,(\text{lr-funcall}\,(l,\ \textit{new-l})) = \texttt{'run})\\
\rightarrow\quad &(\text{listp}\,(\text{car}\,(\text{p-ctrl-stk}\,(\text{lr-funcall}\,(l,\ \textit{new-l}))))\\
&\quad \wedge\quad \text{listp}\,(\text{cdr}\,(\text{car}\,(\text{p-ctrl-stk}\,(\text{lr-funcall}\,(l,\ \textit{new-l})))))\\
&\quad \wedge\quad (\text{cddr}\,(\text{car}\,(\text{p-ctrl-stk}\,(\text{lr-funcall}\,(l,\ \textit{new-l})))) = \textbf{nil})\\
&\quad \wedge\quad (\text{ret-pc}\,(\text{car}\,(\text{p-ctrl-stk}\,(\text{lr-funcall}\,(l,\ \textit{new-l}))))\\
&\qquad\quad = \quad \text{add-addr}\,(\text{lr-return-pc}\,(l),\ \texttt{1})))
\end{aligned}
$$

THEOREM: car-untag-p-pc-lr-funcall
$$
\begin{aligned}
&(\text{p-psw}\,(\text{lr-funcall}\,(l,\ \textit{new-l})) = \texttt{'run})\\
\rightarrow\quad &(\text{car}\,(\text{untag}\,(\text{p-pc}\,(\text{lr-funcall}\,(l,\ \textit{new-l}))))\\
&\quad = \quad \text{user-fname}\,(\text{car}\,(\text{lr-expr}\,(l))))
\end{aligned}
$$

THEOREM: area-name-p-pc-lr-funcall
$$
\begin{aligned}
&(\text{p-psw}\,(\text{lr-funcall}\,(l,\ \textit{new-l})) = \texttt{'run})\\
\rightarrow\quad &(\text{area-name}\,(\text{p-pc}\,(\text{lr-funcall}\,(l,\ \textit{new-l}))) = \text{user-fname}\,(\text{car}\,(\text{lr-expr}\,(l))))
\end{aligned}
$$

THEOREM: strip-cars-bindings-top-p-ctrl-stk-lr-funcall
$$
\begin{aligned}
&(\text{p-psw}\,(\text{lr-funcall}\,(l,\ \textit{new-l})) = \texttt{'run})\\
\rightarrow\quad &(\text{strip-cars}\,(\text{bindings}\,(\text{car}\,(\text{p-ctrl-stk}\,(\text{lr-funcall}\,(l,\ \textit{new-l})))))\\
&\quad = \quad \text{append}\,(\text{formal-vars}\,(\text{assoc}\,(\text{user-fname}\,(\text{car}\,(\text{lr-expr}\,(l))),\\
&\qquad\qquad\qquad\qquad \text{p-prog-segment}\,(l))),\\
&\qquad\qquad\quad \text{strip-cars}\,(\text{temp-var-dcls}\,(\text{assoc}\,(\text{user-fname}\,(\text{car}\,(\text{lr-expr}\,(l))),\\
&\qquad\qquad\qquad\qquad\qquad \text{p-prog-segment}\,(l))))))
\end{aligned}
$$

THEOREM: formal-vars-assoc-comp-programs-lr-programs-properp
$$
\begin{aligned}
&(\text{definedp}\,(\textit{name},\ \text{cdr}\,(\text{p-prog-segment}\,(l))) \wedge \text{lr-programs-properp}\,(l,\ \textit{table}))\\
\rightarrow\quad &(\text{formal-vars}\,(\text{assoc}\,(\textit{name},\ \text{comp-programs}\,(\text{p-prog-segment}\,(l))))\\
&\quad = \quad \text{formal-vars}\,(\text{assoc}\,(\textit{name},\ \text{p-prog-segment}\,(l))))
\end{aligned}
$$

THEOREM: temp-var-dcls-assoc-comp-programs-lr-programs-properp
$$
\begin{aligned}
&(\text{definedp}\,(\textit{name},\ \text{cdr}\,(\text{p-prog-segment}\,(l))) \wedge \text{lr-programs-properp}\,(l,\ \textit{table}))\\
\rightarrow\quad &(\text{temp-var-dcls}\,(\text{assoc}\,(\textit{name},\ \text{comp-programs}\,(\text{p-prog-segment}\,(l))))\\
&\quad = \quad \text{temp-var-dcls}\,(\text{assoc}\,(\textit{name},\ \text{p-prog-segment}\,(l))))
\end{aligned}
$$

THEOREM: definedp-comp-programs-definedp-lr-programs-properp
$$
\begin{aligned}
&(\text{definedp}\,(\textit{name},\ \text{cdr}\,(\text{p-prog-segment}\,(l))) \wedge \text{lr-programs-properp}\,(l,\ \textit{table}))\\
\rightarrow\quad &\text{definedp}\,(\textit{name},\ \text{comp-programs}\,(\text{p-prog-segment}\,(l)))
\end{aligned}
$$

THEOREM: definedp-lr-funcall-prog-segment
$$
\begin{aligned}
&(\text{listp}\,(\text{lr-expr}\,(l))\\
\wedge\quad &(\neg\ \text{subrp}\,(\text{car}\,(\text{lr-expr}\,(l))))\\
\wedge\quad &(\text{car}\,(\text{lr-expr}\,(l)) \neq \texttt{'quote})\\
\wedge\quad &\text{litatom}\,(\text{car}\,(\text{lr-expr}\,(l)))\\
\wedge\quad &\text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l)))
\end{aligned}
$$

$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (*progs* = cdr (p-prog-segment (l))))
$\rightarrow$ definedp (user-fname (car (lr-expr (l))), *progs*)

THEOREM: pop-p-ctrl-stk-lr-funcall
(p-psw (lr-funcall (l, *new-l*)) = `'run`)
$\rightarrow$ (cdr (p-ctrl-stk (lr-funcall (l, *new-l*))) = p-ctrl-stk (*new-l*))

THEOREM: proper-p-alistp-lr-funcall
(lr-programs-properp (l, *table*)
$\wedge$ definedp (user-fname (car (lr-expr (l))), cdr (p-prog-segment (*new-l*)))
$\wedge$ proper-p-prog-segmentp (comp-programs (p-prog-segment (*new-l*)),
lr->p (*new-l*))
$\wedge$ proper-p-temp-stkp (p-temp-stk (*new-l*), lr->p (*new-l*))
$\wedge$ (p-psw (lr-funcall (l, *new-l*)) = `'run`)
$\wedge$ (p-prog-segment (l) = p-prog-segment (*new-l*)))
$\rightarrow$ proper-p-alistp (bindings (car (p-ctrl-stk (lr-funcall (l, *new-l*)))),
lr->p (lr-funcall (l, *new-l*)))

THEOREM: proper-p-ctrl-stkp-lr-funcall
(proper-p-ctrl-stkp (cdr (*ctrl-stk*),
area-name (ret-pc (car (*ctrl-stk*))),
lr->p (*new-l*))
$\wedge$ proper-p-framep (top (*ctrl-stk*), *name*, lr->p (*new-l*))
$\wedge$ listp (*ctrl-stk*))
$\rightarrow$ proper-p-ctrl-stkp (*ctrl-stk*, *name*, lr->p (lr-funcall (l, *new-l*)))

THEOREM: not-lessp-p-max-ctrl-stk-size-lr-funcall
(listp (lr-expr (l))
$\wedge$ ($\neg$ subrp (car (lr-expr (l))))
$\wedge$ (car (lr-expr (l)) $\neq$ `'quote`)
$\wedge$ litatom (car (lr-expr (l)))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (lr-funcall (l, *new-l*)) = `'run`)
$\wedge$ (p-max-ctrl-stk-size (l) = p-max-ctrl-stk-size (*new-l*))
$\wedge$ (p-prog-segment (l) = p-prog-segment (*new-l*)))
$\rightarrow$ (p-max-ctrl-stk-size (l)
$\not<$ p-ctrl-stk-size (p-ctrl-stk (lr-funcall (l, *new-l*))))

THEOREM: offset-p-pc-lr-funcall
(p-psw (lr-funcall (l, *new-l*)) = `'run`)
$\rightarrow$ (offset (p-pc (lr-funcall (l, *new-l*))) = **nil**)

THEOREM: lr-eval-t-lr-funcall-p-psw-run
(p-psw (lr-eval (`t`, lr-funcall (l, *new-l*), c)) = `'run`)
$\rightarrow$ (p-psw (lr-funcall (l, *new-l*)) = `'run`)

THEOREM: proper-p-temp-stkp-lr-funcall
$(\text{listp}(\text{lr-expr}(l))$
$\land\ (\neg\ \text{subrp}(\text{car}(\text{lr-expr}(l))))$
$\land\ (\text{car}(\text{lr-expr}(l)) \neq \texttt{'quote})$
$\land\ \text{litatom}(\text{car}(\text{lr-expr}(l)))$
$\land\ \text{good-posp1}(\text{offset}(\text{p-pc}(l)),\ \text{program-body}(\text{p-current-program}(l)))$
$\land\ \text{lr-programs-properp}(l,\ \textit{table})$
$\land\ (\text{p-psw}(\text{lr-funcall}(l,\ \textit{new-l})) = \texttt{'run})$
$\land\ \text{proper-p-temp-stkp}(\text{p-temp-stk}(\textit{new-l}),\ \text{lr->p}(\textit{new-l}))$
$\land\ (\text{p-prog-segment}(l) = \text{p-prog-segment}(\textit{new-l})))$
$\to\ \text{proper-p-temp-stkp}(\text{p-temp-stk}(\text{lr-funcall}(l,\ \textit{new-l})),$
$\text{lr->p}(\text{lr-funcall}(l,\ \textit{new-l})))$

THEOREM: popn-nlistp
$(\neg\ \text{listp}(x)) \to (\neg\ \text{listp}(\text{popn}(n,\ x)))$

THEOREM: length-popn-lessp-fact
$\text{length}(\textit{list}) \not< \text{length}(\text{popn}(n,\ \textit{list}))$

EVENT: Disable popn-nlistp.

THEOREM: not-lessp-p-max-temp-stk-size-lr-funcall
$((\text{p-max-temp-stk-size}(l) \not< \text{length}(\text{p-temp-stk}(\textit{new-l})))$
$\land\ (\text{p-max-temp-stk-size}(l) = \text{p-max-temp-stk-size}(\textit{new-l})))$
$\to\ (\text{p-max-temp-stk-size}(l) \not< \text{length}(\text{p-temp-stk}(\text{lr-funcall}(l,\ \textit{new-l}))))$

THEOREM: listp-label-instrs
$\text{listp}(\text{label-instrs}(\textit{list},\ n)) = \text{listp}(\textit{list})$

THEOREM: listp-comp-body
$\text{listp}(\text{comp-body}(\textit{body}))$

THEOREM: lessp-offset-lr-return-pc-lr-p-c-size-good-posp
$(\text{good-posp1}(\text{offset}(\text{p-pc}(l)),\ \text{program-body}(\text{p-current-program}(l)))$
$\land\ \text{lr-programs-properp}(l,\ \textit{table})$
$\land\ (\text{subrp}(\text{car}(\text{lr-expr}(l))) \lor \text{litatom}(\text{car}(\text{lr-expr}(l))))$
$\land\ (\text{car}(\text{lr-expr}(l)) \neq \texttt{'if})$
$\land\ (\text{car}(\text{lr-expr}(l)) \neq \texttt{'quote}))$
$\to\ ((1 + \text{offset}(\text{lr-return-pc}(l)))$
$<\ \text{length}(\text{program-body}(\text{assoc}(\text{area-name}(\text{p-pc}(l)),$
$\text{comp-programs}(\text{p-prog-segment}(l))))))$

THEOREM: proper-p-statep-lr-funcall
$(\text{proper-p-statep}(\text{lr->p}(\text{lr-eval}(\texttt{'list},\ \text{lr-set-pos}(l,\ \textit{pos}),\ c)))$
$\land\ \text{listp}(\text{lr-expr}(l))$

$\land$ $(\neg\ \text{subrp}\,(\text{car}\,(\text{lr-expr}\,(l))))$
$\land$ $(\text{car}\,(\text{lr-expr}\,(l)) \neq \texttt{'quote})$
$\land$ $(\text{car}\,(\text{lr-expr}\,(l)) \neq \texttt{'if})$
$\land$ $\text{litatom}\,(\text{car}\,(\text{lr-expr}\,(l)))$
$\land$ $\text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l)))$
$\land$ $\text{lr-programs-properp}\,(l,\ \textit{table})$
$\land$ $(\text{p-psw}\,(\text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ \textit{pos}),\ c)) = \texttt{'run})$
$\land$ $(\text{p-psw}\,(\text{lr-funcall}\,(l,\ \text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ \textit{pos}),\ c))) = \texttt{'run}))$
$\rightarrow$ $\text{proper-p-statep}\,(\text{lr->p}\,(\text{lr-funcall}\,(l,$
$\qquad\qquad \text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ \textit{pos}),\ c))))$

THEOREM: proper-p-statep-lr-set-expr-lr-pop-cstk
let *l2* **be** $\text{lr-eval}\,(\mathbf{t},\ \text{lr-funcall}\,(l,\ \textit{new-l}),\ c - 1)$
in
$(\text{definedp}\,(\text{area-name}\,(\text{p-pc}\,(l)),\ \text{p-prog-segment}\,(l))$
$\land$ $(\text{cdr}\,(\text{p-ctrl-stk}\,(\textit{l2})) = \text{p-ctrl-stk}\,(\textit{new-l}))$
$\land$ $(\text{cdr}\,(\text{p-ctrl-stk}\,(\textit{new-l})) = \text{cdr}\,(\text{p-ctrl-stk}\,(l)))$
$\land$ $(\text{strip-cars}\,(\text{bindings}\,(\text{car}\,(\text{p-ctrl-stk}\,(\textit{new-l}))))$
$\qquad = \ \text{strip-cars}\,(\text{bindings}\,(\text{car}\,(\text{p-ctrl-stk}\,(l)))))$
$\land$ $\text{proper-p-statep}\,(\text{lr->p}\,(\textit{l2}))$
$\land$ $\text{proper-p-statep}\,(\text{lr->p}\,(\textit{new-l}))$
$\land$ $(\text{p-psw}\,(\textit{l2}) = \texttt{'run})$
$\land$ $\text{same-signature}\,(\text{p-data-segment}\,(\textit{new-l}),\ \text{p-data-segment}\,(\textit{l2}))$
$\land$ $(\text{p-prog-segment}\,(\textit{new-l}) = \text{p-prog-segment}\,(l))$
$\land$ $(\text{area-name}\,(\text{p-pc}\,(\textit{new-l})) = \text{area-name}\,(\text{p-pc}\,(l)))$
$\land$ $(\textit{pos} = \text{offset}\,(\text{p-pc}\,(l))))$
$\rightarrow$ $\text{proper-p-statep}\,(\text{lr->p}\,(\text{lr-set-expr}\,(\text{lr-pop-cstk}\,(\textit{l2}),\ l,\ \textit{pos})))$ **endlet**

THEOREM: p-psw-lr-eval-flag-list-flag-t
$((\text{p-psw}\,(\text{lr-eval}\,(\texttt{'list},\ \text{lr-set-expr}\,(\text{lr-eval}\,(\mathbf{t},\ l,\ c),\ l,\ \textit{pos}),\ c)) = \texttt{'run})$
$\land$ $\text{listp}\,(\text{offset}\,(\text{p-pc}\,(l)))$
$\land$ $\text{listp}\,(\text{lr-expr-list}\,(l)))$
$\rightarrow$ $(\text{p-psw}\,(\text{lr-eval}\,(\mathbf{t},\ l,\ c)) = \texttt{'run})$

THEOREM: lr-programs-properp-lr-set-expr
$\text{lr-programs-properp}\,(\text{lr-set-expr}\,(\textit{l1},\ \textit{l2},\ \textit{pos}),\ \textit{table})$
$=$ $\text{lr-programs-properp}\,(\textit{l2},\ \textit{table})$

THEOREM: lr-programs-properp-lr-pop-tstk
$\text{lr-programs-properp}\,(\text{lr-pop-tstk}\,(l),\ \textit{table}) = \text{lr-programs-properp}\,(l,\ \textit{table})$

THEOREM: lr-programs-properp-lr-funcall
$(\text{listp}\,(\text{lr-expr}\,(l))$
$\land$ $(\neg\ \text{subrp}\,(\text{car}\,(\text{lr-expr}\,(l))))$
$\land$ $(\text{car}\,(\text{lr-expr}\,(l)) \neq \texttt{'quote})$

$\wedge$ litatom (car (lr-expr (l)))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*))
$\rightarrow$ lr-programs-properp (lr-funcall (l, lr-eval (`'list`, lr-set-pos (l, *pos*), c)),
table)

Theorem: proper-p-statep-lr->p-lr-set-expr
(lr-programs-properp (*l2*, *table*)
$\wedge$ lr-programs-properp (*l1*, *table*)
$\wedge$ proper-p-statep (lr->p (*l2*))
$\wedge$ proper-p-statep (lr->p (*l1*))
$\wedge$ (cdr (p-ctrl-stk (*l1*)) = cdr (p-ctrl-stk (*l2*)))
$\wedge$ (strip-cars (bindings (car (p-ctrl-stk (*l1*))))
= strip-cars (bindings (car (p-ctrl-stk (*l2*)))))
$\wedge$ (p-prog-segment (*l1*) = p-prog-segment (*l2*))
$\wedge$ (p-word-size (*l1*) = p-word-size (*l2*))
$\wedge$ (p-max-ctrl-stk-size (*l1*) = p-max-ctrl-stk-size (*l2*))
$\wedge$ (p-max-temp-stk-size (*l1*) = p-max-temp-stk-size (*l2*)))
$\rightarrow$ proper-p-statep (lr->p (lr-set-expr (*l1*, *l2*, *pos*)))

Theorem: lr-programs-properp-lr-if-ok
lr-programs-properp (lr-if-ok (l), *table*) = lr-programs-properp (l, *table*)

Theorem: proper-p-statep-lr-if-ok
proper-p-statep (lr->p (lr-if-ok (l))) = proper-p-statep (lr->p (l))

Theorem: lr-eval-preserves-proper-p-statep-lr->p
(proper-p-statep (lr->p (l))
$\wedge$ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (lr-eval (*flag*, l, c)) = **`'run`**))
$\rightarrow$ (proper-p-statep (lr->p (lr-eval (*flag*, l, c)))
$\wedge$ (strip-cars (bindings (car (p-ctrl-stk (lr-eval (*flag*, l, c)))))
= strip-cars (bindings (car (p-ctrl-stk (l)))))
$\wedge$ (cdr (p-ctrl-stk (lr-eval (*flag*, l, c))) = cdr (p-ctrl-stk (l)))
$\wedge$ ((*flag* = `'list`) $\vee$ listp (p-temp-stk (lr-eval (*flag*, l, c))))
$\wedge$ same-signature (p-data-segment (l),
p-data-segment (lr-eval (*flag*, l, c))))

Theorem: lr-params-lr-eval
lr-params (*frame*, lr-eval (*flag*, l, c)) = lr-params (*frame*, l)

Theorem: lr-temps-lr-eval
lr-temps (*frame*, lr-eval (*flag*, l, c)) = lr-temps (*frame*, l)

```
;; Later LR-FREE-LIST-NODES will filter out those nodes that are
;; part of for example pack's or user-defined types that are larger than one
;; node (i.e. have more than two accessors).
```

DEFINITION:
lr-free-list-nodes (*addr*, *data-seg*)
= **if** offset (*addr*) < LR-NODE-SIZE **then nil**
  else let *sub-addr* **be** sub-addr (*addr*, LR-NODE-SIZE)
    in
    if type (fetch (add-addr (*sub-addr*, LR-REF-COUNT-OFFSET),
                data-seg))
      = `'addr`
    then cons (*sub-addr*,
              lr-free-list-nodes (*sub-addr*, *data-seg*))
    else lr-free-list-nodes (*sub-addr*, *data-seg*) **endif endlet endif**

THEOREM: length-delete-member
(*addr* ∈ *node-list*)
→ (length (delete (*addr*, *node-list*)) = (length (*node-list*) − 1))

```
;; Returns smallest address such that the address is too large to be
;; a pointer to a node in DATA-SEG.
```

DEFINITION:
lr-max-node (*data-seg*)
= tag (`'addr`,
     cons (LR-HEAP-NAME, length (value (LR-HEAP-NAME, *data-seg*)) − 1))

DEFINITION:
lr-check-free-nodes (*addr*, *node-list*, *data-seg*, *max-addr*)
= **if** *addr* ∈ *node-list*
  then lr-check-free-nodes (fetch (add-addr (*addr*, LR-REF-COUNT-OFFSET),
                    data-seg),
                  delete (*addr*, *node-list*),
                  data-seg,
                  max-addr)
  else *addr* = *max-addr* **endif**

DEFINITION:
lr-proper-free-listp (*data-seg*)
= (adpp (untag (LR-FP-ADDR), *data-seg*)
  ∧ lr-check-free-nodes (lr-fetch-fp (*data-seg*),
                lr-free-list-nodes (lr-max-node (*data-seg*),
                          data-seg),
                data-seg,
                lr-max-node (*data-seg*)))

EVENT: Disable lr-proper-free-listp.

DEFINITION:
lr-check-f-addrp $(addr,\ data\text{-}seg) = (addr = \text{LR-F-ADDR})$

EVENT: Disable lr-check-f-addrp.

DEFINITION:
lr-check-undef-addrp $(addr,\ data\text{-}seg) = (addr = \text{LR-UNDEF-ADDR})$

EVENT: Disable lr-check-undef-addrp.

DEFINITION:
lr-check-numberp-addrp $(addr,\ data\text{-}seg)$

$$
\begin{aligned}
= \quad & ((\text{type}(\text{fetch}(\text{add-addr}(addr,\ \text{LR-UNBOX-NAT-OFFSET}),\ data\text{-}seg)) = \texttt{'nat}) \\
& \wedge \quad \text{lr-good-pointerp}(\text{fetch}(\text{add-addr}(addr,\ 3),\ data\text{-}seg),\ data\text{-}seg) \\
& \wedge \quad (\text{untag}(\text{fetch}(\text{add-addr}(addr,\ \text{LR-UNBOX-NAT-OFFSET}),\ data\text{-}seg)) \in \mathbf{N}))
\end{aligned}
$$

EVENT: Disable lr-check-numberp-addrp.

DEFINITION:
lr-check-listp-addrp $(addr,\ data\text{-}seg)$

$$
\begin{aligned}
= \quad & (\text{lr-good-pointerp}(\text{fetch}(\text{add-addr}(addr,\ \text{LR-CAR-OFFSET}),\ data\text{-}seg), \\
& \qquad\qquad data\text{-}seg) \\
& \wedge \quad \text{lr-good-pointerp}(\text{fetch}(\text{add-addr}(addr,\ \text{LR-CDR-OFFSET}),\ data\text{-}seg), \\
& \qquad\qquad\qquad data\text{-}seg))
\end{aligned}
$$

EVENT: Disable lr-check-listp-addrp.

DEFINITION:
lr-proper-heapp-nodep $(addr,\ data\text{-}seg)$

$$
\begin{aligned}
= \quad & \textbf{if } \neg\ \text{lr-nodep}(addr,\ data\text{-}seg) \ \textbf{then f} \\
& \textbf{elseif } \text{type}(\text{fetch}(\text{add-addr}(addr,\ \text{LR-REF-COUNT-OFFSET}),\ data\text{-}seg)) \\
& \qquad = \quad \texttt{'addr} \\
& \textbf{then } \text{offset}(addr) \not< (\text{LR-NODE-SIZE} + \text{offset}(\text{LR-F-ADDR})) \\
& \textbf{elseif } \text{type}(\text{fetch}(\text{add-addr}(addr,\ \text{LR-REF-COUNT-OFFSET}),\ data\text{-}seg)) \\
& \qquad \neq \quad \texttt{'nat} \ \textbf{then f} \\
& \textbf{elseif } \text{type}(\text{fetch}(addr,\ data\text{-}seg)) \neq \texttt{'nat} \ \textbf{then f} \\
& \textbf{elseif } \text{untag}(\text{fetch}(addr,\ data\text{-}seg)) = \text{LR-UNDEFINED-TAG} \\
& \textbf{then } \text{lr-check-undef-addrp}(addr,\ data\text{-}seg) \\
& \textbf{elseif } \text{untag}(\text{fetch}(addr,\ data\text{-}seg)) = \text{LR-FALSE-TAG} \\
& \textbf{then } \text{lr-check-f-addrp}(addr,\ data\text{-}seg)
\end{aligned}
$$

elseif offset ($addr$) $<$ offset (LR-T-ADDR) **then f**
elseif untag (fetch ($addr$, $data\text{-}seg$)) $=$ LR-TRUE-TAG **then t**
elseif untag (fetch ($addr$, $data\text{-}seg$)) $=$ LR-ADD1-TAG
then lr-check-numberp-addrp ($addr$, $data\text{-}seg$)
elseif untag (fetch ($addr$, $data\text{-}seg$)) $=$ LR-CONS-TAG
then lr-check-listp-addrp ($addr$, $data\text{-}seg$)
else f endif

EVENT: Disable lr-proper-heapp-nodep.

DEFINITION:
lr-proper-heapp2 ($addr$, $data\text{-}seg$)
$=$ **if** offset ($addr$) $<$ LR-NODE-SIZE **then t**
else let $sub\text{-}addr$ **be** sub-addr ($addr$, LR-NODE-SIZE)
in
lr-proper-heapp-nodep ($sub\text{-}addr$, $data\text{-}seg$)
$\land$ lr-proper-heapp2 ($sub\text{-}addr$, $data\text{-}seg$) **endlet endif**

DEFINITION:
lr-valp ($value$, $addr$, $data\text{-}seg$)
$=$ **if** lr-good-pointerp ($addr$, $data\text{-}seg$)
then let tag **be** untag (fetch ($addr$, $data\text{-}seg$))
in
if listp ($value$)
then (tag $=$ LR-CONS-TAG)
$\land$ lr-valp (car ($value$),
fetch (add-addr ($addr$, LR-CAR-OFFSET),
$data\text{-}seg$),
$data\text{-}seg$)
$\land$ lr-valp (cdr ($value$),
fetch (add-addr ($addr$, LR-CDR-OFFSET),
$data\text{-}seg$),
$data\text{-}seg$)
elseif truep ($value$) **then** tag $=$ LR-TRUE-TAG
elseif falsep ($value$) **then** tag $=$ LR-FALSE-TAG
elseif $value \in \mathbf{N}$
then (tag $=$ LR-ADD1-TAG)
$\land$ ($value$ $=$ untag (fetch (add-addr ($addr$,
LR-UNBOX-NAT-OFFSET),
$data\text{-}seg$)))
else f endif endlet
else f endif

DEFINITION:

lr-proper-heapp1 (*addr*, *data-seg*)
= (lr-proper-heapp2 (*addr*, *data-seg*)
  ∧ lr-valp (**t**, LR-T-ADDR, *data-seg*)
  ∧ lr-valp (0, LR-0-ADDR, *data-seg*))

EVENT: Disable lr-proper-heapp1.

```
;; This is the minimum heap that allows all the predefineds to be defined.
```

DEFINITION:
lr-minimum-heapp (*data-seg*)
= (adpp (untag (LR-UNDEF-ADDR), *data-seg*)
  ∧ adpp (untag (LR-F-ADDR), *data-seg*)
  ∧ adpp (untag (LR-T-ADDR), *data-seg*)
  ∧ adpp (untag (LR-0-ADDR), *data-seg*)
  ∧ adpp (untag (add-addr (LR-0-ADDR, LR-NODE-SIZE)), *data-seg*))

EVENT: Disable lr-minimum-heapp.

```
;; This needs to be augmented to test that the word-size is big enough to
;; hold piton tags.
```

DEFINITION:
lr-proper-heapp (*data-seg*)
= (lr-minimum-heapp (*data-seg*)
  ∧ lr-nodep (lr-max-node (*data-seg*), *data-seg*)
  ∧ lr-proper-free-listp (*data-seg*)
  ∧ lr-proper-heapp1 (lr-max-node (*data-seg*), *data-seg*))

EVENT: Disable lr-proper-heapp.

DEFINITION:
lr-check-result1 (*value*, *temp-stk*, *data-seg*)
= **if** listp (*value*)
  then lr-valp (car (*value*), top (*temp-stk*), *data-seg*)
    ∧ lr-check-result1 (cdr (*value*), pop (*temp-stk*), *data-seg*)
  else t endif

DEFINITION:
lr-check-result (*flag*, *value*, *temp-stk*, *data-seg*, *orig-temp-stk*)
= ((*orig-temp-stk* = **if** *flag* = 'list
      then restn (length (*value*), *temp-stk*)
      else cdr (*temp-stk*) **endif**)
  ∧ **if** *flag* = 'list

then lr-check-result1 (reverse (*value*), *temp-stk*, *data-seg*)
else lr-valp (*value*, top (*temp-stk*), *data-seg*) **endif**
$\wedge$ lr-proper-heapp (*data-seg*))

EVENT: Disable lr-check-result.

DEFINITION:
lr-s-similar-params (*s-params*, *lr-params*, *data-seg*)
= **if** listp (*s-params*)
then if listp (*lr-params*)
then (caar (*s-params*) = caar (*lr-params*))
$\wedge$ lr-valp (cdar (*s-params*), cdar (*lr-params*), *data-seg*)
$\wedge$ lr-s-similar-params (cdr (*s-params*),
cdr (*lr-params*),
data-seg)
else f endif
else *lr-params* $\simeq$ **nil endif**

DEFINITION:
lr-s-similar-temps (*s-temps*, *lr-temps*, *data-seg*)
= **if** listp (*s-temps*)
then if listp (*lr-temps*)
then if cdar (*lr-temps*) = LR-UNDEF-ADDR **then** $\neg$ cadar (*s-temps*)
else cadar (*s-temps*)
$\wedge$ lr-valp (caddar (*s-temps*),
cdar (*lr-temps*),
data-seg) **endif**
$\wedge$ lr-s-similar-temps (cdr (*s-temps*),
cdr (*lr-temps*),
data-seg)
else f endif
else *lr-temps* $\simeq$ **nil endif**

DEFINITION:
lr-s-similar-const-table (*table*, *data-seg*)
= **if** listp (*table*)
then lr-valp (caar (*table*), cdar (*table*), *data-seg*)
$\wedge$ lr-s-similar-const-table (cdr (*table*), *data-seg*)
else t endif

DEFINITION:
lr-s-similar-statesp (*s-params*, *s-temps*, *l*, *table*)
= (lr-s-similar-params (*s-params*,
lr-params (top (p-ctrl-stk (*l*)), *l*),

p-data-segment (l))

$\wedge$ lr-s-similar-temps ($s\text{-}temps$,
lr-temps (top (p-ctrl-stk (l)), l),
p-data-segment (l))

$\wedge$ lr-s-similar-const-table ($table$, p-data-segment (l)))

EVENT: Disable lr-s-similar-statesp.

THEOREM: p-accessors-s->lr1

(p-pc (s->lr1 (s, l, $table$)) = tag ('`pc`, cons (s-pname (s), s-pos (s))))

$\wedge$ (p-ctrl-stk (s->lr1 (s, l, $table$)) = p-ctrl-stk (l))

$\wedge$ (p-temp-stk (s->lr1 (s, l, $table$)) = p-temp-stk (l))

$\wedge$ (p-prog-segment (s->lr1 (s, l, $table$))
= lr-compile-programs (s-progs (s), $table$))

$\wedge$ (p-data-segment (s->lr1 (s, l, $table$)) = p-data-segment (l))

$\wedge$ (p-max-ctrl-stk-size (s->lr1 (s, l, $table$)) = p-max-ctrl-stk-size (l))

$\wedge$ (p-max-temp-stk-size (s->lr1 (s, l, $table$)) = p-max-temp-stk-size (l))

$\wedge$ (p-word-size (s->lr1 (s, l, $table$)) = p-word-size (l))

$\wedge$ (p-psw (s->lr1 (s, l, $table$)) = s-err-flag (s))

THEOREM: s-eval-err-flag-not-run-fact

(s-err-flag (s) $\neq$ '**run**) $\rightarrow$ (s-eval ($flag$, s, $clock$) = s)

`;; OFFSET`

THEOREM: offset-tag-cons

offset (tag (tag, cons ($area$, $offset$))) = $offset$

`;; ADP-NAME`

THEOREM: adp-name-cons

adp-name (cons (x, y)) = x

`;; OFFSET-SUB-ADDR -- see above`

`;; LR-PROPER-P-AREASP`

THEOREM: definedp-litatom-lr-proper-p-areas

(($\neg$ litatom ($name$)) $\wedge$ lr-proper-p-areasp ($data\text{-}seg$))
$\rightarrow$ ($\neg$ definedp ($name$, $data\text{-}seg$))

EVENT: Disable definedp-litatom-lr-proper-p-areas.

THEOREM: member-lr-free-list-nodes-type-addr
(type (fetch (add-addr ($addr$, LR-REF-COUNT-OFFSET), $data\text{-}seg$)) $\neq$ `'addr`)
$\rightarrow$ ($addr \notin$ lr-free-list-nodes ($max\text{-}addr$, $data\text{-}seg$))

EVENT: Disable member-lr-free-list-nodes-type-addr.

THEOREM: lessp-length-deposit
length (cdr (assoc ($name$, deposit (any, $addr$, $data\text{-}seg$))))
$\not<$ length (cdr (assoc ($name$, $data\text{-}seg$)))

`;; GET`

THEOREM: definedp-listp-cdr-assoc-lr-proper-p-areasp
lr-proper-p-areasp ($data\text{-}seg$)
$\rightarrow$ (listp (cdr (assoc ($area\text{-}name$, $data\text{-}seg$)))
= definedp ($area\text{-}name$, $data\text{-}seg$))

EVENT: Disable definedp-listp-cdr-assoc-lr-proper-p-areasp.

`;; LR-MINIMUM-HEAPP`

THEOREM: lr-minimum-heapp-opener-adpp-lr-f-addr
lr-minimum-heapp ($data\text{-}seg$) $\rightarrow$ adpp (identity (untag (LR-F-ADDR)), $data\text{-}seg$)

THEOREM: lr-minimum-heapp-opener-adpp-lr-t-addr
lr-minimum-heapp ($data\text{-}seg$) $\rightarrow$ adpp (identity (untag (LR-T-ADDR)), $data\text{-}seg$)

THEOREM: lr-minimum-heapp-opener-adpp-lr-0-addr
lr-minimum-heapp ($data\text{-}seg$) $\rightarrow$ adpp (identity (untag (LR-0-ADDR)), $data\text{-}seg$)

THEOREM: lr-minimum-heapp-opener-adpp-lr-undef-addr
lr-minimum-heapp ($data\text{-}seg$)
$\rightarrow$ adpp (identity (untag (LR-UNDEF-ADDR)), $data\text{-}seg$)

THEOREM: lr-boundary-offsetp-sub1-length-heap-name
lr-boundary-nodep (lr-max-node ($data\text{-}seg$))
$\rightarrow$ lr-boundary-offsetp (length (cdr (assoc (identity (LR-HEAP-NAME), $data\text{-}seg$))) $-$ 1)

THEOREM: lessp-lr-boundary-offsetp-nodep-plus-node-size-fact-2
(lr-boundary-offsetp ($offset1$)
$\land$ lr-boundary-offsetp ($offset2$)
$\land$ ($n <$ LR-NODE-SIZE))
$\rightarrow$ ((($n + offset1$) $<$ $offset2$) = ($offset1 < offset2$))

THEOREM: lr-boundary-offsetp-times-lr-node-size-anything
lr-boundary-offsetp (identity (LR-NODE-SIZE) $*$ x)

THEOREM: lr-boundary-offsetp-difference-not-equal-lessp-fact-2
(lr-boundary-offsetp (x) ∧ lr-boundary-offsetp (y) ∧ $(x \in \mathbf{N})$ ∧ $(x < y)$)
→ $(((y -$ LR-NODE-SIZE$) < x) = \mathbf{f})$

THEOREM: lr-minimum-heapp-opener-2
lr-minimum-heapp (*data-seg*)
→ (identity (LR-MINIMUM-HEAP-SIZE)
  < length (cdr (assoc (identity (LR-HEAP-NAME), *data-seg*))))

EVENT: Disable lr-minimum-heapp-opener-2.

THEOREM: lr-minimum-heapp-opener-3
lr-minimum-heapp (*data-seg*) → definedp (identity (LR-HEAP-NAME), *data-seg*)

EVENT: Disable lr-minimum-heapp-opener-3.

```
;; LR-PROPER-FREE-LISTP
```

DEFINITION:
lr-node-listp (*list*, *data-seg*)
= **if** listp (*list*)
  then lr-nodep (car (*list*), *data-seg*)
    ∧ lr-node-listp (cdr (*list*), *data-seg*)
  else t endif

EVENT: Disable lr-node-listp.

THEOREM: adpp-adpp-sub-addr
adpp (untag (*addr*), *data-seg*) → adpp (untag (sub-addr (*addr*, *n*)), *data-seg*)

THEOREM: lr-node-listp-lr-free-list-nodes
(lr-boundary-nodep (*addr*)
∧ (area-name (*addr*) = LR-HEAP-NAME)
∧ adpp (untag (*addr*), *data-seg2*)
∧ (type (*addr*) = `'addr`))
→ lr-node-listp (lr-free-list-nodes (*addr*, *data-seg1*), *data-seg2*)

THEOREM: lr-nodep-member-lr-node-listp
(lr-node-listp (*list*, *data-seg*) ∧ (*node* ∈ *list*))
→ ((type (*node*) = `'addr`)
  ∧ (cddr (*node*) = **nil**)
  ∧ listp (*node*)
  ∧ adpp (untag (*node*), *data-seg*)
  ∧ lr-boundary-nodep (*node*)
  ∧ (area-name (*node*) = LR-HEAP-NAME))

EVENT: Disable lr-nodep-member-lr-node-listp.

THEOREM: lr-max-node-lr-nodep-opener-facts
(type (lr-max-node (*data-seg*)) = **'addr**)
∧ (cddr (lr-max-node (*data-seg*)) = **nil**)
∧ (area-name (lr-max-node (*data-seg*)) = LR-HEAP-NAME)

THEOREM: lr-max-node-adpp-definedp-lr-heap-name
lr-proper-p-areasp (*data-seg*)
→ (adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
  = definedp (LR-HEAP-NAME, *data-seg*))

THEOREM: offset-lr-max-node
offset (lr-max-node (*data-seg*))
= (length (cdr (assoc (identity (LR-HEAP-NAME), *data-seg*))) − 1)

EVENT: Disable lr-max-node.

THEOREM: lr-proper-free-listp-opener-1
lr-proper-free-listp (*data-seg*)
→ adpp (identity (untag (LR-FP-ADDR)), *data-seg*)

THEOREM: lr-proper-free-listp-opener-2
(lr-proper-free-listp (*data-seg*)
∧ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
∧ lr-boundary-nodep (lr-max-node (*data-seg*)))
→ ((type (fetch (identity (LR-FP-ADDR), *data-seg*)) = **'addr**)
  ∧ (cddr (fetch (identity (LR-FP-ADDR), *data-seg*)) = **nil**)
  ∧ listp (fetch (identity (LR-FP-ADDR), *data-seg*))
  ∧ adpp (untag (fetch (identity (LR-FP-ADDR), *data-seg*)), *data-seg*)
  ∧ lr-boundary-nodep (fetch (identity (LR-FP-ADDR), *data-seg*))
  ∧ (area-name (fetch (identity (LR-FP-ADDR), *data-seg*))
    = LR-HEAP-NAME))

THEOREM: lr-proper-free-listp-opener-2-adpp-untag-numberp-offset
(lr-proper-free-listp (*data-seg*)
∧ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
∧ lr-boundary-nodep (lr-max-node (*data-seg*)))
→ (offset (fetch (identity (LR-FP-ADDR), *data-seg*)) ∈ **N**)

THEOREM: lr-proper-free-listp-opener-2-adpp-untag-listp
(lr-proper-free-listp (*data-seg*)
∧ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
∧ lr-boundary-nodep (lr-max-node (*data-seg*)))
→ listp (untag (fetch (identity (LR-FP-ADDR), *data-seg*)))

THEOREM: plus-times-fact-1
$(n \not\simeq 0) \rightarrow (((n + (n * w)) < (d * n)) = ((1 + w) < d))$

EVENT: Disable plus-times-fact-1.

THEOREM: lessp-difference-fact-1
$(((x \ \mathbf{mod}\ n) = 0) \wedge ((y \ \mathbf{mod}\ n) = 0) \wedge (x < y) \wedge (x \in \mathbf{N}))$
$\rightarrow \quad ((x < (y - n)) = (x \neq (y - n)))$

THEOREM: lessp-difference-lr-boundary-offsetp-fact-1
$((\mathit{offset} \in \mathbf{N})$
$\wedge$ lr-boundary-offsetp (offset)
$\wedge$ lr-boundary-offsetp (y)
$\wedge$ $(\mathit{offset} < y))$
$\rightarrow$ $((\mathit{offset} < (y -$ identity (LR-NODE-SIZE)))
$=$ $(\mathit{offset} \neq (y -$ identity (LR-NODE-SIZE))))

THEOREM: lessp-lr-node-on-boundaryp-node-size
(lr-boundary-nodep (addr) $\wedge$ (offset $(\mathit{addr}) \in \mathbf{N}$))
$\rightarrow$ ((offset $(\mathit{addr}) <$ identity (LR-NODE-SIZE)) $=$ (offset $(\mathit{addr}) = 0$))

THEOREM: lessp-difference-node-size-sub-addr
((offset $(\mathit{addr}) <$ offset $(\mathit{max\text{-}addr})$)
$\wedge$ (area-name $(\mathit{addr}) =$ area-name $(\mathit{max\text{-}addr})$)
$\wedge$ lr-boundary-nodep $(\mathit{max\text{-}addr})$
$\wedge$ (type $(\mathit{max\text{-}addr}) =$ `'addr`)
$\wedge$ (offset $(\mathit{max\text{-}addr}) \in \mathbf{N}$)
$\wedge$ (cddr $(\mathit{max\text{-}addr}) =$ **nil**)
$\wedge$ lr-boundary-nodep (addr)
$\wedge$ (type $(\mathit{addr}) =$ `'addr`)
$\wedge$ (offset $(\mathit{addr}) \in \mathbf{N}$)
$\wedge$ (cddr $(\mathit{addr}) =$ **nil**)
$\wedge$ listp (untag (addr)))
$\rightarrow$ ((offset $(\mathit{addr}) <$ (offset $(\mathit{max\text{-}addr}) -$ identity (LR-NODE-SIZE)))
$=$ (sub-addr $(\mathit{max\text{-}addr},$ identity (LR-NODE-SIZE)) $\neq \mathit{addr}$))

THEOREM: lr-nodep-lr-proper-heapp-nodep
(lr-proper-heapp2 $(\mathit{max\text{-}addr}, \mathit{data\text{-}seg})$
$\wedge$ (offset $(\mathit{addr}) <$ offset $(\mathit{max\text{-}addr})$)
$\wedge$ lr-nodep $(\mathit{max\text{-}addr}, \mathit{data\text{-}seg})$
$\wedge$ lr-nodep $(\mathit{addr}, \mathit{data\text{-}seg})$)
$\rightarrow$ lr-proper-heapp-nodep $(\mathit{addr}, \mathit{data\text{-}seg})$

EVENT: Disable lessp-difference-node-size-sub-addr.

THEOREM: adpp-area-name-offset-same
$$
\begin{array}{ll}
& (\text{listp}(\text{untag}(\mathit{addr1})) \\
\land & (\text{offset}(\mathit{addr1}) \in \mathbf{N}) \\
\land & (\text{cddr}(\mathit{addr1}) = \mathbf{nil}) \\
\land & \text{listp}(\text{untag}(\mathit{addr2})) \\
\land & (\text{offset}(\mathit{addr2}) \in \mathbf{N}) \\
\land & (\text{cddr}(\mathit{addr2}) = \mathbf{nil}) \\
\land & (\text{type}(\mathit{addr1}) = \text{type}(\mathit{addr2}))) \\
\rightarrow & ((\mathit{addr1} = \mathit{addr2}) \\
& \quad = \quad ((\text{offset}(\mathit{addr1}) = \text{offset}(\mathit{addr2})) \\
& \qquad\quad \land \quad (\text{area-name}(\mathit{addr1}) = \text{area-name}(\mathit{addr2}))))
\end{array}
$$

THEOREM: lr-proper-heapp-nodep-tag-cons
$$
\begin{array}{ll}
& ((\text{untag}(\text{fetch}(\mathit{addr},\ \mathit{data\text{-}seg})) = \text{LR-CONS-TAG}) \\
\land & (\text{type}(\text{fetch}(\text{add-addr}(\mathit{addr},\ \text{LR-REF-COUNT-OFFSET}),\ \mathit{data\text{-}seg})) = \texttt{'nat}) \\
\land & \text{lr-proper-heapp-nodep}(\mathit{addr},\ \mathit{data\text{-}seg}) \\
\land & ((\mathit{offset} = \text{LR-CAR-OFFSET}) \lor (\mathit{offset} = \text{LR-CDR-OFFSET}))) \\
\rightarrow & \text{lr-good-pointerp}(\text{fetch}(\text{add-addr}(\mathit{addr},\ \mathit{offset}),\ \mathit{data\text{-}seg}),\ \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: adpp-add-addr-fact-2
$$
\begin{array}{ll}
& (\text{adpp}(\text{untag}(\mathit{addr1}),\ \mathit{data\text{-}seg}) \\
\land & \text{adpp}(\text{untag}(\text{add-addr}(\mathit{addr1},\ n)),\ \mathit{data\text{-}seg}) \\
\land & \text{adpp}(\text{untag}(\mathit{addr2}),\ \mathit{data\text{-}seg}) \\
\land & (\neg\ \text{adpp}(\text{untag}(\text{add-addr}(\mathit{addr2},\ n)),\ \mathit{data\text{-}seg})) \\
\land & (\text{area-name}(\mathit{addr1}) = \text{area-name}(\mathit{addr2}))) \\
\rightarrow & (\text{offset}(\mathit{addr1}) < \text{offset}(\mathit{addr2}))
\end{array}
$$

THEOREM: fetch-lr-nodep-add-addr
$$
\begin{array}{ll}
& ((\neg\ \text{adpp}(\text{untag}(\text{add-addr}(\mathit{addr},\ n)),\ \mathit{data\text{-}seg})) \land \text{lr-nodep}(\mathit{addr},\ \mathit{data\text{-}seg})) \\
\rightarrow & (\text{fetch}(\text{add-addr}(\mathit{addr},\ n),\ \mathit{data\text{-}seg}) = \texttt{0})
\end{array}
$$

EVENT: Disable fetch-lr-nodep-add-addr.

THEOREM: untag-addr-addr-tag
$$
\text{untag}(\text{add-addr}(\text{tag}(\mathit{tag},\ \mathit{adp}),\ n)) = \text{cons}(\text{car}(\mathit{adp}),\ \text{cdr}(\mathit{adp}) + n)
$$

THEOREM: lr-good-pointerp-lessp-offset-max-heap-node
$$
\begin{array}{ll}
& (\text{adpp}(\text{untag}(\mathit{addr}),\ \mathit{data\text{-}seg}) \\
\land & \text{lr-boundary-nodep}(\mathit{addr}) \\
\land & \text{listp}(\mathit{addr}) \\
\land & (\text{cddr}(\mathit{addr}) = \mathbf{nil}) \\
\land & (\text{type}(\mathit{addr}) = \texttt{'addr}) \\
\land & (\text{area-name}(\mathit{addr}) = \texttt{'heap}) \\
\land & (\text{type}(\text{fetch}(\text{add-addr}(\mathit{addr},\ \text{LR-REF-COUNT-OFFSET}),\ \mathit{data\text{-}seg})) = \texttt{'nat})
\end{array}
$$

$\wedge$ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ (offset (*addr*)
$<$ (length (cdr (assoc (identity (LR-HEAP-NAME), *data-seg*))) $-$ 1))

EVENT: Disable lr-good-pointerp-lessp-offset-max-heap-node.

THEOREM: lr-proper-heapp-opener-1
lr-proper-heapp (*data-seg*)
$\rightarrow$ (lr-minimum-heapp (*data-seg*) $\wedge$ lr-proper-free-listp (*data-seg*))

THEOREM: lr-proper-heapp-opener-3
((*addr* $=$ lr-max-node (*data-seg*)) $\wedge$ lr-proper-heapp (*data-seg*))
$\rightarrow$ lr-proper-heapp2 (*addr*, *data-seg*)

THEOREM: deposit-free-ptr-preserves-lr-valp
(adpp (untag (LR-FP-ADDR), *data-seg*) $\wedge$ lr-valp (*value*, *addr*, *data-seg*))
$\rightarrow$ lr-valp (*value*, *addr*, deposit (*anything*, identity (LR-FP-ADDR), *data-seg*))

THEOREM: lr-proper-p-areasp-deposit-anything-anywhere
lr-proper-p-areasp (*data-seg*)
$\rightarrow$ lr-proper-p-areasp (deposit (*anything*, *addr*, *data-seg*))

THEOREM: lr-node-listp-delete
lr-node-listp (*list*, *data-seg*)
$\rightarrow$ lr-node-listp (delete (*anything*, *list*), *data-seg*)

EVENT: Disable lr-node-listp-delete.

THEOREM: lr-node-listp-deposit-anything-at-all
lr-node-listp (*addr*, *data-seg*)
$\rightarrow$ lr-node-listp (*addr*, deposit (*anything*, *addr2*, *data-seg*))

EVENT: Disable lr-node-listp-deposit-anything-at-all.

THEOREM: cdr-assoc-member-strip-cdrs
definedp (*name*, *list*) $\rightarrow$ (cdr (assoc (*name*, *list*)) $\in$ strip-cdrs (*list*))

EVENT: Disable cdr-assoc-member-strip-cdrs.

THEOREM: lr-set-error-lr->p
lr->p (lr-set-error (*p*, *flag*)) $=$ lr-set-error (lr->p (*p*), *flag*)

THEOREM: lr-params-lr-set-expr
$((\text{area-name}(\text{p-pc}(l)) = \text{area-name}(\text{p-pc}(\mathit{l2})))$
$\land \quad (\text{p-prog-segment}(l) = \text{p-prog-segment}(\mathit{l2})))$
$\to \quad (\text{lr-params}(\mathit{frame}, \text{lr-set-expr}(l, \mathit{l2}, \mathit{pos})) = \text{lr-params}(\mathit{frame}, l))$

THEOREM: lr-temps-lr-set-expr
$((\text{area-name}(\text{p-pc}(l)) = \text{area-name}(\text{p-pc}(\mathit{l2})))$
$\land \quad (\text{p-prog-segment}(l) = \text{p-prog-segment}(\mathit{l2})))$
$\to \quad (\text{lr-temps}(\mathit{frame}, \text{lr-set-expr}(l, \mathit{l2}, \mathit{pos})) = \text{lr-temps}(\mathit{frame}, l))$

THEOREM: p-current-program-lr-push-tstk
$\text{p-current-program}(\text{lr-push-tstk}(l, \mathit{any})) = \text{p-current-program}(l)$

THEOREM: p-current-program-lr-set-temp
$\text{p-current-program}(\text{lr-set-temp}(l, \mathit{value}, \mathit{var})) = \text{p-current-program}(l)$

THEOREM: p-current-program-lr-pop-tstk
$\text{p-current-program}(\text{lr-pop-tstk}(l)) = \text{p-current-program}(l)$

THEOREM: p-current-program-lr-do-temp-fetch
$\text{p-current-program}(\text{lr-do-temp-fetch}(l)) = \text{p-current-program}(l)$

THEOREM: strip-cars-restn
$\text{strip-cars}(\text{restn}(n, \mathit{list})) = \text{restn}(n, \text{strip-cars}(\mathit{list}))$

EVENT: Disable strip-cars-restn.

THEOREM: strip-cars-firstn
$\text{strip-cars}(\text{firstn}(n, \mathit{list})) = \text{firstn}(n, \text{strip-cars}(\mathit{list}))$

EVENT: Disable strip-cars-firstn.

THEOREM: lr-params-lr-pop-tstk
$\text{lr-params}(\mathit{frame}, \text{lr-pop-tstk}(l)) = \text{lr-params}(\mathit{frame}, l)$

THEOREM: lr-temps-lr-pop-tstk
$\text{lr-temps}(\mathit{frame}, \text{lr-pop-tstk}(l)) = \text{lr-temps}(\mathit{frame}, l)$

THEOREM: lr-minimum-heapp-same-signature
$\text{same-signature}(\mathit{data\text{-}seg1}, \mathit{data\text{-}seg2})$
$\to \quad (\text{lr-minimum-heapp}(\mathit{data\text{-}seg2}) = \text{lr-minimum-heapp}(\mathit{data\text{-}seg1}))$

EVENT: Disable lr-minimum-heapp-same-signature.

THEOREM: put-not-listp
$((\neg \text{listp}(\mathit{list1})) \wedge (\neg \text{listp}(\mathit{list2})))$
$\rightarrow \quad (\text{put}(\mathit{val}, n, \mathit{list1}) = \text{put}(\mathit{val}, n, \mathit{list2}))$

THEOREM: put-zero
$\text{put}(\mathit{val}, n, \mathtt{0}) = \text{put}(\mathit{val}, n, \mathbf{nil})$

EVENT: Disable put-zero.

THEOREM: put-put
$((\mathit{offset1} \in \mathbf{N}) \wedge (\mathit{offset2} \in \mathbf{N}))$
$\rightarrow \quad (\text{put}(\mathit{val1}, \mathit{offset1}, \text{put}(\mathit{val2}, \mathit{offset2}, \mathit{list}))$
$\quad = \quad \mathbf{if}\ \mathit{offset1} = \mathit{offset2}\ \mathbf{then}\ \text{put}(\mathit{val1}, \mathit{offset1}, \mathit{list})$
$\quad\quad \mathbf{else}\ \text{put}(\mathit{val2}, \mathit{offset2}, \text{put}(\mathit{val1}, \mathit{offset1}, \mathit{list}))\ \mathbf{endif})$

THEOREM: proper-p-data-segmentp-implies-lr-proper-p-areasp
$\text{proper-p-data-segmentp}(\mathit{data\text{-}seg}, p) \rightarrow \text{lr-proper-p-areasp}(\mathit{data\text{-}seg})$

THEOREM: proper-p-statep-lr->p-implies-lr-proper-p-areasp
$\text{proper-p-statep}(\text{lr->p}(l)) \rightarrow \text{lr-proper-p-areasp}(\text{p-data-segment}(l))$

EVENT: Disable proper-p-data-segmentp-implies-lr-proper-p-areasp.

THEOREM: lr-proper-free-listp-type-fetch-free-ptr
$(\text{lr-proper-free-listp}(\mathit{data\text{-}seg})$
$\wedge \quad \text{adpp}(\text{untag}(\text{lr-max-node}(\mathit{data\text{-}seg})), \mathit{data\text{-}seg})$
$\wedge \quad \text{lr-boundary-nodep}(\text{lr-max-node}(\mathit{data\text{-}seg}))$
$\wedge \quad \text{lr-proper-p-areasp}(\mathit{data\text{-}seg}))$
$\rightarrow \quad (\text{type}(\text{fetch}(\text{add-addr}(\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \mathit{data\text{-}seg}),$
$\quad\quad\quad\quad \text{identity}(\text{LR-REF-COUNT-OFFSET})),$
$\quad\quad \mathit{data\text{-}seg}))$
$\quad \neq \quad \mathtt{'nat})$

THEOREM: put-assoc-put-assoc-1
$\text{put-assoc}(\mathit{val1}, \mathit{name}, \text{put-assoc}(\mathit{val2}, \mathit{name}, \mathit{alist}))$
$= \quad \text{put-assoc}(\mathit{val1}, \mathit{name}, \mathit{alist})$

THEOREM: put-assoc-put-assoc-2
$\text{put-assoc}(\mathit{val1}, \mathit{name1}, \text{put-assoc}(\mathit{val2}, \mathit{name2}, \mathit{alist}))$
$= \quad \mathbf{if}\ \mathit{name1} = \mathit{name2}\ \mathbf{then}\ \text{put-assoc}(\mathit{val1}, \mathit{name1}, \mathit{alist})$
$\quad \mathbf{else}\ \text{put-assoc}(\mathit{val2}, \mathit{name2}, \text{put-assoc}(\mathit{val1}, \mathit{name1}, \mathit{alist}))\ \mathbf{endif}$

THEOREM: deposit-deposit
$((\text{offset}(\mathit{addr1}) \in \mathbf{N}) \wedge (\text{offset}(\mathit{addr2}) \in \mathbf{N}))$

$$
\begin{array}{l}
\rightarrow \quad (\text{deposit}\,(\mathit{value1},\ \mathit{addr1},\ \text{deposit}\,(\mathit{value2},\ \mathit{addr2},\ \mathit{data\text{-}seg}))\\
\qquad = \quad \textbf{if}\ (\text{area-name}\,(\mathit{addr1}) = \text{area-name}\,(\mathit{addr2}))\\
\qquad\qquad\quad \wedge \quad (\text{offset}\,(\mathit{addr1}) = \text{offset}\,(\mathit{addr2}))\\
\qquad\qquad \textbf{then}\ \text{deposit}\,(\mathit{value1},\ \mathit{addr1},\ \mathit{data\text{-}seg})\\
\qquad\qquad \textbf{else}\ \text{deposit}\,(\mathit{value2},\\
\qquad\qquad\qquad\qquad \mathit{addr2},\\
\qquad\qquad\qquad\qquad \text{deposit}\,(\mathit{value1},\ \mathit{addr1},\ \mathit{data\text{-}seg}))\ \textbf{endif})
\end{array}
$$

THEOREM: deposit-ref-count-move-outward

$$
\begin{array}{l}
(\text{offset}\,(\mathit{addr}) \in \mathbf{N})\\
\rightarrow \quad (\text{deposit}\,(\mathit{value1},\\
\qquad\qquad \mathit{addr},\\
\qquad\qquad \text{deposit}\,(\mathit{value2},\\
\qquad\qquad\qquad \text{add-addr}\,(\mathit{addr},\ \text{identity}\,(\textsc{lr-ref-count-offset})),\\
\qquad\qquad\qquad \mathit{data\text{-}seg}))\\
\qquad = \quad \text{deposit}\,(\mathit{value2},\\
\qquad\qquad\qquad \text{add-addr}\,(\mathit{addr},\ \textsc{lr-ref-count-offset}),\\
\qquad\qquad\qquad \text{deposit}\,(\mathit{value1},\ \mathit{addr},\ \mathit{data\text{-}seg})))
\end{array}
$$

DEFINITION:

$$
\begin{array}{l}
\text{ihint-2}\,(\mathit{flag},\ s,\ l,\ \mathit{table},\ c)\\
= \quad \textbf{if}\ \text{s-err-flag}\,(s) \neq \texttt{'run}\ \textbf{then t}\\
\quad\ \ \textbf{elseif}\ \mathit{flag} = \texttt{'list}\\
\quad\ \ \textbf{then if}\ \text{s-pos}\,(s) \simeq \textbf{nil}\ \textbf{then t}\\
\qquad\qquad \textbf{elseif}\ \text{listp}\,(\text{s-expr-list}\,(s))\\
\qquad\qquad \textbf{then}\ \text{ihint-2}\,(\mathbf{t},\ s,\ l,\ \mathit{table},\ c)\\
\qquad\qquad\qquad \wedge \quad \text{ihint-2}\,(\texttt{'list},\\
\qquad\qquad\qquad\qquad\quad \text{s-set-expr}\,(\text{s-eval}\,(\mathbf{t},\ s,\ c),\ s,\ \text{nx}\,(\text{s-pos}\,(s))),\\
\qquad\qquad\qquad\qquad\quad \text{lr-eval}\,(\mathbf{t},\ \text{s->lr1}\,(s,\ l,\ \mathit{table}),\ c),\\
\qquad\qquad\qquad\qquad\quad \mathit{table},\\
\qquad\qquad\qquad\qquad\quad c)\\
\qquad\qquad \textbf{else t endif}\\
\quad\ \ \textbf{elseif}\ c \simeq \texttt{0}\ \textbf{then t}\\
\quad\ \ \textbf{elseif}\ \text{litatom}\,(\text{s-expr}\,(s))\ \textbf{then t}\\
\quad\ \ \textbf{elseif}\ \text{s-expr}\,(s) \simeq \textbf{nil}\ \textbf{then t}\\
\quad\ \ \textbf{elseif}\ \text{car}\,(\text{s-expr}\,(s)) = \texttt{'if}\\
\quad\ \ \textbf{then let}\ \mathit{lrtest}\ \textbf{be}\ \text{lr-if-ok}\,(\text{lr-eval}\,(\mathbf{t},\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{s->lr1}\,(\text{s-set-pos}\,(s,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{dv}\,(\text{s-pos}\,(s),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \texttt{1})),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad l,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{table}),\\
\qquad\qquad\qquad\qquad\qquad\qquad c)),\\
\qquad\qquad\quad \mathit{stest}\ \textbf{be}\ \text{s-eval}\,(\mathbf{t},\ \text{s-set-pos}\,(s,\ \text{dv}\,(\text{s-pos}\,(s),\ \texttt{1})),\ c)
\end{array}
$$

```
    in
    if p-psw (lrtest) = 'run
    then if top (p-temp-stk (lrtest)) ≠ LR-F-ADDR
         then ihint-2 (t,
                       s-set-pos (s, dv (s-pos (s), 1)),
                       l,
                       table,
                       c)
              ∧  ihint-2 (t,
                          s-set-expr (stest,
                                      s,
                                      dv (s-pos (s), 2)),
                          lr-pop-tstk (lrtest),
                          table,
                          c)
         else ihint-2 (t,
                       s-set-pos (s, dv (s-pos (s), 1)),
                       l,
                       table,
                       c)
              ∧  ihint-2 (t,
                          s-set-expr (stest,
                                      s,
                                      dv (s-pos (s), 3)),
                          lr-pop-tstk (lrtest),
                          table,
                          c) endif
    else ihint-2 (t,
                  s-set-pos (s, dv (s-pos (s), 1)),
                  l,
                  table,
                  c) endif endlet
elseif car (s-expr (s)) = S-TEMP-EVAL
then ihint-2 (t, s-set-pos (s, dv (s-pos (s), 1)), l, table, c)
elseif car (s-expr (s)) = S-TEMP-TEST
then if p-max-temp-stk-size (l) ≮ (2 + length (p-temp-stk (l)))
     then if lr-eval-temp-setp (s->lr1 (s, l, table))  then t
          else ihint-2 (t, s-set-pos (s, dv (s-pos (s), 1)), l, table, c) endif
     else t endif
elseif car (s-expr (s)) = S-TEMP-FETCH  then t
elseif car (s-expr (s)) = 'quote  then t
elseif s-err-flag (s-eval ('list, s-set-pos (s, dv (s-pos (s), 1)), c))
     ≠  'run
then ihint-2 ('list, s-set-pos (s, dv (s-pos (s), 1)), l, table, c)
```

```
    elseif subrp (car (s-expr (s)))
    then ihint-2 ('list, s-set-pos (s, dv (s-pos (s), 1)), l, table, c)
    elseif litatom (car (s-expr (s)))
    then let s-arg-s  be  s-eval ('list, s-set-pos (s, dv (s-pos (s), 1)), c),
             lr-arg-s  be  lr-eval ('list,
                                    s->lr1 (s-set-pos (s, dv (s-pos (s), 1)),
                                            l,
                                            table),
                                    c)
         in
         ihint-2 ('list, s-set-pos (s, dv (s-pos (s), 1)), l, table, c)
         ∧   ihint-2 (t,
                      s-fun-call-state (s-arg-s, car (s-expr (s))),
                      lr-funcall (s->lr1 (s, l, table), lr-arg-s),
                      table,
                      c − 1) endlet
    else t endif
```

DEFINITION:

```
induct-hint-4 (x, temp-stk)
=   if listp (x)  then induct-hint-4 (cdr (x), cdr (temp-stk))
    else t endif
```

THEOREM: lr-check-result1-append

```
lr-check-result1 (append (x, y), temp-stk, data-seg)
=   (lr-check-result1 (x, temp-stk, data-seg)
     ∧   lr-check-result1 (y, restn (length (x), temp-stk), data-seg))
```

THEOREM: lr-proper-heapp-opener-4

```
lr-proper-heapp (data-seg)
→   (adpp (untag (lr-max-node (data-seg)), data-seg)
     ∧   lr-boundary-nodep (lr-max-node (data-seg)))
```

THEOREM: length-strip-cars

```
length (strip-cars (temp-vars)) = length (temp-vars)
```

THEOREM: definedp-lr-compile-programs

```
definedp (name, lr-compile-programs (progs, const-table))
=   definedp (name, progs)
```

THEOREM: lr-valp-deposit-fetch-free-pointer-offset-helper-1

```
((type (fetch (add-addr (addr, identity (LR-REF-COUNT-OFFSET)), data-seg))
  =   'nat)
 ∧   lr-good-pointerp (addr, data-seg)
 ∧   lr-nodep (free-addr, data-seg)
```

$$
\begin{array}{ll}
\wedge & (\text{offset}\,(\mathit{addr}) = \text{offset}\,(\mathit{free\text{-}addr}))) \\
\rightarrow & (\text{type}\,(\text{fetch}\,(\text{add-addr}\,(\mathit{free\text{-}addr},\ \text{identity}\,(\textsc{lr-ref-count-offset})), \\
& \qquad\qquad\quad \mathit{data\text{-}seg})) \\
& \ = \quad \texttt{'nat})
\end{array}
$$

THEOREM: lr-boundary-nodep-equal-plus-fact-zero

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{type}\,(\mathit{addr1}) = \texttt{'addr})} \\
\wedge & (\text{cddr}\,(\mathit{addr1}) = \textbf{nil}) \\
\wedge & \text{listp}\,(\mathit{addr1}) \\
\wedge & \text{listp}\,(\text{untag}\,(\mathit{addr1})) \\
\wedge & (\text{offset}\,(\mathit{addr1}) \in \mathbf{N}) \\
\wedge & \text{lr-boundary-nodep}\,(\mathit{addr1}) \\
\wedge & (\text{type}\,(\mathit{addr2}) = \texttt{'addr}) \\
\wedge & (\text{cddr}\,(\mathit{addr2}) = \textbf{nil}) \\
\wedge & \text{listp}\,(\mathit{addr2}) \\
\wedge & \text{listp}\,(\text{untag}\,(\mathit{addr2})) \\
\wedge & (\text{offset}\,(\mathit{addr2}) \in \mathbf{N}) \\
\wedge & \text{lr-boundary-nodep}\,(\mathit{addr2}) \\
\wedge & (\text{area-name}\,(\mathit{addr2}) = \text{area-name}\,(\mathit{addr1})) \\
\wedge & (m < \textsc{lr-node-size})) \\
\rightarrow & ((\text{offset}\,(\mathit{addr1}) = (m + \text{offset}\,(\mathit{addr2}))) \\
& \ = \quad ((m \simeq \mathsf{0}) \wedge (\mathit{addr1} = \mathit{addr2})))
\end{array}
$$

THEOREM: lr-boundary-nodep-equal-plus-fact

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{type}\,(\mathit{addr1}) = \texttt{'addr})} \\
\wedge & (\text{cddr}\,(\mathit{addr1}) = \textbf{nil}) \\
\wedge & \text{listp}\,(\mathit{addr1}) \\
\wedge & \text{listp}\,(\text{untag}\,(\mathit{addr1})) \\
\wedge & (\text{offset}\,(\mathit{addr1}) \in \mathbf{N}) \\
\wedge & \text{lr-boundary-nodep}\,(\mathit{addr1}) \\
\wedge & (\text{type}\,(\mathit{addr2}) = \texttt{'addr}) \\
\wedge & (\text{cddr}\,(\mathit{addr2}) = \textbf{nil}) \\
\wedge & \text{listp}\,(\mathit{addr2}) \\
\wedge & \text{listp}\,(\text{untag}\,(\mathit{addr2})) \\
\wedge & (\text{offset}\,(\mathit{addr2}) \in \mathbf{N}) \\
\wedge & \text{lr-boundary-nodep}\,(\mathit{addr2}) \\
\wedge & (n < \textsc{lr-node-size}) \\
\wedge & (m < \textsc{lr-node-size}) \\
\wedge & (\text{area-name}\,(\mathit{addr1}) = \text{area-name}\,(\mathit{addr2}))) \\
\rightarrow & (((n + \text{offset}\,(\mathit{addr1})) = (m + \text{offset}\,(\mathit{addr2}))) \\
& \ = \quad ((\text{fix}\,(n) = \text{fix}\,(m)) \wedge (\mathit{addr1} = \mathit{addr2})))
\end{array}
$$

THEOREM: lr-valp-deposit-fetch-free-pointer-offset

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{type}\,(\text{fetch}\,(\text{add-addr}\,(\mathit{free\text{-}addr},\ \textsc{lr-ref-count-offset}),\ \mathit{data\text{-}seg})) \neq \texttt{'nat})} \\
\wedge & \text{lr-nodep}\,(\mathit{free\text{-}addr},\ \mathit{data\text{-}seg})
\end{array}
$$

$\wedge$ $(n <$ LR-NODE-SIZE$)$
$\wedge$ lr-valp (*value*, *addr*, *data-seg*))
$\rightarrow$ lr-valp (*value*, *addr*, deposit (*anything*, add-addr (*free-addr*, *n*), *data-seg*))

EVENT: Disable lr-valp-deposit-fetch-free-pointer-offset.

EVENT: Disable lr-valp-deposit-fetch-free-pointer-offset-helper-1.

THEOREM: lr-valp-deposit-fetch-free-pointer
((type (fetch (add-addr (*free-addr*, LR-REF-COUNT-OFFSET), *data-seg*)) $\neq$ `'nat`)
$\wedge$ lr-nodep (*free-addr*, *data-seg*)
$\wedge$ lr-valp (*value*, *addr*, *data-seg*))
$\rightarrow$ lr-valp (*value*, *addr*, deposit (*anything*, *free-addr*, *data-seg*))

EVENT: Disable lr-valp-deposit-fetch-free-pointer.

THEOREM: not-equal-x-add1-add1-x
$(x = (1 + (1 + x))) = \mathbf{f}$

THEOREM: not-equal-x-add1-x
$(x = (1 + x)) = \mathbf{f}$

THEOREM: p-run-subr-preserves-lr-valp
(good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\wedge$ lr-programs-properp (*l*, *table*)
$\wedge$ lr-programs-properp (*new-l*, *table*)
$\wedge$ listp (lr-expr (*l*))
$\wedge$ proper-p-statep (lr->p (*new-l*))
$\wedge$ lr-proper-free-listp (p-data-segment (*new-l*))
$\wedge$ adpp (untag (lr-max-node (p-data-segment (*new-l*))), p-data-segment (*new-l*))
$\wedge$ lr-boundary-nodep (lr-max-node (p-data-segment (*new-l*)))
$\wedge$ (p-psw (*new-l*) = `'run`)
$\wedge$ (p-psw (p-run-subr (car (lr-expr (*l*)),
p-set-pc (lr->p (*new-l*), lr-return-pc (*l*))))
= `'run`)
$\wedge$ lr-valp (*value*, *addr*, p-data-segment (*new-l*))
$\wedge$ (p-prog-segment (*l*) = p-prog-segment (*new-l*)))
$\rightarrow$ lr-valp (*value*,
addr,
p-data-segment (p-run-subr (car (lr-expr (*l*)),
p-set-pc (lr->p (*new-l*),
lr-return-pc (*l*)))))

THEOREM: numberp-offset-sub-addr
$\text{offset}(\text{sub-addr}(\textit{addr},\ n)) \in \mathbf{N}$

THEOREM: lr-free-list-nodes-deposit-non-ref-count
$(\text{lr-nodep}(\textit{addr},\ \textit{data-seg})$
$\land\ \ \text{lr-nodep}(\textit{max-addr},\ \textit{data-seg})$
$\land\ \ (\textit{offset} \neq \text{LR-REF-COUNT-OFFSET})$
$\land\ \ (\textit{offset} \in \mathbf{N})$
$\land\ \ (\textit{offset} < \text{LR-NODE-SIZE}))$
$\rightarrow\ \ (\text{lr-free-list-nodes}(\textit{max-addr},$
$\qquad\qquad \text{deposit}(\textit{anything},\ \text{add-addr}(\textit{addr},\ \textit{offset}),\ \textit{data-seg}))$
$\qquad = \ \text{lr-free-list-nodes}(\textit{max-addr},\ \textit{data-seg}))$

THEOREM: lr-nodep-member-lr-node-listp-adpp-untag-listp
$(\text{lr-node-listp}(\textit{list},\ \textit{data-seg}) \land (\textit{node} \in \textit{list})) \rightarrow \text{listp}(\text{untag}(\textit{node}))$

EVENT: Disable lr-nodep-member-lr-node-listp-adpp-untag-listp.

THEOREM: lr-nodep-member-lr-node-listp-adpp-untag-numberp-offset
$(\text{lr-node-listp}(\textit{list},\ \textit{data-seg}) \land (\textit{node} \in \textit{list})) \rightarrow (\text{offset}(\textit{node}) \in \mathbf{N})$

THEOREM: lr-nodep-member-lr-node-listp-lr-boundaryp-offsetp
$(\text{lr-node-listp}(\textit{list},\ \textit{data-seg}) \land (\textit{node} \in \textit{list}))$
$\rightarrow\ \ \text{lr-boundary-offsetp}(\text{offset}(\textit{node}))$

THEOREM: lr-check-free-nodes-deposit-non-ref-count
$(\text{lr-nodep}(\textit{addr2},\ \textit{data-seg})$
$\land\ \ \text{lr-nodep}(\textit{max-addr},\ \textit{data-seg})$
$\land\ \ (\textit{offset} \neq \text{LR-REF-COUNT-OFFSET})$
$\land\ \ (\textit{offset} \in \mathbf{N})$
$\land\ \ (\textit{offset} < \text{LR-NODE-SIZE})$
$\land\ \ \text{lr-node-listp}(\textit{node-list},\ \textit{data-seg}))$
$\rightarrow\ \ (\text{lr-check-free-nodes}(\textit{addr1},$
$\qquad\qquad \textit{node-list},$
$\qquad\qquad \text{deposit}(\textit{anything},\ \text{add-addr}(\textit{addr2},\ \textit{offset}),\ \textit{data-seg}),$
$\qquad\qquad \textit{max-addr})$
$\qquad = \ \text{lr-check-free-nodes}(\textit{addr1},\ \textit{node-list},\ \textit{data-seg},\ \textit{max-addr}))$

EVENT: Disable lr-nodep-member-lr-node-listp-adpp-untag-numberp-offset.

THEOREM: adpp-deposit-other-area
$(\text{adp-name}(\textit{adp}) \neq \text{area-name}(\textit{addr}))$
$\rightarrow\ \ (\text{adpp}(\textit{adp},\ \text{deposit}(\textit{anything},\ \textit{addr},\ \textit{data-seg})) = \text{adpp}(\textit{adp},\ \textit{data-seg}))$

EVENT: Disable adpp-deposit-other-area.

THEOREM: length-deposit

length (cdr (assoc (*name*, deposit (*anything*, *addr*, *data-seg*))))
= **if** definedp (area-name (*addr*), *data-seg*)
  then if area-name (*addr*) = *name*
    then if offset (*addr*) < length (cdr (assoc (*name*, *data-seg*)))
      then length (cdr (assoc (*name*, *data-seg*)))
      else 1 + offset (*addr*) **endif**
    else length (cdr (assoc (*name*, *data-seg*))) **endif**
  else length (cdr (assoc (*name*, *data-seg*))) **endif**

THEOREM: same-signature-deposit

(adpp (untag (*addr*), *segment2*) ∧ lr-proper-p-areasp (*segment2*))
→ (same-signature (*segment1*, deposit (*anything*, *addr*, *segment2*))
  = same-signature (*segment1*, *segment2*))

THEOREM: lr-max-node-same-signature

same-signature (*data-seg1*, *data-seg2*)
→ (lr-max-node (*data-seg2*) = lr-max-node (*data-seg1*))

EVENT: Disable lr-max-node-same-signature.

THEOREM: lr-max-node-deposit

(adpp (untag (*addr*), *data-seg*) ∧ lr-proper-p-areasp (*data-seg*))
→ (lr-max-node (deposit (*anything*, *addr*, *data-seg*))
  = lr-max-node (*data-seg*))

THEOREM: not-adpp-untag-node-not-definedp-lr-heap-name

(¬ definedp (area-name (*addr*), *data-seg*))
→ (¬ adpp (untag (*addr*), *data-seg*))

THEOREM: sub-addr-area-name-offset-same

(listp (untag (*addr1*))
∧ (offset (*addr1*) ∈ **N**)
∧ (cddr (*addr1*) = **nil**)
∧ (type (*addr1*) = type (*addr2*)))
→ ((*addr1* = sub-addr (*addr2*, *n*))
  = ((offset (*addr1*) = (offset (*addr2*) − *n*))
    ∧ (area-name (*addr1*) = area-name (*addr2*))))

THEOREM: lr-free-list-nodes-member-greater-offset

(offset (*addr*) $\not<$ offset (*max-addr*))
→ (*addr* ∉ lr-free-list-nodes (*max-addr*, *data-seg*))

THEOREM: lr-free-list-nodes-deposit-lr-ref-count-offset

((type (*addr*) = `'addr`)

$\wedge$ (cddr (*addr*) = **nil**)
$\wedge$ adpp (untag (*addr*), *data-seg*)
$\wedge$ lr-boundary-nodep (*addr*)
$\wedge$ (area-name (*addr*) = `'heap`)
$\wedge$ (type (*max-addr*) = `'addr`)
$\wedge$ adpp (untag (*max-addr*), *data-seg*)
$\wedge$ lr-boundary-nodep (*max-addr*)
$\wedge$ (area-name (*max-addr*) = `'heap`)
$\wedge$ (type (*ref-count*) = `'nat`)
$\wedge$ (untag (*ref-count*) $\in$ **N**))
$\rightarrow$ (lr-free-list-nodes (*max-addr*,
deposit (*ref-count*,
add-addr (*addr*,
identity (LR-REF-COUNT-OFFSET)),
data-seg))
= delete (*addr*, lr-free-list-nodes (*max-addr*, *data-seg*)))

EVENT: Disable lr-free-list-nodes-member-greater-offset.

EVENT: Disable not-adpp-untag-node-not-definedp-lr-heap-name.

DEFINITION:
no-duplicatesp (*list*)
= **if** listp (*list*)
then if car (*list*) $\in$ cdr (*list*) **then f**
else no-duplicatesp (cdr (*list*)) **endif**
else t endif

THEOREM: not-member-occurences-0
($x \notin z$) $\rightarrow$ (occurrences (x, z) = `0`)

EVENT: Disable not-member-occurences-0.

THEOREM: no-duplicatesp-occurences-1
(no-duplicatesp (*list*) $\wedge$ ($e \in$ *list*)) $\rightarrow$ (occurrences (e, *list*) = `1`)

THEOREM: no-duplicatesp-lr-free-list-nodes
no-duplicatesp (lr-free-list-nodes (*addr*, *data-seg*))

THEOREM: member-area-name-offset-same
((*addr1* $\in$ *node-list*)
$\wedge$ (offset (*addr1*) $\in$ **N**)
$\wedge$ (cddr (*addr1*) = **nil**)

$\wedge$ listp (untag (*addr1*))
$\wedge$ listp (untag (*addr2*))
$\wedge$ (offset (*addr2*) $\in \mathbf{N}$)
$\wedge$ (cddr (*addr2*) = **nil**)
$\wedge$ (type (*addr1*) = type (*addr2*))
$\wedge$ (area-name (*addr1*) = area-name (*addr2*))
$\wedge$ (offset (*addr2*) = offset (*addr1*)))
$\rightarrow$ (*addr2* $\in$ *node-list*)

THEOREM: lr-check-free-nodes-delete-deposit
(lr-check-free-nodes (*addr2*, *node-list*, *data-seg*, *max-addr*)
$\wedge$ (*addr1* $\notin$ *node-list*)
$\wedge$ lr-nodep (*addr1*, *data-seg*)
$\wedge$ lr-node-listp (*node-list*, *data-seg*)
$\wedge$ (type (*ref-count*) = `'nat`)
$\wedge$ (untag (*ref-count*) $\in \mathbf{N}$))
$\rightarrow$ lr-check-free-nodes (*addr2*,
node-list,
deposit (*ref-count*,
add-addr (*addr1*,
identity (LR-REF-COUNT-OFFSET)),
data-seg),
max-addr)

EVENT: Disable lr-check-free-nodes-delete-deposit.

EVENT: Disable member-area-name-offset-same.

THEOREM: lr-check-free-nodes-deposit-free-ptr
(adpp (identity (untag (LR-FP-ADDR)), *data-seg*)
$\wedge$ lr-node-listp (*node-list*, *data-seg*))
$\rightarrow$ (lr-check-free-nodes (*addr*,
node-list,
deposit (*anything*, identity (LR-FP-ADDR), *data-seg*),
max-addr)
= lr-check-free-nodes (*addr*, *node-list*, *data-seg*, *max-addr*))

THEOREM: lr-free-list-nodes-deposit-free-ptr
(lr-nodep (*max-addr*, *data-seg*) $\wedge$ adpp (identity (untag (LR-FP-ADDR)), *data-seg*))
$\rightarrow$ (lr-free-list-nodes (*max-addr*,
deposit (*anything*, identity (LR-FP-ADDR), *data-seg*))
= lr-free-list-nodes (*max-addr*, *data-seg*))

THEOREM: deposit-ref-count-move-inward-2

$$
\begin{array}{l}
(\text{lr-nodep}\,(\mathit{addr},\ \mathit{data\text{-}seg}) \\
\ \wedge \quad (\mathit{offset} \neq \text{LR-REF-COUNT-OFFSET}) \\
\ \wedge \quad (\mathit{offset} \not\simeq \mathsf{0}) \\
\ \wedge \quad (\mathit{offset} < \text{LR-NODE-SIZE})) \\
\rightarrow \quad (\text{deposit}\,(\mathit{any1}, \\
\qquad\qquad\qquad \text{add-addr}\,(\mathit{addr},\ \text{identity}\,(\text{LR-REF-COUNT-OFFSET})), \\
\qquad\qquad\qquad \text{deposit}\,(\mathit{any2},\ \text{add-addr}\,(\mathit{addr},\ \mathit{offset}),\ \mathit{data\text{-}seg})) \\
\qquad = \quad \text{deposit}\,(\mathit{any2}, \\
\qquad\qquad\qquad\quad \text{add-addr}\,(\mathit{addr},\ \mathit{offset}), \\
\qquad\qquad\qquad\quad \text{deposit}\,(\mathit{any1}, \\
\qquad\qquad\qquad\qquad\quad \text{add-addr}\,(\mathit{addr},\ \text{identity}\,(\text{LR-REF-COUNT-OFFSET})), \\
\qquad\qquad\qquad\qquad\quad \mathit{data\text{-}seg})))
\end{array}
$$

EVENT: Disable deposit-ref-count-move-inward-2.

THEOREM: lr-free-list-nodes-deposit-lr-nodep

$$
\begin{array}{l}
(\text{lr-nodep}\,(\mathit{addr},\ \mathit{data\text{-}seg}) \wedge \text{lr-nodep}\,(\mathit{max\text{-}addr},\ \mathit{data\text{-}seg})) \\
\rightarrow \quad (\text{lr-free-list-nodes}\,(\mathit{max\text{-}addr},\ \text{deposit}\,(\mathit{anything},\ \mathit{addr},\ \mathit{data\text{-}seg})) \\
\qquad = \quad \text{lr-free-list-nodes}\,(\mathit{max\text{-}addr},\ \mathit{data\text{-}seg}))
\end{array}
$$

THEOREM: lr-check-free-nodes-deposit-lr-nodep

$$
\begin{array}{l}
(\text{lr-nodep}\,(\mathit{addr2},\ \mathit{data\text{-}seg}) \\
\ \wedge \quad \text{lr-nodep}\,(\mathit{max\text{-}addr},\ \mathit{data\text{-}seg}) \\
\ \wedge \quad \text{lr-node-listp}\,(\mathit{node\text{-}list},\ \mathit{data\text{-}seg})) \\
\rightarrow \quad (\text{lr-check-free-nodes}\,(\mathit{addr1}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{node\text{-}list}, \\
\qquad\qquad\qquad\qquad\qquad \text{deposit}\,(\mathit{anything},\ \mathit{addr2},\ \mathit{data\text{-}seg}), \\
\qquad\qquad\qquad\qquad\qquad \mathit{max\text{-}addr}) \\
\qquad = \quad \text{lr-check-free-nodes}\,(\mathit{addr1},\ \mathit{node\text{-}list},\ \mathit{data\text{-}seg},\ \mathit{max\text{-}addr}))
\end{array}
$$

THEOREM: same-signature-cons

$$
\begin{array}{l}
\text{same-signature}\,(\mathit{data\text{-}seg1},\ \text{cons}\,(x,\ \mathit{data\text{-}seg2})) \\
= \quad \textbf{if}\ \text{listp}\,(\mathit{data\text{-}seg1}) \\
\qquad \textbf{then}\ \ (\text{signature}\,(\text{car}\,(\mathit{data\text{-}seg1})) = \text{signature}\,(x)) \\
\qquad\qquad\quad \wedge \quad \text{same-signature}\,(\text{cdr}\,(\mathit{data\text{-}seg1}),\ \mathit{data\text{-}seg2}) \\
\qquad \textbf{else f endif}
\end{array}
$$

THEOREM: same-signature-nil

$$
\begin{array}{l}
(\mathit{data\text{-}seg1} \simeq \textbf{nil}) \\
\rightarrow \quad (\text{same-signature}\,(\mathit{data\text{-}seg1},\ \mathit{data\text{-}seg2}) = (\mathit{data\text{-}seg2} \simeq \textbf{nil}))
\end{array}
$$

THEOREM: listp-put-assoc

$$
\text{listp}\,(\text{put-assoc}\,(\mathit{val},\ \mathit{name},\ \mathit{alist})) = \text{listp}\,(\mathit{alist})
$$

THEOREM: not-same-signature-deposit-too-large-addr
$(\text{definedp}(\text{area-name}(addr),\ data\text{-}seg2)$
$\land\ \ \text{lr-proper-p-areasp}(data\text{-}seg2)$
$\land\ \ (\text{offset}(addr) \not< \text{length}(\text{value}(\text{area-name}(addr),\ data\text{-}seg1))))$
$\rightarrow\ \ (\neg\ \text{same-signature}(data\text{-}seg1,\ \text{deposit}(any,\ addr,\ data\text{-}seg2)))$

EVENT: Disable same-signature-cons.

EVENT: Disable same-signature-nil.

THEOREM: adpp-deposit-a-list
$\text{adpp}(adp,\ data\text{-}seg) \rightarrow \text{adpp}(adp,\ \text{deposit-a-list}(list,\ addr2,\ data\text{-}seg))$

THEOREM: lr-proper-p-areasp-deposit-a-list
$\text{lr-proper-p-areasp}(data\text{-}seg)$
$\rightarrow\ \ \text{lr-proper-p-areasp}(\text{deposit-a-list}(list,\ addr,\ data\text{-}seg))$

THEOREM: definedp-deposit-a-list
$\text{definedp}(tag,\ \text{deposit-a-list}(list,\ addr,\ data\text{-}seg)) = \text{definedp}(tag,\ data\text{-}seg)$

THEOREM: sub1-plus-not-zerop-fact-1
$(x \not\simeq 0) \rightarrow ((((y + x) - 1) < y) = \mathbf{f})$

THEOREM: not-adpp-untag-add-addr-adpp-untag
$\text{adpp}(\text{untag}(addr),\ data\text{-}seg)$
$\rightarrow\ \ (\text{adpp}(\text{untag}(\text{add-addr}(addr,\ n)),\ data\text{-}seg)$
$\quad =\ \ ((\text{offset}(addr) + n)$
$\quad\quad <\ \ \text{length}(\text{cdr}(\text{assoc}(\text{area-name}(addr),\ data\text{-}seg)))))$

THEOREM: not-same-signature-deposit-a-list-too-large-addr
$(\text{definedp}(\text{area-name}(addr),\ data\text{-}seg2)$
$\land\ \ \text{lr-proper-p-areasp}(data\text{-}seg2)$
$\land\ \ (\text{offset}(addr) \not< \text{length}(\text{value}(\text{area-name}(addr),\ data\text{-}seg1))))$
$\rightarrow\ \ (\text{same-signature}(data\text{-}seg1,\ \text{deposit-a-list}(list,\ addr,\ data\text{-}seg2))$
$\quad =\ \ \textbf{if}\ \text{listp}(list)\ \textbf{then f}$
$\quad\quad \textbf{else}\ \text{same-signature}(data\text{-}seg1,\ data\text{-}seg2)\ \textbf{endif})$

THEOREM: same-signature-deposit-a-list
$(\text{adpp}(\text{untag}(addr),\ data\text{-}seg2)$
$\land\ \ \text{lr-proper-p-areasp}(data\text{-}seg2)$
$\land\ \ \text{same-signature}(data\text{-}seg1,\ data\text{-}seg2))$
$\rightarrow\ \ (\text{same-signature}(data\text{-}seg1,\ \text{deposit-a-list}(list,\ addr,\ data\text{-}seg2))$
$\quad =\ \ ((\text{offset}(addr) + (\text{length}(list) - 1))$
$\quad\quad <\ \ \text{length}(\text{cdr}(\text{assoc}(\text{area-name}(addr),\ data\text{-}seg1)))))$

EVENT: Disable not-same-signature-deposit-too-large-addr.

EVENT: Disable not-same-signature-deposit-a-list-too-large-addr.

THEOREM: deposit-good-node-preserves-lr-proper-free-listp
(lr-proper-free-listp (*data-seg*)
 ∧ lr-proper-p-areasp (*data-seg*)
 ∧ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
 ∧ lr-boundary-nodep (lr-max-node (*data-seg*))
 ∧ (offset (fetch (LR-FP-ADDR, *data-seg*))
  < (((length (cdr (assoc (LR-HEAP-NAME, *data-seg*))) − 1) − 1) − 1))
 ∧ (type (*tag*) = 'nat)
 ∧ (type (*ref-count*) = 'nat)
 ∧ (untag (*ref-count*) ∈ **N**))
→ lr-proper-free-listp (deposit (fetch (add-addr (fetch (identity (LR-FP-ADDR),
                 data-seg),
               identity (LR-REF-COUNT-OFFSET)),
           data-seg),
         identity (LR-FP-ADDR),
         deposit-a-list (list (*tag*, *ref-count*, *x*, *y*),
               fetch (identity (LR-FP-ADDR),
                 data-seg),
               data-seg)))

THEOREM: p-run-subr-preserves-lr-proper-free-listp
(good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
 ∧ lr-programs-properp (*l*, *table*)
 ∧ lr-programs-properp (*new-l*, *table*)
 ∧ listp (lr-expr (*l*))
 ∧ proper-p-statep (lr->p (*new-l*))
 ∧ lr-proper-free-listp (p-data-segment (*new-l*))
 ∧ adpp (untag (lr-max-node (p-data-segment (*new-l*))), p-data-segment (*new-l*))
 ∧ lr-boundary-nodep (lr-max-node (p-data-segment (*new-l*)))
 ∧ (p-psw (*new-l*) = 'run)
 ∧ (p-psw (p-run-subr (car (lr-expr (*l*)),
         p-set-pc (lr->p (*new-l*), lr-return-pc (*l*))))
  = 'run)
 ∧ (p-prog-segment (*l*) = p-prog-segment (*new-l*))
 ∧ (area-name (p-pc (*l*)) = area-name (p-pc (*new-l*))))
→ lr-proper-free-listp (p-data-segment (p-run-subr (car (lr-expr (*l*)),
              p-set-pc (lr->p (*new-l*),
                  lr-return-pc (*l*)))))

EVENT: Disable same-signature-deposit.

THEOREM: lr-apply-subr-preserves-lr-proper-free-listp
let *new-l* **be** lr-eval ('`list`, lr-set-pos (l, *pos*), c)
in
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ listp (lr-expr (l))
 ∧ subrp (car (lr-expr (l)))
 ∧ (car (lr-expr (l)) ≠ '`if`)
 ∧ proper-p-statep (lr->p (*new-l*))
 ∧ lr-proper-free-listp (p-data-segment (*new-l*))
 ∧ adpp (untag (lr-max-node (p-data-segment (*new-l*))),
 p-data-segment (*new-l*))
 ∧ lr-boundary-nodep (lr-max-node (p-data-segment (*new-l*)))
 ∧ (p-psw (*new-l*) = '`run`)
 ∧ (p-psw (lr-apply-subr (l, *new-l*)) = '`run`))
→ lr-proper-free-listp (p-data-segment (lr-apply-subr (l, *new-l*))) **endlet**

THEOREM: lr-eval-preserves-proper-p-statep-lr->p-rewrite
(proper-p-statep (lr->p (l))
 ∧ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ (p-psw (lr-eval (*flag*, l, c)) = '`run`))
→ proper-p-statep (lr->p (lr-eval (*flag*, l, c)))

THEOREM: lr-eval-preserves-cdr-p-ctrl-stk
(proper-p-statep (lr->p (l))
 ∧ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ (p-psw (lr-eval (*flag*, l, c)) = '`run`))
→ (cdr (p-ctrl-stk (lr-eval (*flag*, l, c))) = cdr (p-ctrl-stk (l)))

THEOREM: lr-eval-preserves-strip-cars-bindings-car-p-ctrl-stk
(proper-p-statep (lr->p (l))
 ∧ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ (p-psw (lr-eval (*flag*, l, c)) = '`run`))
→ (strip-cars (bindings (car (p-ctrl-stk (lr-eval (*flag*, l, c)))))
 = strip-cars (bindings (car (p-ctrl-stk (l)))))

THEOREM: lr-eval-preserves-lr-max-node
(proper-p-statep (lr->p (l))
 ∧ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ (p-psw (lr-eval (*flag*, l, c)) = '`run`))
→ (lr-max-node (p-data-segment (lr-eval (*flag*, l, c)))
 = lr-max-node (p-data-segment (l)))

THEOREM: lr-eval-preserves-adpp
$$\begin{array}{ll} & (\text{proper-p-statep}\,(\text{lr->p}\,(l)) \\ \land & \text{good-posp}\,(\mathit{flag},\ \text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\ \land & \text{lr-programs-properp}\,(l,\ \mathit{table}) \\ \land & (\text{p-psw}\,(\text{lr-eval}\,(\mathit{flag},\ l,\ c)) = \texttt{'run})) \\ \to & (\text{adpp}\,(\mathit{adp},\ \text{p-data-segment}\,(\text{lr-eval}\,(\mathit{flag},\ l,\ c))) \\ & \quad = \ \text{adpp}\,(\mathit{adp},\ \text{p-data-segment}\,(l))) \end{array}$$

THEOREM: lr-eval-preserves-length-assoc-data-segment
$$\begin{array}{ll} & (\text{proper-p-statep}\,(\text{lr->p}\,(l)) \\ \land & \text{good-posp}\,(\mathit{flag},\ \text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\ \land & \text{lr-programs-properp}\,(l,\ \mathit{table}) \\ \land & (\text{p-psw}\,(\text{lr-eval}\,(\mathit{flag},\ l,\ c)) = \texttt{'run})) \\ \to & (\text{length}\,(\text{cdr}\,(\text{assoc}\,(\mathit{name},\ \text{p-data-segment}\,(\text{lr-eval}\,(\mathit{flag},\ l,\ c))))) \\ & \quad = \ \text{length}\,(\text{cdr}\,(\text{assoc}\,(\mathit{name},\ \text{p-data-segment}\,(l))))) \end{array}$$

THEOREM: lr-eval-preserves-proper-p-statep-lr->p-lr-set-pos
$$\begin{array}{ll} & (\text{proper-p-statep}\,(\text{lr->p}\,(l)) \\ \land & \text{good-posp}\,(\mathit{flag},\ \mathit{pos},\ \text{program-body}\,(\text{p-current-program}\,(l))) \\ \land & \text{lr-programs-properp}\,(l,\ \mathit{table}) \\ \land & (\text{p-psw}\,(\text{lr-eval}\,(\mathit{flag},\ \text{lr-set-pos}\,(l,\ \mathit{pos}),\ c)) = \texttt{'run})) \\ \to & \text{proper-p-statep}\,(\text{lr->p}\,(\text{lr-eval}\,(\mathit{flag},\ \text{lr-set-pos}\,(l,\ \mathit{pos}),\ c))) \end{array}$$

THEOREM: lr-eval-preserves-strip-cars-bindings-car-p-ctrl-stk-lr-set-pos
$$\begin{array}{ll} & (\text{proper-p-statep}\,(\text{lr->p}\,(l)) \\ \land & \text{good-posp1}\,(\mathit{pos},\ \text{program-body}\,(\text{p-current-program}\,(l))) \\ \land & \text{lr-programs-properp}\,(l,\ \mathit{table}) \\ \land & (\text{p-psw}\,(\text{lr-eval}\,(\mathbf{t},\ \text{lr-set-pos}\,(l,\ \mathit{pos}),\ c)) = \texttt{'run})) \\ \to & (\text{strip-cars}\,(\text{bindings}\,(\text{car}\,(\text{p-ctrl-stk}\,(\text{lr-eval}\,(\mathbf{t},\ \text{lr-set-pos}\,(l,\ \mathit{pos}),\ c))))) \\ & \quad = \ \text{strip-cars}\,(\text{bindings}\,(\text{car}\,(\text{p-ctrl-stk}\,(l))))) \end{array}$$

THEOREM: lr-eval-preserves-cdr-p-ctrl-stk-lr-set-pos
$$\begin{array}{ll} & (\text{proper-p-statep}\,(\text{lr->p}\,(l)) \\ \land & \text{good-posp1}\,(\mathit{pos},\ \text{program-body}\,(\text{p-current-program}\,(l))) \\ \land & \text{lr-programs-properp}\,(l,\ \mathit{table}) \\ \land & (\text{p-psw}\,(\text{lr-eval}\,(\mathbf{t},\ \text{lr-set-pos}\,(l,\ \mathit{pos}),\ c)) = \texttt{'run})) \\ \to & (\text{cdr}\,(\text{p-ctrl-stk}\,(\text{lr-eval}\,(\mathbf{t},\ \text{lr-set-pos}\,(l,\ \mathit{pos}),\ c))) \\ & \quad = \ \text{cdr}\,(\text{p-ctrl-stk}\,(l))) \end{array}$$

THEOREM: lr-eval-preserves-adpp-lr-set-pos
$$\begin{array}{ll} & (\text{proper-p-statep}\,(\text{lr->p}\,(l)) \\ \land & \text{good-posp}\,(\mathit{flag},\ \mathit{pos},\ \text{program-body}\,(\text{p-current-program}\,(l))) \\ \land & \text{lr-programs-properp}\,(l,\ \mathit{table}) \\ \land & (\text{p-psw}\,(\text{lr-eval}\,(\mathit{flag},\ \text{lr-set-pos}\,(l,\ \mathit{pos}),\ c)) = \texttt{'run})) \\ \to & (\text{adpp}\,(\mathit{adp},\ \text{p-data-segment}\,(\text{lr-eval}\,(\mathit{flag},\ \text{lr-set-pos}\,(l,\ \mathit{pos}),\ c))) \\ & \quad = \ \text{adpp}\,(\mathit{adp},\ \text{p-data-segment}\,(l))) \end{array}$$

THEOREM: lr-eval-preserves-lr-max-node-lr-set-pos
(proper-p-statep (lr->p (l))
 ∧ good-posp (*flag*, *pos*, program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ (p-psw (lr-eval (*flag*, lr-set-pos (l, *pos*), c)) = `'run`))
→ (lr-max-node (p-data-segment (lr-eval (*flag*, lr-set-pos (l, *pos*), c)))
 = lr-max-node (p-data-segment (l)))

THEOREM: lr-eval-preserves-lr-proper-free-listp
(proper-p-statep (lr->p (l))
 ∧ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ (p-psw (lr-eval (*flag*, l, c)) = `'run`)
 ∧ adpp (untag (lr-max-node (p-data-segment (l))), p-data-segment (l))
 ∧ lr-boundary-nodep (lr-max-node (p-data-segment (l)))
 ∧ lr-proper-free-listp (p-data-segment (l)))
→ lr-proper-free-listp (p-data-segment (lr-eval (*flag*, l, c)))

THEOREM: lr-apply-subr-preserves-lr-valp
let *new-l* **be** lr-eval (`'list`, lr-set-pos (l, *pos*), c)
in
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ listp (lr-expr (l))
 ∧ subrp (car (lr-expr (l)))
 ∧ (car (lr-expr (l)) ≠ `'if`)
 ∧ proper-p-statep (lr->p (*new-l*))
 ∧ lr-proper-free-listp (p-data-segment (*new-l*))
 ∧ adpp (untag (lr-max-node (p-data-segment (*new-l*))),
 p-data-segment (*new-l*))
 ∧ lr-boundary-nodep (lr-max-node (p-data-segment (*new-l*)))
 ∧ (p-psw (*new-l*) = `'run`)
 ∧ (p-psw (lr-apply-subr (l, *new-l*)) = `'run`)
 ∧ lr-valp (*value*, *addr*, p-data-segment (*new-l*)))
→ lr-valp (*value*, *addr*, p-data-segment (lr-apply-subr (l, *new-l*))) **endlet**

THEOREM: lr-eval-preserves-lr-proper-free-listp-lr-set-pos
(proper-p-statep (lr->p (l))
 ∧ good-posp (*flag*, *pos*, program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ (p-psw (lr-eval (*flag*, lr-set-pos (l, *pos*), c)) = `'run`)
 ∧ adpp (untag (lr-max-node (p-data-segment (l))), p-data-segment (l))
 ∧ lr-boundary-nodep (lr-max-node (p-data-segment (l)))
 ∧ lr-proper-free-listp (p-data-segment (l)))
→ lr-proper-free-listp (p-data-segment (lr-eval (*flag*, lr-set-pos (l, *pos*), c)))

THEOREM: lr-eval-preserves-lr-valp

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{proper-p-statep}\,(\text{lr->p}\,(l))} \\
\wedge & \text{good-posp}\,(\mathit{flag},\ \text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
\wedge & \text{lr-programs-properp}\,(l,\ \mathit{table}) \\
\wedge & (\text{p-psw}\,(\text{lr-eval}\,(\mathit{flag},\ l,\ c)) = \texttt{'run}) \\
\wedge & \text{lr-proper-free-listp}\,(\text{p-data-segment}\,(l)) \\
\wedge & \text{adpp}\,(\text{untag}\,(\text{lr-max-node}\,(\text{p-data-segment}\,(l))),\ \text{p-data-segment}\,(l)) \\
\wedge & \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\text{p-data-segment}\,(l))) \\
\wedge & \text{lr-valp}\,(\mathit{value},\ \mathit{addr},\ \text{p-data-segment}\,(l))) \\
\rightarrow & \text{lr-valp}\,(\mathit{value},\ \mathit{addr},\ \text{p-data-segment}\,(\text{lr-eval}\,(\mathit{flag},\ l,\ c)))
\end{array}
$$

THEOREM: lr-check-f-addrp-deposit-anything-anywhere

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{lr-check-f-addrp}\,(\mathit{addr},\ \text{deposit}\,(\mathit{anything},\ \mathit{anywhere},\ \mathit{data\text{-}seg}))} \\
= & \text{lr-check-f-addrp}\,(\mathit{addr},\ \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: lr-check-undef-addrp-deposit-anything-anywhere

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{lr-check-undef-addrp}\,(\mathit{addr},\ \text{deposit}\,(\mathit{anything},\ \mathit{anywhere},\ \mathit{data\text{-}seg}))} \\
= & \text{lr-check-undef-addrp}\,(\mathit{addr},\ \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: lr-check-listp-addrp-deposit-free-ptr-0

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{lr-check-listp-addrp}\,(\mathit{addr},\ \text{deposit}\,(\mathit{any},\ \text{identity}\,(\textsc{lr-fp-addr}),\ \mathit{data\text{-}seg}))} \\
= & \text{lr-check-listp-addrp}\,(\mathit{addr},\ \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: lr-check-numberp-addrp-deposit-free-ptr-0

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{lr-check-numberp-addrp}\,(\mathit{addr},\ \text{deposit}\,(\mathit{any},\ \text{identity}\,(\textsc{lr-fp-addr}),\ \mathit{data\text{-}seg}))} \\
= & \text{lr-check-numberp-addrp}\,(\mathit{addr},\ \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: lr-proper-heapp-nodep-deposit-free-ptr-0

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{lr-proper-heapp-nodep}\,(\mathit{addr},\ \text{deposit}\,(\mathit{any},\ \text{identity}\,(\textsc{lr-fp-addr}),\ \mathit{data\text{-}seg}))} \\
= & \text{lr-proper-heapp-nodep}\,(\mathit{addr},\ \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: lr-proper-heapp2-deposit-free-ptr-0

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{lr-proper-heapp2}\,(\mathit{addr},\ \text{deposit}\,(\mathit{any},\ \text{identity}\,(\textsc{lr-fp-addr}),\ \mathit{data\text{-}seg}))} \\
= & \text{lr-proper-heapp2}\,(\mathit{addr},\ \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: lr-boundary-offsetp-equal-plus-fact-zero

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-boundary-offsetp}\,(\mathit{offset1})} \\
\wedge & \text{lr-boundary-offsetp}\,(\mathit{offset2}) \\
\wedge & (n < \textsc{lr-node-size}) \\
\wedge & (\mathit{offset1} \in \mathbf{N}) \\
\wedge & (\mathit{offset2} \in \mathbf{N})) \\
\rightarrow & (((n + \mathit{offset1}) = \mathit{offset2}) \\
 & \quad = \quad ((n \simeq \mathsf{0}) \wedge (\mathit{offset1} = \mathit{offset2})))
\end{array}
$$

THEOREM: fetch-add-addr-deposit-a-list-node

(adpp (untag (*max-addr*), *data-seg*)
 ∧ lr-boundary-nodep (*max-addr*)
 ∧ adpp (untag (*addr*), *data-seg*)
 ∧ lr-boundary-nodep (*addr*)
 ∧ (area-name (*addr*) = area-name (*max-addr*))
 ∧ ($n <$ LR-NODE-SIZE))
→ (fetch (add-addr (*max-addr*, *n*),
          deposit-a-list (list (*x0*, *x1*, *x2*, *x3*), *addr*, *data-seg*))
    = **if** offset (*addr*) = offset (*max-addr*)
      then get (*n*, list (*x0*, *x1*, *x2*, *x3*))
      else fetch (add-addr (*max-addr*, *n*), *data-seg*) **endif**)

THEOREM: fetch-deposit-a-list-node
(adpp (untag (*max-addr*), *data-seg*)
 ∧ lr-boundary-nodep (*max-addr*)
 ∧ adpp (untag (*addr*), *data-seg*)
 ∧ lr-boundary-nodep (*addr*)
 ∧ (area-name (*addr*) = area-name (*max-addr*)))
→ (fetch (*max-addr*, deposit-a-list (list (*x0*, *x1*, *x2*, *x3*), *addr*, *data-seg*))
    = **if** offset (*addr*) = offset (*max-addr*) **then** *x0*
      else fetch (*max-addr*, *data-seg*) **endif**)

EVENT: Disable lr-boundary-offsetp-equal-plus-fact-zero.

THEOREM: lr-check-f-addrp-deposit-a-list
lr-check-f-addrp (*addr*, deposit-a-list (*list*, *anywhere*, *data-seg*))
= lr-check-f-addrp (*addr*, *data-seg*)

THEOREM: lr-check-undef-addrp-deposit-a-list
lr-check-undef-addrp (*addr*, deposit-a-list (*list*, *anywhere*, *data-seg*))
= lr-check-undef-addrp (*addr*, *data-seg*)

THEOREM: lr-check-numberp-addrp-deposit-a-list-cons
((offset (*addr*) ≠ offset (*max-addr*))
 ∧ (type (*max-addr*) = `'addr`)
 ∧ (cddr (*max-addr*) = **nil**)
 ∧ listp (*max-addr*)
 ∧ adpp (untag (*max-addr*), *data-seg*)
 ∧ lr-boundary-nodep (*max-addr*)
 ∧ (area-name (*max-addr*) = LR-HEAP-NAME)
 ∧ (type (*addr*) = `'addr`)
 ∧ (cddr (*addr*) = **nil**)
 ∧ listp (*addr*)
 ∧ adpp (untag (*addr*), *data-seg*)

$\land$ lr-boundary-nodep (*addr*)
$\land$ (area-name (*addr*) = LR-HEAP-NAME)
$\land$ lr-check-numberp-addrp (*max-addr*, *data-seg*)
$\land$ (type (*tag*) = `'nat`)
$\land$ (untag (*tag*) $\in \mathbf{N}$))
$\rightarrow$ lr-check-numberp-addrp (*max-addr*,
deposit-a-list (list (*x0*, *tag*, *x2*, *x3*),
addr,
data-seg))

THEOREM: lr-check-listp-addrp-deposit-a-list-cons
(lr-good-pointerp (*good-pointer1*, *data-seg*)
$\land$ lr-good-pointerp (*good-pointer2*, *data-seg*)
$\land$ adpp (untag (*addr*), *data-seg*)
$\land$ lr-boundary-nodep (*addr*)
$\land$ adpp (untag (*max-addr*), *data-seg*)
$\land$ lr-boundary-nodep (*max-addr*)
$\land$ (area-name (*addr*) = LR-HEAP-NAME)
$\land$ (area-name (*max-addr*) = LR-HEAP-NAME)
$\land$ (type (*tag*) = `'nat`)
$\land$ (untag (*tag*) $\in \mathbf{N}$)
$\land$ (offset (*addr*) = offset (*max-addr*)))
$\rightarrow$ lr-check-listp-addrp (*max-addr*,
deposit-a-list (list (identity (tag (`'nat`,
LR-CONS-TAG)),
tag,
good-pointer1,
good-pointer2),
addr,
data-seg))

THEOREM: lr-check-listp-addrp-deposit-a-list-other-place
(adpp (untag (*addr*), *data-seg*)
$\land$ lr-boundary-nodep (*addr*)
$\land$ adpp (untag (*max-addr*), *data-seg*)
$\land$ lr-boundary-nodep (*max-addr*)
$\land$ (area-name (*addr*) = LR-HEAP-NAME)
$\land$ (area-name (*max-addr*) = LR-HEAP-NAME)
$\land$ lr-check-listp-addrp (*max-addr*, *data-seg*)
$\land$ (offset (*addr*) $\neq$ offset (*max-addr*))
$\land$ (type (*ref-count*) = `'nat`))
$\rightarrow$ lr-check-listp-addrp (*max-addr*,
deposit-a-list (list (*x0*, *ref-count*, *x2*, *x3*),
addr,
data-seg))

THEOREM: lr-proper-heapp-nodep-deposit-a-list-cons

$$
\begin{array}{l}
(\text{lr-nodep}\,(\mathit{max\text{-}addr},\ \mathit{data\text{-}seg}) \\
\ \land\ \ \text{lr-nodep}\,(\mathit{addr},\ \mathit{data\text{-}seg}) \\
\ \land\ \ \text{lr-proper-heapp-nodep}\,(\mathit{max\text{-}addr},\ \mathit{data\text{-}seg}) \\
\ \land\ \ (\text{type}\,(\text{fetch}\,(\text{add-addr}\,(\mathit{addr},\ \text{LR-REF-COUNT-OFFSET}),\ \mathit{data\text{-}seg})) \neq \texttt{'nat}) \\
\ \land\ \ \text{lr-good-pointerp}\,(\mathit{good\text{-}pointer1},\ \mathit{data\text{-}seg}) \\
\ \land\ \ \text{lr-good-pointerp}\,(\mathit{good\text{-}pointer2},\ \mathit{data\text{-}seg}) \\
\ \land\ \ (\text{type}\,(\mathit{tag}) = \texttt{'nat}) \\
\ \land\ \ (\text{untag}\,(\mathit{tag}) \in \mathbf{N})) \\
\ \to\ \ \text{lr-proper-heapp-nodep}\,(\mathit{max\text{-}addr}, \\
\qquad\qquad \text{deposit-a-list}\,(\text{list}\,(\text{identity}\,(\text{tag}\,(\texttt{'nat},\ \text{LR-CONS-TAG})), \\
\qquad\qquad\qquad \mathit{tag},\ \mathit{good\text{-}pointer1},\ \mathit{good\text{-}pointer2}), \\
\qquad\qquad\quad \mathit{addr}, \\
\qquad\qquad\quad \mathit{data\text{-}seg}))
\end{array}
$$

THEOREM: lr-proper-heapp2-deposit-a-list-cons

$$
\begin{array}{l}
(\text{lr-nodep}\,(\mathit{max\text{-}addr},\ \mathit{data\text{-}seg}) \\
\ \land\ \ \text{lr-nodep}\,(\mathit{addr},\ \mathit{data\text{-}seg}) \\
\ \land\ \ \text{lr-proper-heapp2}\,(\mathit{max\text{-}addr},\ \mathit{data\text{-}seg}) \\
\ \land\ \ (\text{type}\,(\text{fetch}\,(\text{add-addr}\,(\mathit{addr},\ \text{LR-REF-COUNT-OFFSET}),\ \mathit{data\text{-}seg})) \neq \texttt{'nat}) \\
\ \land\ \ \text{lr-good-pointerp}\,(\mathit{good\text{-}pointer1},\ \mathit{data\text{-}seg}) \\
\ \land\ \ \text{lr-good-pointerp}\,(\mathit{good\text{-}pointer2},\ \mathit{data\text{-}seg}) \\
\ \land\ \ (\text{type}\,(\mathit{tag}) = \texttt{'nat}) \\
\ \land\ \ (\text{untag}\,(\mathit{tag}) \in \mathbf{N})) \\
\ \to\ \ \text{lr-proper-heapp2}\,(\mathit{max\text{-}addr}, \\
\qquad\qquad \text{deposit-a-list}\,(\text{list}\,(\text{identity}\,(\text{tag}\,(\texttt{'nat},\ \text{LR-CONS-TAG})), \\
\qquad\qquad\qquad \mathit{tag},\ \mathit{good\text{-}pointer1},\ \mathit{good\text{-}pointer2}), \\
\qquad\qquad\quad \mathit{addr}, \\
\qquad\qquad\quad \mathit{data\text{-}seg}))
\end{array}
$$

THEOREM: not-psw-run-lr-eval

$$(\text{p-psw}\,(l) \neq \texttt{'run}) \to (\text{lr-eval}\,(\mathit{flag},\ l,\ c) = l)$$

THEOREM: program-body-assoc-lr-compile-programs

$$
\begin{array}{l}
\text{program-body}\,(\text{assoc}\,(\mathit{name},\ \text{lr-compile-programs}\,(\mathit{progs},\ \mathit{table}))) \\
=\ \ \text{lr-compile-body}\,(\mathbf{t}, \\
\qquad\qquad \text{s-body}\,(\text{assoc}\,(\mathit{name},\ \mathit{progs})), \\
\qquad\qquad \text{lr-make-temp-name-alist}\,(\text{s-temp-list}\,(\text{assoc}\,(\mathit{name},\ \mathit{progs})), \\
\qquad\qquad\qquad \text{s-formals}\,(\text{assoc}\,(\mathit{name},\ \mathit{progs}))), \\
\qquad\qquad \mathit{table})
\end{array}
$$

THEOREM: listp-lr-compile-body
$\text{listp}(\text{lr-compile-body}(\textit{flag}, \textit{body}, \textit{temp-name-alist}, \textit{table})) = \text{listp}(\textit{body})$

THEOREM: car-lr-compile-body
$(\textit{flag} \neq \texttt{'list})$
$\rightarrow \quad (\text{car}(\text{lr-compile-body}(\textit{flag}, \textit{body}, \textit{temp-name-alist}, \textit{table})) = \text{car}(\textit{body}))$

THEOREM: good-posp1-expand-list-temps
$(((\textit{temp} = \text{S-TEMP-EVAL}) \vee (\textit{temp} = \text{S-TEMP-TEST})) \wedge \text{listp}(\textit{pos}))$
$\rightarrow \quad (\text{good-posp1}(\textit{pos}, \text{list}(\textit{temp}, \textit{body}, \textit{name}))$
$\qquad = \quad ((\text{car}(\textit{pos}) = \texttt{1}) \wedge \text{good-posp1}(\text{cdr}(\textit{pos}), \textit{body})))$

THEOREM: length-lr-compile-body-list
$\text{length}(\text{lr-compile-body}(\texttt{'list}, \textit{body}, \textit{temp-name-alist}, \textit{table}))$
$= \quad \text{length}(\textit{body})$

THEOREM: get-lr-compile-body-list
$\text{get}(n, \text{lr-compile-body}(\texttt{'list}, \textit{body}, \textit{temp-name-alist}, \textit{table}))$
$= \quad \text{lr-compile-body}(\mathbf{t}, \text{get}(n, \textit{body}), \textit{temp-name-alist}, \textit{table})$

THEOREM: get-lr-compile-body
$(\text{listp}(\textit{body})$
$\wedge \quad (\text{car}(\textit{body}) \neq \text{S-TEMP-FETCH})$
$\wedge \quad (\text{car}(\textit{body}) \neq \text{S-TEMP-EVAL})$
$\wedge \quad (\text{car}(\textit{body}) \neq \text{S-TEMP-TEST})$
$\wedge \quad (\text{car}(\textit{body}) \neq \texttt{'quote})$
$\wedge \quad (n \not\simeq \texttt{0}))$
$\rightarrow \quad (\text{get}(n, \text{lr-compile-body}(\mathbf{t}, \textit{body}, \textit{temp-name-alist}, \textit{table}))$
$\qquad = \quad \text{lr-compile-body}(\mathbf{t}, \text{get}(n, \textit{body}), \textit{temp-name-alist}, \textit{table}))$

THEOREM: length-lr-compile-body-t
$(\text{listp}(\textit{body})$
$\wedge \quad (\text{car}(\textit{body}) \neq \text{S-TEMP-FETCH})$
$\wedge \quad (\text{car}(\textit{body}) \neq \text{S-TEMP-EVAL})$
$\wedge \quad (\text{car}(\textit{body}) \neq \text{S-TEMP-TEST})$
$\wedge \quad (\text{car}(\textit{body}) \neq \texttt{'quote}))$
$\rightarrow \quad (\text{length}(\text{lr-compile-body}(\mathbf{t}, \textit{body}, \textit{temp-name-alist}, \textit{table}))$
$\qquad = \quad \text{length}(\textit{body}))$

THEOREM: good-posp1-lr-compile-body
$\text{good-posp1}(\textit{pos}, \text{lr-compile-body}(\mathbf{t}, \textit{body}, \textit{temp-name-alist}, \textit{table}))$
$= \quad \text{good-posp1}(\textit{pos}, \textit{body})$

THEOREM: cur-expr-lr-compile-body-t
$\text{good-posp1}(\textit{pos}, \textit{body})$
$\rightarrow \quad (\text{cur-expr}(\textit{pos}, \text{lr-compile-body}(\mathbf{t}, \textit{body}, \textit{temp-name-alist}, \textit{table}))$
$\qquad = \quad \text{lr-compile-body}(\mathbf{t}, \text{cur-expr}(\textit{pos}, \textit{body}), \textit{temp-name-alist}, \textit{table}))$

THEOREM: lr-check-result1-singleton-list-opener
lr-check-result1 (list (x), *temp-stk*, *data-seg*)
$=$ lr-valp (x, car (*temp-stk*), *data-seg*)

THEOREM: proper-p-temp-stkp-plistp-p-temp-stk
proper-p-temp-stkp (*temp-stk*, p) $\rightarrow$ plistp (*temp-stk*)

THEOREM: proper-p-statep-lr->p-plistp-p-temp-stk
proper-p-statep (lr->p (l)) $\rightarrow$ plistp (p-temp-stk (l))

THEOREM: proper-p-statep-lr->p-not-0-p-temp-stk
proper-p-statep (lr->p (l)) $\rightarrow$ (p-temp-stk (l) $\neq$ `0`)

THEOREM: plistp-lastcdr-nil
plistp (*list*) $\rightarrow$ (lastcdr (*list*) $=$ **nil**)

THEOREM: lr-eval-preserves-lr-valp-lr-set-expr
(proper-p-statep (lr->p (l))
$\wedge$ proper-p-statep (lr->p (lr-set-expr (*l1*, l, *pos*)))
$\wedge$ good-posp (*flag*, *pos*, program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (lr-eval (*flag*, lr-set-expr (*l1*, l, *pos*), c)) $=$ `'run`)
$\wedge$ lr-proper-free-listp (p-data-segment (l))
$\wedge$ lr-proper-free-listp (p-data-segment (*l1*))
$\wedge$ adpp (untag (lr-max-node (p-data-segment (l))), p-data-segment (l))
$\wedge$ lr-boundary-nodep (lr-max-node (p-data-segment (l)))
$\wedge$ adpp (untag (lr-max-node (p-data-segment (*l1*))), p-data-segment (*l1*))
$\wedge$ lr-boundary-nodep (lr-max-node (p-data-segment (*l1*)))
$\wedge$ lr-valp (*value*, *addr*, p-data-segment (*l1*))
$\wedge$ (length (cdr (assoc (LR-HEAP-NAME, p-data-segment (*l1*))))
$=$ length (cdr (assoc (LR-HEAP-NAME, p-data-segment (l))))))
$\rightarrow$ lr-valp (*value*,
addr,
p-data-segment (lr-eval (*flag*, lr-set-expr (*l1*, l, *pos*), c)))

THEOREM: lr-eval-preserves-proper-p-statep-lr->p-lr-set-expr
(proper-p-statep (lr->p (l))
$\wedge$ good-posp (*flag*, *pos*, program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (lr-eval (*flag*, lr-set-expr (*l1*, l, *pos*), c)) $=$ `'run`)
$\wedge$ lr-programs-properp (*l1*, *table*)
$\wedge$ proper-p-statep (lr->p (*l1*))
$\wedge$ (cdr (p-ctrl-stk (*l1*)) $=$ cdr (p-ctrl-stk (l)))
$\wedge$ (strip-cars (bindings (car (p-ctrl-stk (*l1*))))
$=$ strip-cars (bindings (car (p-ctrl-stk (l)))))

$\land$ (p-prog-segment (*l1*) = p-prog-segment (*l*))
$\land$ (p-word-size (*l1*) = p-word-size (*l*))
$\land$ (p-max-ctrl-stk-size (*l1*) = p-max-ctrl-stk-size (*l*))
$\land$ (p-max-temp-stk-size (*l1*) = p-max-temp-stk-size (*l*)))
$\rightarrow$ proper-p-statep (lr->p (lr-eval (*flag*, lr-set-expr (*l1*, *l*, *pos*), *c*)))

THEOREM: lr-check-result-flag-list-cons-value
let *l2* **be** lr-eval (`'list`, lr-set-expr (lr-eval (**t**, *l*, *c*), *l*, nx (*pos*)), *c*)
in
(good-posp (`'list`, *pos*, program-body (p-current-program (*l*)))
$\land$ lr-programs-properp (*l*, *table*)
$\land$ proper-p-statep (lr->p (*l*))
$\land$ listp (lr-expr-list (*l*))
$\land$ listp (offset (p-pc (*l*)))
$\land$ lr-proper-heapp (p-data-segment (*l*))
$\land$ lr-check-result (**t**,
value1,
p-temp-stk (lr-eval (**t**, *l*, *c*)),
p-data-segment (lr-eval (**t**, *l*, *c*)),
p-temp-stk (*l*))
$\land$ lr-check-result (`'list`,
value2,
p-temp-stk (*l2*),
p-data-segment (*l2*),
p-temp-stk (lr-eval (**t**, *l*, *c*)))
$\land$ (p-psw (*l2*) = `'run`)
$\land$ (*pos* = offset (p-pc (*l*)))
$\land$ (*temp-stk* = p-temp-stk (*l*)))
$\rightarrow$ lr-check-result (`'list`,
cons (*value1*, *value2*),
p-temp-stk (*l2*),
p-data-segment (*l2*),
temp-stk) **endlet**

THEOREM: lr-check-result-nil
lr-proper-heapp (*data-seg*)
$\rightarrow$ lr-check-result (`'list`, **nil**, *temp-stk*, *data-seg*, *temp-stk*)

THEOREM: litatom-lr-compile-body
litatom (lr-compile-body (**t**, *body*, *temp-name-alist*, *table*)) = litatom (*body*)

THEOREM: lr-params-lr-push-tstk
lr-params (*frame*, lr-push-tstk (*l*, *anything*)) = lr-params (*frame*, *l*)

THEOREM: lr-temps-lr-push-tstk
lr-temps (*frame*, lr-push-tstk (*l*, *anything*)) = lr-temps (*frame*, *l*)

THEOREM: program-body-p-current-program-s->lr1
program-body (p-current-program (s->lr1 (s, l, *table*)))
$=$ lr-compile-body (**t**,
s-body (s-prog (s)),
lr-make-temp-name-alist (s-temp-list (s-prog (s)),
s-formals (s-prog (s))),
table)

THEOREM: name-car-lr-compile-programs-progs
name (car (lr-compile-programs (s-progs (s), *table*))) $=$ caar (s-progs (s))

THEOREM: car-car-lr-compile-programs-progs
caar (lr-compile-programs (s-progs (s), *table*)) $=$ caar (s-progs (s))

THEOREM: s-good-statep-program-body-car-lr-compile-programs
s-good-statep (s, c)
$\rightarrow$ (program-body (car (lr-compile-programs (s-progs (s), *table*)))
$=$ lr-compile-body (**t**,
s-body (car (s-progs (s))),
lr-make-temp-name-alist (s-temp-list (car (s-progs (s))),
s-formals (car (s-progs (s)))),
table))

THEOREM: good-posp-lr-compile-body
good-posp (*flag*, *pos*, lr-compile-body (**t**, *body*, *temp-name-alist*, *table*))
$=$ good-posp (*flag*, *pos*, *body*)

THEOREM: strip-cars-lr-compile-programs
strip-cars (lr-compile-programs (*progs*, *table*)) $=$ strip-cars (*progs*)

THEOREM: listp-lr-expr-list-s->lr1
good-posp (`'list`, s-pos (s), s-body (s-prog (s)))
$\rightarrow$ (listp (lr-expr-list (s->lr1 (s, l, *table*))) $=$ listp (s-expr-list (s)))

THEOREM: formal-vars-lr-compile-programs
formal-vars (assoc (*name*, lr-compile-programs (*progs*, *table*)))
$=$ s-formals (assoc (*name*, *progs*))

THEOREM: formal-vars-p-current-program-s->lr1
formal-vars (p-current-program (s->lr1 (s, l, *table*))) $=$ s-formals (s-prog (s))

THEOREM: temp-var-dcls-lr-compile-programs
definedp (*name*, *progs*)
$\rightarrow$ (temp-var-dcls (assoc (*name*, lr-compile-programs (*progs*, *table*)))
$=$ lr-make-temp-var-dcls (lr-make-temp-name-alist (s-temp-list (assoc (*name*,
progs)),
s-formals (assoc (*name*,
progs)))))

Theorem: temp-var-dcls-assoc-p-current-program-s->lr1
definedp (s-pname (s), s-progs (s))
$\rightarrow$ (temp-var-dcls (p-current-program (s->lr1 (s, l, *table*)))
$=$ lr-make-temp-var-dcls (lr-make-temp-name-alist (s-temp-list (s-prog (s)),
s-formals (s-prog (s)))))

Theorem: lr-set-expr-s->lr1-s-set-expr
s->lr1 (s-set-expr (s-eval (**t**, s, c), s, nx (s-pos (s))),
lr-eval (**t**, s->lr1 (s, l, *table*), c),
table)
$=$ lr-set-error (lr-set-expr (lr-eval (**t**, s->lr1 (s, l, *table*), c),
s->lr1 (s, l, *table*),
nx (s-pos (s))),
s-err-flag (s-eval (**t**, s, c)))

Theorem: p-current-program-lr-set-error
p-current-program (lr-set-error (l, *err-flag*)) $=$ p-current-program (l)

Theorem: lr-set-error-lr-set-error
lr-set-error (lr-set-error (l, *err-flag1*), *err-flag2*)
$=$ lr-set-error (l, *err-flag2*)

Theorem: proper-p-statep-lr-set-error
proper-p-statep (lr-set-error (l, *err-flag*)) $=$ proper-p-statep (l)

Theorem: lr-params-lr-set-error
lr-params (*frame*, lr-set-error (l, *err-flag*)) $=$ lr-params (*frame*, l)

Theorem: lr-temps-lr-set-error
lr-temps (*frame*, lr-set-error (l, *err-flag*)) $=$ lr-temps (*frame*, l)

Theorem: lr-s-similar-statesp-lr-set-error
lr-s-similar-statesp (*params*, *temps*, lr-set-error (l, *err-flag*), *table*)
$=$ lr-s-similar-statesp (*params*, *temps*, l, *table*)

Theorem: lr-s-similar-statesp-lr-set-expr
((area-name (p-pc (*l1*)) $=$ area-name (p-pc (*l2*)))
$\wedge$ (p-prog-segment (*l1*) $=$ p-prog-segment (*l2*)))
$\rightarrow$ (lr-s-similar-statesp (*params*, *temps*, lr-set-expr (*l1*, *l2*, *pos*), *table*)
$=$ lr-s-similar-statesp (*params*, *temps*, *l1*, *table*))

Theorem: lr-eval-zerop-clock
(($c \simeq$ `0`) $\wedge$ (*flag* $\neq$ `'list`) $\wedge$ (p-psw (l) $=$ `'run`))
$\rightarrow$ (lr-eval (*flag*, l, c) $=$ lr-set-error (l, `'out-of-time`))

THEOREM: litatom-lr-expr-s->lr1
(good-posp1 (s-pos (*s*), s-body (s-prog (*s*))) ∧ litatom (s-expr (*s*)))
→ (lr-expr (s->lr1 (*s*, *l*, *table*)) = s-expr (*s*))

THEOREM: lr-eval-litatom-opener
(good-posp1 (s-pos (*s*), s-body (s-prog (*s*)))
∧ (*flag* ≠ `'list`)
∧ (*c* ≄ `0`)
∧ litatom (s-expr (*s*))
∧ (s-err-flag (*s*) = `'run`))
→ (lr-eval (*flag*, s->lr1 (*s*, *l*, *table*), *c*)
= lr-push-tstk (s->lr1 (*s*, *l*, *table*),
local-var-value (s-expr (*s*), p-ctrl-stk (*l*))))

THEOREM: lr-s-similar-statesp-lr-push-tstk-litatom
lr-s-similar-statesp (*s-params*, *s-temps*, lr-push-tstk (*l*, *value*), *table*)
= lr-s-similar-statesp (*s-params*, *s-temps*, *l*, *table*)

THEOREM: lr-s-similar-params-assoc-definedp
(lr-s-similar-params (*s-params*, *lr-params*, *data-seg*)
∧ definedp (*name*, *lr-params*))
→ lr-valp (cdr (assoc (*name*, *s-params*)), cdr (assoc (*name*, *lr-params*)), *data-seg*)

THEOREM: proper-p-statep-lr->p-strip-cars-bindings-ctrl-stk
(proper-p-statep (lr->p (*l*))
∧ definedp (area-name (p-pc (*l*)), p-prog-segment (*l*)))
→ (strip-cars (bindings (car (p-ctrl-stk (*l*))))
= append (formal-vars (assoc (area-name (p-pc (*l*)), p-prog-segment (*l*))),
strip-cars (temp-var-dcls (assoc (area-name (p-pc (*l*)),
p-prog-segment (*l*))))))

DEFINITION:
induct-hint-11 (*v*, *y*)
= **if** listp (*v*)
then if listp (*y*) **then** induct-hint-11 (cdr (*v*), cdr (*y*))
else t endif
else t endif

THEOREM: equal-append-same-length-fact
(length (*v*) = length (*y*))
→ ((append (strip-cars (*v*), *w*) = append (*y*, *z*))
= ((strip-cars (*v*) = plist (*y*)) ∧ (*w* = *z*)))

THEOREM: definedp-strip-cars-append-member-x
(strip-cars (*x*) = append (*y*, *z*))
→ ((*e* ∈ *y*) = definedp (*e*, firstn (length (*y*), *x*)))

THEOREM: proper-p-statep-lr->p-member-formals-definedp-bindings
$$
\begin{array}{l}
(\text{proper-p-statep}\,(\text{lr->p}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table})))\\
\ \wedge\quad \text{definedp}\,(\text{s-pname}\,(s),\ \text{s-progs}\,(s))\\
\ \wedge\quad (x \in \text{s-formals}\,(\text{assoc}\,(\text{s-pname}\,(s),\ \text{s-progs}\,(s)))))\\
\rightarrow\quad \text{definedp}\,(x,\\
\qquad\qquad\quad \text{firstn}\,(\text{length}\,(\text{s-formals}\,(\text{assoc}\,(\text{s-pname}\,(s),\ \text{s-progs}\,(s)))),\\
\qquad\qquad\qquad\quad \text{bindings}\,(\text{car}\,(\text{p-ctrl-stk}\,(l)))))
\end{array}
$$

THEOREM: lr-valp-addr-0
$$\neg\ \text{lr-valp}\,(\mathit{addr},\ \mathtt{0},\ \mathit{data\text{-}seg})$$

THEOREM: lr-valp-cdr-assoc-firstn-cdr-assoc
$$
\begin{array}{l}
\text{lr-valp}\,(\mathit{addr},\ \text{cdr}\,(\text{assoc}\,(\mathit{name},\ \text{firstn}\,(n,\ \mathit{list}))),\ \mathit{data\text{-}seg})\\
\rightarrow\quad \text{lr-valp}\,(\mathit{addr},\ \text{cdr}\,(\text{assoc}\,(\mathit{name},\ \mathit{list})),\ \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: lr-s-similar-statesp-lr-s-similar-params-opener
$$
\begin{array}{l}
(\text{lr-s-similar-statesp}\,(\mathit{s\text{-}params},\ \mathit{s\text{-}temps},\ l,\ \mathit{table})\\
\ \wedge\quad (\mathit{frame} = \text{car}\,(\text{p-ctrl-stk}\,(l)))\\
\ \wedge\quad (\mathit{data\text{-}seg} = \text{p-data-segment}\,(l)))\\
\rightarrow\quad \text{lr-s-similar-params}\,(\mathit{s\text{-}params},\ \text{lr-params}\,(\mathit{frame},\ l),\ \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: lr-s-similar-statesp-lr-s-similar-temps-opener
$$
\begin{array}{l}
(\text{lr-s-similar-statesp}\,(\mathit{s\text{-}params},\ \mathit{s\text{-}temps},\ l,\ \mathit{table})\\
\ \wedge\quad (\mathit{frame} = \text{car}\,(\text{p-ctrl-stk}\,(l)))\\
\ \wedge\quad (\mathit{data\text{-}seg} = \text{p-data-segment}\,(l)))\\
\rightarrow\quad \text{lr-s-similar-temps}\,(\mathit{s\text{-}temps},\ \text{lr-temps}\,(\mathit{frame},\ l),\ \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: strip-cars-lr-make-temp-var-dcls
$$\text{strip-cars}\,(\text{lr-make-temp-var-dcls}\,(\mathit{temp\text{-}alist})) = \text{strip-cdrs}\,(\mathit{temp\text{-}alist})$$

THEOREM: lr-check-result-lr-push-tstk
$$
\begin{array}{l}
\textbf{let}\ \mathit{value}\ \ \textbf{be}\ \ \text{cdr}\,(\text{assoc}\,(\text{s-expr}\,(s),\ \text{bindings}\,(\text{car}\,(\text{p-ctrl-stk}\,(l)))))\\
\textbf{in}\\
(\text{good-posp1}\,(\text{s-pos}\,(s),\ \text{s-body}\,(\text{s-prog}\,(s)))\\
\ \wedge\quad \text{definedp}\,(\text{s-pname}\,(s),\ \text{s-progs}\,(s))\\
\ \wedge\quad (\text{s-err-flag}\,(s) = \mathtt{'run})\\
\ \wedge\quad \text{litatom}\,(\text{s-expr}\,(s))\\
\ \wedge\quad \text{proper-p-statep}\,(\text{lr->p}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table})))\\
\ \wedge\quad \text{lr-proper-heapp}\,(\text{p-data-segment}\,(l))\\
\ \wedge\quad \text{lr-programs-properp}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table})\\
\ \wedge\quad (\text{p-psw}\,(\text{lr-push-tstk}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{value})) = \mathtt{'run})\\
\ \wedge\quad \text{lr-s-similar-statesp}\,(\text{s-params}\,(s),\\
\qquad\qquad\qquad\qquad\quad \text{s-temps}\,(s),\\
\qquad\qquad\qquad\qquad\quad \text{s->lr1}\,(s,\ l,\ \mathit{table}),\\
\qquad\qquad\qquad\qquad\quad \mathit{table}))
\end{array}
$$

→ lr-check-result (**t**,
cdr (assoc (s-expr (*s*), s-params (*s*))),
p-temp-stk (lr-push-tstk (s->lr1 (*s*, *l*, *table*),
value)),
p-data-segment (*l*),
p-temp-stk (*l*)) **endlet**

THEOREM: s->lr1-s-set-pos-lr-set-pos
s->lr1 (s-set-pos (*s*, *pos*), *l*, *table*) = lr-set-pos (s->lr1 (*s*, *l*, *table*), *pos*)

THEOREM: lr-params-lr-set-pos
lr-params (*frame*, lr-set-pos (*l*, *pos*)) = lr-params (*frame*, *l*)

THEOREM: lr-temps-lr-set-pos
lr-temps (*frame*, lr-set-pos (*l*, *pos*)) = lr-temps (*frame*, *l*)

THEOREM: lr-s-similar-statesp-lr-s-set-pos
lr-s-similar-statesp (*s-params*, *s-temps*, lr-set-pos (*l*, *pos*), *table*)
= lr-s-similar-statesp (*s-params*, *s-temps*, *l*, *table*)

THEOREM: lr-set-expr-s->lr1-s-set-expr-lr-pop-tstk
s->lr1 (s-set-expr (s-eval (**t**, s-set-pos (*s*, *pos*), *c*), *s*, dv (s-pos (*s*), *n*)),
lr-pop-tstk (lr-if-ok (lr-eval (**t**, lr-set-pos (s->lr1 (*s*, *l*, *table*), *pos*), *c*))),
table)
= lr-set-error (lr-set-expr (lr-pop-tstk (lr-if-ok (lr-eval (**t**,
lr-set-pos (s->lr1 (*s*,
l,
table),
pos),
c))),
s->lr1 (*s*, *l*, *table*),
dv (s-pos (*s*), *n*)),
s-err-flag (s-eval (**t**, s-set-pos (*s*, *pos*), *c*)))

THEOREM: lr-s-similar-statesp-lr-pop-tstk
lr-s-similar-statesp (*s-params*, *s-temps*, lr-pop-tstk (*l*), *table*)
= lr-s-similar-statesp (*s-params*, *s-temps*, *l*, *table*)

THEOREM: listp-lr-expr-s->lr1
good-posp1 (s-pos (*s*), s-body (s-prog (*s*)))
→ (listp (lr-expr (s->lr1 (*s*, *l*, *table*))) = listp (s-expr (*s*)))

THEOREM: litatom-lr-expr-s->lr1-s-expr
good-posp1 (s-pos (*s*), s-body (s-prog (*s*)))
→ (litatom (lr-expr (s->lr1 (*s*, *l*, *table*))) = litatom (s-expr (*s*)))

THEOREM: car-lr-expr-s->lr1
good-posp1 (s-pos (s), s-body (s-prog (s)))
$\rightarrow$ (car (lr-expr (s->lr1 (s, l, *table*))) = car (s-expr (s)))

THEOREM: equal-p-psw-lr-eval-run-lr-eval-lr-set-error
(p-psw (l) = `'run`)
$\rightarrow$ (lr-eval (*flag*, lr-set-error (l, `'run`), c) = lr-eval (*flag*, l, c))

THEOREM: lr-proper-heapp-lr-good-pointerp-lr-proper-heapp-nodep
(lr-good-pointerp (*addr*, *data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ lr-proper-heapp (*data-seg*))
$\rightarrow$ lr-proper-heapp-nodep (*addr*, *data-seg*)

THEOREM: lr-check-result-f-not-lr-f-addr
((car (*temp-stk*) $\neq$ LR-F-ADDR)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ listp (*temp-stk*))
$\rightarrow$ (lr-check-result (**t**, **f**, *temp-stk*, *data-seg*, *orig-temp-stk*) = **f**)

THEOREM: lr-check-result-t-chain
((*flag* $\neq$ `'list`)
$\land$ lr-check-result (**t**, *ans*, *temp-stk2*, *data-seg2*, cdr (*temp-stk1*))
$\land$ lr-check-result (**t**, *anything*, *temp-stk1*, *data-seg1*, *temp-stk0*))
$\rightarrow$ lr-check-result (*flag*, *ans*, *temp-stk2*, *data-seg2*, *temp-stk0*)

THEOREM: lr-check-result-not-f-lr-f-addr
((car (*temp-stk*) = LR-F-ADDR)
$\land$ listp (*temp-stk*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ (*ans* $\neq$ **f**))
$\rightarrow$ (lr-check-result (**t**, *ans*, *temp-stk*, *data-seg*, *orig-temp-stk*) = **f**)

THEOREM: lr-eval-leaves-listp-p-temp-stk
(proper-p-statep (lr->p (l))
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ (*flag* $\neq$ `'list`)
$\land$ lr-programs-properp (l, *table*)
$\land$ (p-psw (lr-eval (*flag*, l, c)) = `'run`))
$\rightarrow$ listp (p-temp-stk (lr-eval (*flag*, l, c)))

THEOREM: lr-eval-s->lr1-if-opener-1
let *lr-test* **be** lr-if-ok (lr-eval (**t**,
lr-set-pos (s->lr1 (s, l, *table*),
dv (s-pos (s), `1`)),

c))
in
(good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧ listp (s-expr (s))
 ∧ (car (s-expr (s)) = `'if`)
 ∧ (*flag* ≠ `'list`)
 ∧ (p-psw (*lr-test*) = `'run`)
 ∧ (top (p-temp-stk (*lr-test*)) ≠ LR-F-ADDR))
→ (lr-eval (*flag*, s->lr1 (s, l, *table*), c)
 = lr-eval (**t**,
 lr-set-expr (lr-pop-tstk (*lr-test*),
 s->lr1 (s, l, *table*),
 dv (s-pos (s), `2`)),
 c)) **endlet**

THEOREM: lr-eval-s->lr1-if-opener-2
let *lr-test* **be** lr-if-ok (lr-eval (**t**,
 lr-set-pos (s->lr1 (s, l, *table*),
 dv (s-pos (s), `1`)),
 c))
in
(good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧ listp (s-expr (s))
 ∧ (car (s-expr (s)) = `'if`)
 ∧ (*flag* ≠ `'list`)
 ∧ (p-psw (*lr-test*) = `'run`)
 ∧ (top (p-temp-stk (*lr-test*)) = LR-F-ADDR))
→ (lr-eval (*flag*, s->lr1 (s, l, *table*), c)
 = lr-eval (**t**,
 lr-set-expr (lr-pop-tstk (*lr-test*),
 s->lr1 (s, l, *table*),
 dv (s-pos (s), `3`)),
 c)) **endlet**

THEOREM: lr-eval-s->lr1-if-opener-3
(good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧ (*flag* ≠ `'list`)
 ∧ listp (s-expr (s))
 ∧ (car (s-expr (s)) = `'if`)
 ∧ ($c \not\simeq$ `0`)
 ∧ s-good-statep (s, c)
 ∧ (p-psw (lr-if-ok (lr-eval (**t**,
 lr-set-pos (s->lr1 (s, l, *table*), dv (s-pos (s), `1`)),
 c)))

```
       ≠   'run))
 →    (lr-eval (flag, s->lr1 (s, l, table), c)
       =   lr-if-ok (lr-eval (t,
                              lr-set-pos (s->lr1 (s, l, table), dv (s-pos (s), 1)),
                              c)))
```

THEOREM: lr-eval-s->lr1-temp-eval-opener

```
 (good-posp1 (s-pos (s), s-body (s-prog (s)))
  ∧    (car (s-expr (s)) = S-TEMP-EVAL)
  ∧    (flag ≠ 'list)
  ∧    (c ≄ 0)
  ∧    listp (s-expr (s))
  ∧    s-good-statep (s, c))
 →    (lr-eval (flag, s->lr1 (s, l, table), c)
       =   lr-set-temp (lr-eval (t,
                                 lr-set-pos (s->lr1 (s, l, table),
                                             dv (s-pos (s), 1)),
                                 c),
                        top (p-temp-stk (lr-eval (t,
                                                  lr-set-pos (s->lr1 (s,
                                                                      l,
                                                                      table),
                                                              dv (s-pos (s), 1)),
                                                  c))),
                        caddr (lr-expr (s->lr1 (s, l, table)))))
```

THEOREM: lr-eval-s->lr1-temp-test-opener

```
 (good-posp1 (s-pos (s), s-body (s-prog (s)))
  ∧    (car (s-expr (s)) = S-TEMP-TEST)
  ∧    (flag ≠ 'list)
  ∧    (c ≄ 0)
  ∧    listp (s-expr (s))
  ∧    s-good-statep (s, c))
 →    (lr-eval (flag, s->lr1 (s, l, table), c)
       =   if p-max-temp-stk-size (l) ≮ (2 + length (p-temp-stk (l)))
           then if lr-eval-temp-setp (s->lr1 (s, l, table))
                then lr-do-temp-fetch (s->lr1 (s, l, table))
                else lr-set-temp (lr-eval (t,
                                           lr-set-pos (s->lr1 (s,
                                                               l,
                                                               table),
                                                       dv (s-pos (s), 1)),
                                           c),
                                  top (p-temp-stk (lr-eval (t,
```

```
                                        lr-set-pos (s->lr1 (s,
                                                           l,
                                                           table),
                                                    dv (s-pos (s),
                                                        1)),
                                        c))),
                            caddr (lr-expr (s->lr1 (s, l, table)))) endif
         else lr-set-error (s->lr1 (s, l, table),
                            'lr-temp-setp-temp-stack-overflow) endif)
```

Theorem: lr-eval-s->lr1-temp-fetch-opener

```
(good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧  (car (s-expr (s)) = S-TEMP-FETCH)
 ∧  (flag ≠ 'list)
 ∧  (c ≄ 0)
 ∧  listp (s-expr (s))
 ∧  s-good-statep (s, c))
→   (lr-eval (flag, s->lr1 (s, l, table), c)
      =  lr-do-temp-fetch (s->lr1 (s, l, table)))
```

Theorem: lr-eval-s->lr1-quote-opener

```
(good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧  (car (s-expr (s)) = 'quote)
 ∧  (flag ≠ 'list)
 ∧  (c ≄ 0)
 ∧  listp (s-expr (s))
 ∧  s-good-statep (s, c))
→   (lr-eval (flag, s->lr1 (s, l, table), c)
      =  lr-push-tstk (s->lr1 (s, l, table),
                       cadr (lr-expr (s->lr1 (s, l, table)))))
```

Theorem: lr-params-lr-set-temp

```
lr-params (frame, lr-set-temp (l, value, var-name)) = lr-params (frame, l)
```

Theorem: lr-temps-lr-set-temp

```
lr-temps (frame, lr-set-temp (l, value, var-name)) = lr-temps (frame, l)
```

Theorem: firstn-put-assoc

```
firstn (n, put-assoc (val, name, alist)) = put-assoc (val, name, firstn (n, alist))
```

Theorem: strip-cars-nil-fact

```
(nil = strip-cars (y)) = (¬ listp (y))
```

Definition:

```
induct-hint-13 (e, x, y)
```

$=$ **if** listp (x)
then if listp (y)
then if $e = \text{caar}(x)$ **then t**
else induct-hint-13 $(e,\ \text{cdr}(x),\ \text{cdr}(y))$ **endif**
else t endif
else t endif

THEOREM: strip-cars-equal-definedp-equal
$(\text{strip-cars}(x) = \text{strip-cars}(y)) \rightarrow (\text{definedp}(e,\ x) = \text{definedp}(e,\ y))$

THEOREM: lr-eval-preserves-definedp-firstn-bindings-car-p-ctrl-stk
(proper-p-statep (lr->p (l))
$\wedge$ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l))))
$\wedge$ lr-programs-properp $(l,\ table)$
$\wedge$ (p-psw (lr-eval (*flag*, l, c)) = **'run**))
$\rightarrow$ (definedp (x, firstn (n, bindings (car (p-ctrl-stk (lr-eval (*flag*, l, c))))))
$=$ definedp (x, firstn (n, bindings (car (p-ctrl-stk (l)))))))

DEFINITION:
disjointp (*list1*, *list2*)
$=$ **if** listp (*list1*)
then (car (*list1*) $\notin$ *list2*) $\wedge$ disjointp (cdr (*list1*), *list2*)
else t endif

THEOREM: member-disjointp-non-member-1
$(\text{disjointp}(x,\ y) \wedge (e \in x)) \rightarrow (e \notin y)$

THEOREM: lr-eval-preserves-definedp-fn-bindings-car-ctrl-stk-set-pos
(proper-p-statep (lr->p (l))
$\wedge$ good-posp1 (*pos*, program-body (p-current-program (l))))
$\wedge$ lr-programs-properp $(l,\ table)$
$\wedge$ (p-psw (lr-eval (**t**, lr-set-pos (l, *pos*), c)) = **'run**))
$\rightarrow$ (definedp (x,
firstn (n,
bindings (car (p-ctrl-stk (lr-eval (**t**,
lr-set-pos (l, *pos*),
c))))))
$=$ definedp (x, firstn (n, bindings (car (p-ctrl-stk (l)))))))

THEOREM: lr-params-p-frame-not-definedp-put-assoc-anything
(proper-p-statep (lr->p (l))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l))))
$\wedge$ good-posp1 (*pos*, program-body (p-current-program (l))))
$\wedge$ lr-programs-properp $(l,\ table)$
$\wedge$ disjointp (formal-vars (p-current-program (l)),

```
                  strip-cars (temp-var-dcls (p-current-program (l))))
 ∧   listp (lr-expr (l))
 ∧   ((car (lr-expr (l)) = S-TEMP-EVAL)
      ∨   (car (lr-expr (l)) = S-TEMP-TEST))
 ∧   (p-psw (lr-eval (t, lr-set-pos (l, pos), c)) = 'run))
→   (lr-params (p-frame (put-assoc (anything,
                                    caddr (lr-expr (l)),
                                    bindings (car (p-ctrl-stk (lr-eval (t,
                                                                        lr-set-pos (l,
                                                                                    pos),
                                                                        c))))),
                          ret-pc),
                   l)
      =   lr-params (car (p-ctrl-stk (lr-eval (t, lr-set-pos (l, pos), c))), l))
```

DEFINITION:

```
induct-hint-14 (s-temps, lr-temps, temp-alist)
=   if listp (s-temps)
    then if listp (lr-temps)
         then if listp (temp-alist)
              then if cdar (lr-temps) = LR-UNDEF-ADDR
                   then induct-hint-14 (cdr (s-temps),
                                        cdr (lr-temps),
                                        cdr (temp-alist))
                   else induct-hint-14 (cdr (s-temps),
                                       cdr (lr-temps),
                                       cdr (temp-alist)) endif
              else t endif
         else t endif
    else t endif
```

THEOREM: put-assoc-opener-1

```
((name ≠ caar (alist)) ∧ listp (alist))
→   (put-assoc (val, name, alist)
      =   cons (car (alist), put-assoc (val, name, cdr (alist))))
```

THEOREM: put-assoc-opener-2

```
(listp (alist3)
 ∧   (caar (alist3) ∉ strip-cars (cdr (alist3)))
 ∧   definedp (s-expr, alist1)
 ∧   (strip-cars (alist1) = strip-cars (alist2))
 ∧   (strip-cdrs (alist2) = strip-cars (cdr (alist3))))
→   (put-assoc (val, cdr (assoc (s-expr, alist2)), alist3)
      =   cons (car (alist3),
                put-assoc (val, cdr (assoc (s-expr, alist2)), cdr (alist3))))
```

THEOREM: not-lr-valp-lr-undef-addr
(lr-proper-heapp (*data-seg*) $\wedge$ lr-proper-p-areasp (*data-seg*))
$\rightarrow$ ($\neg$ lr-valp (*value*, identity (LR-UNDEF-ADDR), *data-seg*))

THEOREM: lr-s-similar-temps-put-assoc-put-assoc-helper-1
(listp (*s-temps*)
$\wedge$ listp (*lr-temps*)
$\wedge$ lr-s-similar-temps (*s-temps*, *lr-temps*, *data-seg*)
$\wedge$ lr-valp (*value*, *addr*, *data-seg*)
$\wedge$ lr-proper-heapp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ (*name1* = caar (*s-temps*))
$\wedge$ (*name2* = caar (*lr-temps*)))
$\rightarrow$ lr-s-similar-temps (put-assoc (list (**t**, *value*), *name1*, *s-temps*),
put-assoc (*addr*, *name2*, *lr-temps*),
data-seg)

THEOREM: lr-s-similar-temps-put-assoc-put-assoc-helper
(lr-s-similar-temps (*s-temps*, *lr-temps*, *data-seg*)
$\wedge$ lr-valp (*value*, *addr*, *data-seg*)
$\wedge$ lr-proper-heapp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ (strip-cars (*temp-alist*) = strip-cars (*s-temps*))
$\wedge$ (strip-cdrs (*temp-alist*) = strip-cars (*lr-temps*))
$\wedge$ no-duplicatesp (strip-cars (*lr-temps*))
$\wedge$ definedp (*s-expr*, *s-temps*))
$\rightarrow$ lr-s-similar-temps (put-assoc (list (**t**, *value*), *s-expr*, *s-temps*),
put-assoc (*addr*,
cdr (assoc (*s-expr*, *temp-alist*)),
lr-temps),
data-seg)

THEOREM: disjointp-cons-arg2
(disjointp (*list1*, *list2*) $\wedge$ ($x \notin$ *list1*))
$\rightarrow$ disjointp (*list1*, cons (x, *list2*))

THEOREM: disjointp-nlistp-arg2
(*list2* $\simeq$ **nil**) $\rightarrow$ disjointp (*list1*, *list2*)

THEOREM: disjointp-lr-make-temp-name-alist-1
disjointp (*formals*,
strip-cdrs (lr-make-temp-name-alist-1 (*initial*,
num-list,
temp-list,
formals)))

THEOREM: lr-s-similar-statesp-s-change-temp-helper-2

$$
\begin{array}{l}
\textbf{let } \textit{lr-eval} \textbf{ be } \text{lr-eval}(\mathbf{t}, \text{lr-set-pos}(\text{s->lr1}(s, l, \textit{table}), \textit{pos}), c) \\
\textbf{in} \\
(\text{proper-p-statep}(\text{lr->p}(\text{s->lr1}(s, l, \textit{table}))) \\
\ \wedge \quad \text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s))) \\
\ \wedge \quad \text{good-posp1}(\textit{pos}, \text{s-body}(\text{s-prog}(s))) \\
\ \wedge \quad \text{lr-s-similar-statesp}(\textit{s-params}, \text{s-temps}(\textit{s-eval}), \textit{lr-eval}, \textit{table}) \\
\ \wedge \quad \text{lr-programs-properp}(\text{s->lr1}(s, l, \textit{table}), \textit{table}) \\
\ \wedge \quad \text{s-good-statep}(s, c) \\
\ \wedge \quad \text{listp}(\text{s-expr}(s)) \\
\ \wedge \quad ((\text{car}(\text{s-expr}(s)) = \text{S-TEMP-EVAL}) \\
\qquad \vee \quad (\text{car}(\text{s-expr}(s)) = \text{S-TEMP-TEST})) \\
\ \wedge \quad (\text{p-psw}(\textit{lr-eval}) = \texttt{'run}) \\
\ \wedge \quad (\textit{lr-expr} = \text{caddr}(\text{lr-expr}(\text{s->lr1}(s, l, \textit{table}))))) \\
\rightarrow \quad (\text{lr-s-similar-statesp}(\textit{s-params}, \\
\qquad\qquad \text{s-temps}(\text{s-change-temp}(\textit{s-eval}, \\
\qquad\qquad\qquad \textit{s-expr}, \\
\qquad\qquad\qquad \textit{value})), \\
\qquad\qquad \text{lr-set-temp}(\textit{lr-eval}, \textit{addr}, \textit{lr-expr}), \\
\qquad\qquad \textit{table}) \\
\qquad = \quad \text{lr-s-similar-temps}(\text{put-assoc}(\text{list}(\mathbf{t}, \textit{value}), \\
\qquad\qquad\qquad \textit{s-expr}, \\
\qquad\qquad\qquad \text{s-temps}(\textit{s-eval})), \\
\qquad\qquad \text{lr-temps}(\text{p-frame}(\text{put-assoc}(\textit{addr}, \\
\qquad\qquad\qquad\qquad \textit{lr-expr}, \\
\qquad\qquad\qquad\qquad \text{bindings}(\text{car}(\text{p-ctrl-stk}(\textit{lr-eval})))), \\
\qquad\qquad\qquad \text{ret-pc}(\text{car}(\text{p-ctrl-stk}(\textit{lr-eval})))), \\
\qquad\qquad\qquad \text{s->lr1}(s, l, \textit{table})), \\
\qquad\qquad \text{p-data-segment}(\textit{lr-eval}))) \textbf{ endlet}
\end{array}
$$

THEOREM: good-posp1-dv-1-temps-lr-expr

$$
\begin{array}{l}
(((\text{car}(\text{s-expr}(s)) = \text{S-TEMP-EVAL}) \vee (\text{car}(\text{s-expr}(s)) = \text{S-TEMP-TEST})) \\
\ \wedge \quad \text{listp}(\text{s-expr}(s)) \\
\ \wedge \quad \text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{assoc}(\text{s-pname}(s), \text{s-progs}(s))))) \\
\rightarrow \quad \text{good-posp1}(\text{dv}(\text{s-pos}(s), \texttt{1}), \text{s-body}(\text{assoc}(\text{s-pname}(s), \text{s-progs}(s))))
\end{array}
$$

THEOREM: put-assoc-restn

$$
\begin{array}{l}
(\neg \text{ definedp}(\textit{name}, \text{firstn}(n, \textit{alist}))) \\
\rightarrow \quad (\text{put-assoc}(\textit{val}, \textit{name}, \text{restn}(n, \textit{alist})) \\
\qquad = \quad \text{restn}(n, \text{put-assoc}(\textit{val}, \textit{name}, \textit{alist})))
\end{array}
$$

THEOREM: disjointp-plist-arg-2

$$\text{disjointp}(x, \text{plist}(y)) = \text{disjointp}(x, y)$$

THEOREM: not-disjointp-member-arg1-cons-arg2

$$(v \in y) \rightarrow (\neg \text{ disjointp}(y, \text{cons}(v, z)))$$

THEOREM: member-disjointp-cons-arg2
$(v \notin y) \rightarrow (\text{disjointp}\,(y,\ \text{cons}\,(v,\ z)) = \text{disjointp}\,(y,\ z))$

THEOREM: disjointp-commutative
$\text{disjointp}\,(x,\ y) = \text{disjointp}\,(y,\ x)$

THEOREM: disjointp-lr-make-temp-name-alist-2
$$\begin{aligned}&\text{disjointp (strip-cdrs (lr-make-temp-name-alist-1}\,(\textit{initial}, \\ &\qquad\qquad\qquad\qquad \textit{num-list}, \\ &\qquad\qquad\qquad\qquad \textit{temp-list}, \\ &\qquad\qquad\qquad\qquad \textit{formals})), \\ &\qquad \textit{formals})\end{aligned}$$

THEOREM: proper-p-statep-lr->p-s->lr1-strip-cars-bindings-ctrl-stk
$$\begin{aligned}&(\text{proper-p-statep (lr->p (s->lr1}\,(s,\ l,\ \textit{table})))\\ &\ \wedge\ \ \text{definedp (s-pname}\,(s),\ \text{s-progs}\,(s)))\\ &\rightarrow\ (\text{strip-cars (bindings (car (p-ctrl-stk}\,(l))))\\ &\qquad = \ \text{append (s-formals (assoc (s-pname}\,(s),\ \text{s-progs}\,(s))),\\ &\qquad\qquad \text{strip-cdrs (lr-make-temp-name-alist (s-temp-list (assoc (s-pname}\,(s),\\ &\qquad\qquad\qquad\qquad \text{s-progs}\,(s))),\\ &\qquad\qquad\qquad \text{s-formals (assoc (s-pname}\,(s),\\ &\qquad\qquad\qquad\qquad \text{s-progs}\,(s))))))))\end{aligned}$$

THEOREM: lr-programs-properp-lr->p-s->lr1-definedp-s-pname
$$\begin{aligned}&(\neg\ \text{definedp (s-pname}\,(s),\ \text{s-progs}\,(s)))\\ &\rightarrow\ (\neg\ \text{lr-programs-properp (s->lr1}\,(s,\ l,\ \textit{table}),\ \textit{table}))\end{aligned}$$

THEOREM: lr-temps-p-frame-put-assoc
$$\begin{aligned}&(\text{proper-p-statep (lr->p (s->lr1}\,(s,\ l,\ \textit{table})))\\ &\ \wedge\ \ \text{lr-programs-properp (s->lr1}\,(s,\ l,\ \textit{table}),\ \textit{table})\\ &\ \wedge\ \ \text{definedp (s-pname}\,(s),\ \text{s-progs}\,(s))\\ &\ \wedge\ \ \text{listp (s-expr}\,(s))\\ &\ \wedge\ \ ((\text{car (s-expr}\,(s)) = \text{S-TEMP-EVAL})\\ &\qquad \vee\ \ (\text{car (s-expr}\,(s)) = \text{S-TEMP-TEST}))\\ &\ \wedge\ \ \text{good-posp1 (s-pos}\,(s),\ \text{s-body (s-prog}\,(s)))\\ &\ \wedge\ \ (\text{p-psw (lr-eval}\,(\mathbf{t},\ l2,\ c)) = \texttt{'run})\\ &\ \wedge\ \ (\textit{lr-expr} = \text{caddr (lr-expr (s->lr1}\,(s,\ l,\ \textit{table}))))\\ &\ \wedge\ \ (l2 = \text{lr-set-pos (s->lr1}\,(s,\ l,\ \textit{table}),\ \text{dv (s-pos}\,(s),\ \texttt{1})))\\ &\rightarrow\ (\text{lr-temps (p-frame (put-assoc}\,(\textit{val},\\ &\qquad\qquad\qquad \textit{lr-expr},\\ &\qquad\qquad\qquad \text{bindings (car (p-ctrl-stk (lr-eval}\,(\mathbf{t},\ l2,\ c))))),\\ &\qquad\qquad \textit{ret-pc}),\\ &\qquad\quad \text{s->lr1}\,(s,\ l,\ \textit{table}))\\ &\qquad = \ \text{put-assoc}\,(\textit{val},\\ &\qquad\qquad \text{caddr (lr-expr (s->lr1}\,(s,\ l,\ \textit{table}))),\end{aligned}$$

lr-temps (car (p-ctrl-stk (lr-eval (**t**, *l2*, *c*))),
s->lr1 (*s*, *l*, *table*))))

THEOREM: strip-cars-lr-temps-strip-cars-temp-var-dcls
(s-good-statep (*s*, *c*)
∧ proper-p-statep (lr->p (s->lr1 (*s*, *l*, *table*)))
∧ (*frame* = top (p-ctrl-stk (s->lr1 (*s*, *l*, *table*)))))
→ (strip-cars (lr-temps (*frame*, s->lr1 (*s*, *l*, *table*)))
= strip-cdrs (lr-make-temp-name-alist (s-temp-list (assoc (s-pname (*s*),
s-progs (*s*))),
s-formals (assoc (s-pname (*s*),
s-progs (*s*))))))

EVENT: Disable proper-p-statep-lr->p-s->lr1-strip-cars-bindings-ctrl-stk.

THEOREM: lr-s-similar-statesp-s-change-temp-helper-1
(good-posp1 (s-pos (*s*), s-body (s-prog (*s*)))
∧ listp (s-expr (*s*))
∧ ((car (s-expr (*s*)) = S-TEMP-EVAL)
∨ (car (s-expr (*s*)) = S-TEMP-TEST)
∨ (car (s-expr (*s*)) = S-TEMP-FETCH)))
→ (caddr (lr-expr (s->lr1 (*s*, *l*, *table*)))
= cdr (assoc (cadr (s-expr (*s*)),
lr-make-temp-name-alist (s-temp-list (s-prog (*s*)),
s-formals (s-prog (*s*))))))

THEOREM: lr-s-similar-statesp-s->lr1-lr-similar-temps
lr-s-similar-statesp (*s-params*, *s-temps*, *l*, *table*)
→ lr-s-similar-temps (*s-temps*,
lr-temps (top (p-ctrl-stk (*l*)), *l*),
p-data-segment (*l*))

THEOREM: count-codelist1-cons
count-codelist1 (cons (*x*, *y*)) = (*x* + (10 ∗ count-codelist1 (*y*)))

THEOREM: equal-append-initial
(append (*x*, *y*) = append (*x*, *z*)) = (*y* = *z*)

THEOREM: plist-listp-x-append-x-not-0
plistp (*x*) → ((append (*x*, 0) = 0) = (*x* = **nil**))

THEOREM: equal-append-final-0
(append (*y*, 0) = append (*z*, 0)) = (plist (*y*) = plist (*z*))

THEOREM: count-codelist1-append-non-listp
$(\neg\ \text{listp}\,(z))$
$\rightarrow\quad (\text{count-codelist1}\,(\text{append}\,(\textit{num-list},\ z)) = \text{count-codelist1}\,(\textit{num-list}))$

THEOREM: not-equal-make-symbol-car-gensym
$(\text{count-codelist1}\,(\textit{num-list1}) < \text{count-codelist1}\,(\textit{num-list2}))$
$\rightarrow\quad (\text{make-symbol}\,(\textit{initial},\ \textit{num-list1})$
$\qquad \neq\quad \text{car}\,(\text{gensym}\,(\textit{initial},\ \textit{num-list2},\ \textit{atom-list})))$

THEOREM: count-codelist1-cdr-gensym
$(\text{count-codelist1}\,(\textit{num-list1}) < \text{count-codelist1}\,(\textit{num-list2}))$
$\rightarrow\quad (\text{count-codelist1}\,(\textit{num-list1})$
$\qquad <\quad \text{count-codelist1}\,(\text{cdr}\,(\text{gensym}\,(\textit{initial},\ \textit{num-list2},\ \textit{atom-list}))))$

THEOREM: not-member-make-symbol-lr-make-temp-name-alist-1-incr
$(\text{count-codelist1}\,(\textit{num-list1}) < \text{count-codelist1}\,(\textit{num-list2}))$
$\rightarrow\quad ((\text{make-symbol}\,(\textit{initial},\ \textit{num-list1})$
$\qquad \in\quad \text{strip-cdrs}\,(\text{lr-make-temp-name-alist-1}\,(\textit{initial},\ \textit{num-list2},\ \textit{temp-list},\ \textit{formals})))$
$\qquad =\quad \mathbf{f})$

THEOREM: not-member-car-gensym-lr-make-temp-name-alist-1-cdr
$(\text{car}\,(\text{gensym}\,(\textit{initial},\ \textit{num-list},\ \textit{atoms}))$
$\ \in\quad \text{strip-cdrs}\,(\text{lr-make-temp-name-alist-1}\,(\textit{initial},\ \text{cdr}\,(\text{gensym}\,(\textit{initial},\ \textit{num-list},\ \textit{atoms})),\ \textit{temp-list},\ \textit{formals})))$
$=\quad \mathbf{f}$

THEOREM: no-duplicatesp-strip-cdrs-lr-make-temp-name-alist-1
$\text{no-duplicatesp}\,(\text{strip-cdrs}\,(\text{lr-make-temp-name-alist-1}\,(\textit{initial},\ \textit{num-list},\ \textit{temp-list},\ \textit{formals})))$

THEOREM: no-duplicatesp-strip-cdrs-lr-make-temp-name-alist
$\text{no-duplicatesp}\,(\text{strip-cdrs}\,(\text{lr-make-temp-name-alist}\,(\textit{temp-list},\ \textit{formals})))$

THEOREM: definedp-s-temps-s-eval
$(\text{s-err-flag}\,(\text{s-eval}\,(\textit{flag},\ s,\ c)) = \texttt{'run})$
$\rightarrow\quad (\text{definedp}\,(x,\ \text{s-temps}\,(\text{s-eval}\,(\textit{flag},\ s,\ c))) = \text{definedp}\,(x,\ \text{s-temps}\,(s)))$

THEOREM: strip-cars-lr-make-temp-name-alist-1

$$
\begin{array}{l}
\text{strip-cars}\,(\text{lr-make-temp-name-alist-1}\,(\mathit{initial},\ \mathit{num\text{-}list},\ \mathit{temp\text{-}list},\ \mathit{formals})) \\
= \quad \text{plist}\,(\mathit{temp\text{-}list})
\end{array}
$$

THEOREM: strip-cars-lr-make-temp-name-alist

$$
\text{strip-cars}\,(\text{lr-make-temp-name-alist}\,(\mathit{temp\text{-}list},\ \mathit{formals})) = \text{plist}\,(\mathit{temp\text{-}list})
$$

THEOREM: lr-eval-preserves-strip-cars-lr-temps-car-p-ctrl-stk

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{proper-p-statep}\,(\text{lr->p}\,(l))} \\
\land & \text{good-posp}\,(\mathit{flag},\ \text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
\land & \text{lr-programs-properp}\,(l,\ \mathit{table}) \\
\land & (\text{p-psw}\,(\text{lr-eval}\,(\mathit{flag},\ l,\ c)) = \texttt{'run})) \\
\to & (\text{strip-cars}\,(\text{lr-temps}\,(\text{car}\,(\text{p-ctrl-stk}\,(\text{lr-eval}\,(\mathit{flag},\ l,\ c))),\ \mathit{l2})) \\
 & \quad = \quad \text{strip-cars}\,(\text{lr-temps}\,(\text{car}\,(\text{p-ctrl-stk}\,(l)),\ \mathit{l2})))
\end{array}
$$

THEOREM: lr-s-similar-statesp-s-change-temp

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textbf{let}\ \mathit{s\text{-}l}\ \ \textbf{be}\ \ \text{s->lr1}\,(s,\ l,\ \mathit{table}),} \\
\multicolumn{2}{l}{\quad\ \mathit{lr\text{-}eval}\ \ \textbf{be}\ \ \text{lr-eval}\,(\mathbf{t},\ \text{lr-set-pos}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{pos}),\ c)} \\
\multicolumn{2}{l}{\textbf{in}} \\
\multicolumn{2}{l}{(\text{proper-p-statep}\,(\text{lr->p}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table})))} \\
\land & \text{lr-s-similar-statesp}\,(\mathit{s\text{-}params},\ \text{s-temps}\,(\mathit{s\text{-}eval}),\ \mathit{lr\text{-}eval},\ \mathit{table}) \\
\land & \text{good-posp1}\,(\text{s-pos}\,(s),\ \text{s-body}\,(\text{s-prog}\,(s))) \\
\land & \text{lr-programs-properp}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table}) \\
\land & \text{s-good-statep}\,(s,\ c) \\
\land & \text{listp}\,(\text{p-temp-stk}\,(\mathit{lr\text{-}eval})) \\
\land & \text{lr-check-result}\,(\mathbf{t}, \\
 & \qquad\qquad\qquad \text{s-ans}\,(\mathit{s\text{-}eval}), \\
 & \qquad\qquad\qquad \text{p-temp-stk}\,(\mathit{lr\text{-}eval}), \\
 & \qquad\qquad\qquad \text{p-data-segment}\,(\mathit{lr\text{-}eval}), \\
 & \qquad\qquad\qquad \mathit{orig\text{-}temp\text{-}stk}) \\
\land & \text{listp}\,(\text{s-expr}\,(s)) \\
\land & ((\text{car}\,(\text{s-expr}\,(s)) = \text{S-TEMP-EVAL}) \\
 & \ \lor \quad (\text{car}\,(\text{s-expr}\,(s)) = \text{S-TEMP-TEST})) \\
\land & (\text{p-psw}\,(\mathit{lr\text{-}eval}) = \texttt{'run}) \\
\land & (\text{s-err-flag}\,(\mathit{s\text{-}eval}) = \texttt{'run}) \\
\land & (\mathit{s\text{-}eval} = \text{s-eval}\,(\mathbf{t},\ \text{s-set-pos}\,(s,\ \mathit{pos}),\ c)) \\
\land & (\mathit{value} = \text{caddr}\,(\text{lr-expr}\,(\mathit{s\text{-}l}))) \\
\land & (\mathit{pos} = \text{dv}\,(\text{s-pos}\,(s),\ \texttt{1}))) \\
\to & \text{lr-s-similar-statesp}\,(\mathit{s\text{-}params}, \\
 & \qquad\qquad \text{s-temps}\,(\text{s-change-temp}\,(\mathit{s\text{-}eval}, \\
 & \qquad\qquad\qquad\qquad \text{cadr}\,(\text{s-expr}\,(s)), \\
 & \qquad\qquad\qquad\qquad \text{s-ans}\,(\mathit{s\text{-}eval}))), \\
 & \qquad\qquad \text{lr-set-temp}\,(\mathit{lr\text{-}eval}, \\
 & \qquad\qquad\qquad \text{car}\,(\text{p-temp-stk}\,(\mathit{lr\text{-}eval})), \\
 & \qquad\qquad\qquad \mathit{value}),
\end{array}
$$

table) **endlet**

EVENT: Disable lr-s-similar-statesp-s-change-temp-helper-2.

THEOREM: lr-temps-lr-do-temp-fetch
lr-temps (*frame*, lr-do-temp-fetch (l)) = lr-temps (*frame*, l)

THEOREM: lr-params-lr-do-temp-fetch
lr-params (*frame*, lr-do-temp-fetch (l)) = lr-params (*frame*, l)

THEOREM: lr-s-simlar-statesp-lr-do-temp-fetch
lr-s-similar-statesp (*s-params*, *s-temps*, lr-do-temp-fetch (l), *table*)
= lr-s-similar-statesp (*s-params*, *s-temps*, l, *table*)

THEOREM: not-member-no-duplicates-cdr-assoc-helper
(no-duplicatesp (*list*)
∧ (strip-cdrs (*alist*) = *list*)
∧ (*name* ∉ *list*)
∧ definedp (*s-expr*, *alist*))
→ (cdr (assoc (*s-expr*, *alist*)) ≠ *name*)

THEOREM: not-member-no-duplicates-cdr-assoc
(no-duplicatesp (*list*)
∧ (strip-cdrs (*alist1*) = *list*)
∧ (*name* ∉ *list*)
∧ definedp (*s-expr*, *alist2*)
∧ (strip-cars (*alist2*) = strip-cars (*alist1*)))
→ (cdr (assoc (*s-expr*, *alist1*)) ≠ *name*)

THEOREM: not-equal-lr-s-eval-temp-setp-not-lr-s-similar-temps
((*lr-expr* = cdr (assoc (*s-expr*, *temp-alist*)))
∧ lr-proper-heapp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*)
∧ lr-s-similar-temps (*s-temps*, *lr-temps*, *data-seg*)
∧ (strip-cars (*temp-alist*) = strip-cars (*s-temps*))
∧ (strip-cdrs (*temp-alist*) = strip-cars (*lr-temps*))
∧ no-duplicatesp (strip-cars (*lr-temps*))
∧ definedp (*s-expr*, *s-temps*))
→ ((cdr (assoc (*lr-expr*, *lr-temps*)) ≠ LR-UNDEF-ADDR)
↔ cadr (assoc (*s-expr*, *s-temps*)))

THEOREM: definedp-strip-cars-append-member-x-2
(strip-cars (x) = append (y, z))
→ (($e \in z$) = definedp (e, restn (length (y), x)))

THEOREM: not-iff-lr-s-temp-setp-not-lr-s-similar-statesp-helper

$$((x \in \text{strip-cdrs}\,(\text{lr-make-temp-name-alist}\,(\text{s-temp-list}\,(\text{assoc}\,(\text{s-pname}\,(s), \text{s-progs}\,(s))), \text{s-formals}\,(\text{assoc}\,(\text{s-pname}\,(s), \text{s-progs}\,(s))))))$$
$$\land\ \text{proper-p-statep}\,(\text{lr->p}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table})))$$
$$\land\ \text{definedp}\,(\text{s-pname}\,(s),\ \text{s-progs}\,(s)))$$
$$\to\ (\text{cdr}\,(\text{assoc}\,(x,\ \text{lr-temps}\,(\text{car}\,(\text{p-ctrl-stk}\,(l)),\ \text{s->lr1}\,(s,\ l,\ \mathit{table}))))$$
$$=\ \text{cdr}\,(\text{assoc}\,(x,\ \text{bindings}\,(\text{car}\,(\text{p-ctrl-stk}\,(l))))))$$

THEOREM: lr-programs-properp-member-lr-expr-temps

$$(\text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l)))$$
$$\land\ \text{lr-programs-properp}\,(l,\ \mathit{table})$$
$$\land\ ((\text{car}\,(\text{lr-expr}\,(l)) = \text{S-TEMP-FETCH})$$
$$\lor\ (\text{car}\,(\text{lr-expr}\,(l)) = \text{S-TEMP-EVAL})$$
$$\lor\ (\text{car}\,(\text{lr-expr}\,(l)) = \text{S-TEMP-TEST})))$$
$$\to\ (\text{caddr}\,(\text{lr-expr}\,(l))$$
$$\in\ \text{strip-cars}\,(\text{temp-var-dcls}\,(\text{p-current-program}\,(l))))$$

THEOREM: not-iff-lr-s-temp-setp-not-lr-s-similar-statesp

$$(\text{proper-p-statep}\,(\text{lr->p}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table})))$$
$$\land\ \text{lr-proper-heapp}\,(\text{p-data-segment}\,(l))$$
$$\land\ \text{good-posp1}\,(\text{s-pos}\,(s),\ \text{s-body}\,(\text{s-prog}\,(s)))$$
$$\land\ \text{lr-programs-properp}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table})$$
$$\land\ \text{listp}\,(\text{s-expr}\,(s))$$
$$\land\ ((\text{car}\,(\text{s-expr}\,(s)) = \text{S-TEMP-TEST})$$
$$\lor\ (\text{car}\,(\text{s-expr}\,(s)) = \text{S-TEMP-FETCH}))$$
$$\land\ (\neg\ (\text{lr-eval-temp-setp}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}))$$
$$\leftrightarrow\ \text{s-temp-setp}\,(\text{cadr}\,(\text{s-expr}\,(s)),\ \text{s-temps}\,(s))))$$
$$\land\ \text{s-good-statep}\,(s,\ c))$$
$$\to\ (\neg\ \text{lr-s-similar-statesp}\,(\mathit{s\text{-}params},\ \text{s-temps}\,(s),\ \text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table}))$$

THEOREM: lr-valp-lr-s-eval-lr-s-similar-temps

$$((\mathit{lr\text{-}expr} = \text{cdr}\,(\text{assoc}\,(\mathit{s\text{-}expr},\ \mathit{temp\text{-}alist})))$$
$$\land\ \text{lr-proper-heapp}\,(\mathit{data\text{-}seg})$$
$$\land\ \text{lr-proper-p-areasp}\,(\mathit{data\text{-}seg})$$
$$\land\ \text{lr-s-similar-temps}\,(\mathit{s\text{-}temps},\ \mathit{lr\text{-}temps},\ \mathit{data\text{-}seg})$$
$$\land\ (\text{strip-cars}\,(\mathit{temp\text{-}alist}) = \text{strip-cars}\,(\mathit{s\text{-}temps}))$$
$$\land\ (\text{strip-cdrs}\,(\mathit{temp\text{-}alist}) = \text{strip-cars}\,(\mathit{lr\text{-}temps}))$$
$$\land\ \text{no-duplicatesp}\,(\text{strip-cars}\,(\mathit{lr\text{-}temps}))$$
$$\land\ \text{definedp}\,(\mathit{s\text{-}expr},\ \mathit{s\text{-}temps})$$
$$\land\ (\text{cdr}\,(\text{assoc}\,(\mathit{lr\text{-}expr},\ \mathit{lr\text{-}temps})) \neq \text{LR-UNDEF-ADDR}))$$
$$\to\ \text{lr-valp}\,(\text{caddr}\,(\text{assoc}\,(\mathit{s\text{-}expr},\ \mathit{s\text{-}temps})),\ \text{cdr}\,(\text{assoc}\,(\mathit{lr\text{-}expr},\ \mathit{lr\text{-}temps})),\ \mathit{data\text{-}seg})$$

THEOREM: member-cdr-assoc-strip-cdrs-definedp
definedp $(x,\ alist) \rightarrow$ (cdr (assoc $(x,\ alist)) \in$ strip-cdrs $(alist)$)

THEOREM: definedp-pairlist
definedp $(x,$ pairlist $(temp\text{-}list,\ anything)) = (x \in temp\text{-}list)$

THEOREM: definedp-lr-make-temp-name-alist-1
definedp $(x,$ lr-make-temp-name-alist-1 $(initial,\ num\text{-}list,\ temp\text{-}list,\ formals))$
$= \quad (x \in temp\text{-}list)$

THEOREM: definedp-lr-make-temp-name-alist
definedp $(x,$ lr-make-temp-name-alist $(temp\text{-}list,\ formals))$
$= \quad (x \in temp\text{-}list)$

THEOREM: p-temp-stk-lr-do-temp-fetch-p-psw-run
(p-psw (lr-do-temp-fetch (l)) = `'run`)
$\rightarrow$ (p-temp-stk (lr-do-temp-fetch (l))
 $=$ push (local-var-value (caddr (lr-expr (l)), p-ctrl-stk (l)),
 p-temp-stk (l))))

THEOREM: lr-check-result-lr-do-temp-fetch
(proper-p-statep (lr->p (s->lr1 $(s,\ l,\ table)$)))
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s))))
$\wedge$ s-good-statep $(s,\ c)$
$\wedge$ $(c \not\simeq 0)$
$\wedge$ listp (s-expr (s))
$\wedge$ ((car (s-expr (s)) = S-TEMP-TEST)
 $\vee$ (car (s-expr (s)) = S-TEMP-FETCH))
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-s-similar-statesp (s-params (s), s-temps (s), s->lr1 $(s,\ l,\ table)$, $table$)
$\wedge$ lr-eval-temp-setp (s->lr1 $(s,\ l,\ table)$)
$\wedge$ ($value$ = caddr (lr-expr (s->lr1 $(s,\ l,\ table)$))))
$\rightarrow$ lr-check-result (**t**,
 caddr (assoc (cadr (s-expr (s)), s-temps (s))),
 cons (cdr (assoc ($value$, bindings (car (p-ctrl-stk (l)))))),
 p-temp-stk (l)),
 p-data-segment (l),
 p-temp-stk (l))

THEOREM: lr-do-temp-fetch-run-lr-eval-temp-setp
(p-psw (lr-do-temp-fetch (l)) = `'run`) $\rightarrow$ lr-eval-temp-setp (l)

THEOREM: lr-s-similar-const-table-lr-valp-assoc
(definedp $(value,\ table) \wedge$ lr-s-similar-const-table $(table,\ data\text{-}seg)$)
$\rightarrow$ lr-valp ($value$, cdr (assoc $(value,\ table)$), $data\text{-}seg$)

THEOREM: lr-proper-exprp-list-quote-opener

$$
\begin{array}{l}
(\mathit{flag} \neq \texttt{'list}) \\
\rightarrow \quad (\text{lr-proper-exprp}\,(\mathit{flag}, \\
\qquad\qquad\qquad \text{list}\,(\texttt{'quote},\ \mathit{addr}), \\
\qquad\qquad\qquad \mathit{program\text{-}names}, \\
\qquad\qquad\qquad \mathit{formals}, \\
\qquad\qquad\qquad \mathit{temps}, \\
\qquad\qquad\qquad \mathit{table}) \\
\qquad = \quad ((\text{type}\,(\mathit{addr}) = \texttt{'addr}) \land (\mathit{addr} \in \text{strip-cdrs}\,(\mathit{table}))))
\end{array}
$$

THEOREM: lr-check-result-lr-push-tstk-quote

$$
\begin{array}{l}
(\text{good-posp1}\,(\text{s-pos}\,(s),\ \text{s-body}\,(\text{s-prog}\,(s))) \\
\ \land \quad \text{listp}\,(\text{s-expr}\,(s)) \\
\ \land \quad (\text{car}\,(\text{s-expr}\,(s)) = \texttt{'quote}) \\
\ \land \quad \text{lr-programs-properp}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table}) \\
\ \land \quad \text{lr-s-similar-statesp}\,(\text{s-params}\,(s),\ \text{s-temps}\,(s),\ \text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table}) \\
\ \land \quad \text{s-good-statep}\,(s,\ c) \\
\ \land \quad \text{lr-proper-heapp}\,(\text{p-data-segment}\,(l)) \\
\ \land \quad (\text{p-psw}\,(\text{lr-push-tstk}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}), \\
\qquad\qquad\qquad\qquad \text{cadr}\,(\text{lr-expr}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}))))) \\
\qquad = \quad \texttt{'run})) \\
\rightarrow \quad \text{lr-check-result}\,(\mathbf{t}, \\
\qquad\qquad\qquad \text{cadr}\,(\text{s-expr}\,(s)), \\
\qquad\qquad\qquad \text{p-temp-stk}\,(\text{lr-push-tstk}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}), \\
\qquad\qquad\qquad\qquad\qquad \text{cadr}\,(\text{lr-expr}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}))))), \\
\qquad\qquad\qquad \text{p-data-segment}\,(l), \\
\qquad\qquad\qquad \text{p-temp-stk}\,(l))
\end{array}
$$

THEOREM: lr-eval-subrp-user-funcall-opener

$$
\begin{array}{l}
\textbf{let}\ \mathit{lr\text{-}eval\text{-}list}\ \ \textbf{be}\ \text{lr-eval}\,(\texttt{'list}, \\
\qquad\qquad\qquad \text{lr-set-pos}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \text{dv}\,(\text{s-pos}\,(s),\ \texttt{1})), \\
\qquad\qquad\qquad c) \\
\textbf{in} \\
((\mathit{flag} \neq \texttt{'list}) \\
\ \land \quad (c \not\simeq \texttt{0}) \\
\ \land \quad \text{listp}\,(\text{s-expr}\,(s)) \\
\ \land \quad (\text{car}\,(\text{s-expr}\,(s)) \neq \texttt{'if}) \\
\ \land \quad (\text{car}\,(\text{s-expr}\,(s)) \neq \text{S-TEMP-EVAL}) \\
\ \land \quad (\text{car}\,(\text{s-expr}\,(s)) \neq \text{S-TEMP-TEST}) \\
\ \land \quad (\text{car}\,(\text{s-expr}\,(s)) \neq \text{S-TEMP-FETCH}) \\
\ \land \quad (\text{car}\,(\text{s-expr}\,(s)) \neq \texttt{'quote}) \\
\ \land \quad \text{good-posp1}\,(\text{s-pos}\,(s),\ \text{s-body}\,(\text{s-prog}\,(s))) \\
\ \land \quad \text{s-good-statep}\,(s,\ c)) \\
\rightarrow \quad (\text{lr-eval}\,(\mathit{flag},\ \text{s->lr1}\,(s,\ l,\ \mathit{table}),\ c)
\end{array}
$$

$=$ **if** p-psw (*lr-eval-list*) $\neq$ `'run` **then** *lr-eval-list*
elseif subrp (car (s-expr (s)))
then lr-apply-subr (s->lr1 (s, l, *table*), *lr-eval-list*)
elseif litatom (car (s-expr (s)))
then lr-set-expr (lr-pop-cstk (lr-eval (**t**,
lr-funcall (s->lr1 (s,
l,
table),
lr-eval-list),
$c - 1$)),
s->lr1 (s, l, *table*),
s-pos (s))
else lr-set-error (s->lr1 (s, l, *table*),
`'bad-instruction`) **endif**) **endlet**

THEOREM: length-cdr-lr-expr-funcall
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ listp (lr-expr (l))
$\wedge$ (subrp (car (lr-expr (l))) $\vee$ litatom (car (lr-expr (l)))))
$\rightarrow$ (length (cdr (lr-expr (l))) $=$ arity (car (lr-expr (l))))

DEFINITION:
induct-hint-8 (n, *value*, *temp-stk*)
$=$ **if** $n \simeq$ `0` **then t**
else induct-hint-8 ($n - 1$, cdr (*value*), cdr (*temp-stk*)) **endif**

THEOREM: lr-check-result1-lr-valp-get-n-lessp-length
(lr-check-result1 (*values*, *temp-stk*, *data-seg*) $\wedge$ ($n <$ length (*values*)))
$\rightarrow$ lr-valp (get (n, *values*), get (n, *temp-stk*), *data-seg*)

THEOREM: lr-valp-lr-good-pointerp
lr-valp (*value*, *addr*, *data-seg*) $\rightarrow$ lr-good-pointerp (*addr*, *data-seg*)

THEOREM: lr-check-result1-lr-good-pointerp-get-n-lessp-car
(lr-check-result1 (*values*, *temp-stk*, *data-seg*) $\wedge$ (length (*values*) $\not<$ `1`))
$\rightarrow$ ((type (car (*temp-stk*)) $=$ `'addr`)
$\wedge$ (cddr (car (*temp-stk*)) $=$ **nil**)
$\wedge$ listp (car (*temp-stk*))
$\wedge$ adpp (untag (car (*temp-stk*)), *data-seg*)
$\wedge$ lr-boundary-nodep (car (*temp-stk*))
$\wedge$ (area-name (car (*temp-stk*)) $=$ identity (LR-HEAP-NAME))
$\wedge$ (type (fetch (add-addr (car (*temp-stk*),
identity (LR-REF-COUNT-OFFSET)),
data-seg))
$=$ `'nat`))

THEOREM: lr-check-result1-lr-good-pointerp-get-n-lessp-cadr

(lr-check-result1 (*values*, *temp-stk*, *data-seg*) $\wedge$ (length (*values*) $\not<$ `2`))
$\rightarrow$ ((type (cadr (*temp-stk*)) = **`'addr`**)
$\wedge$ (cddr (cadr (*temp-stk*)) = **nil**)
$\wedge$ listp (cadr (*temp-stk*))
$\wedge$ adpp (untag (cadr (*temp-stk*)), *data-seg*)
$\wedge$ lr-boundary-nodep (cadr (*temp-stk*))
$\wedge$ (area-name (cadr (*temp-stk*)) = identity (LR-HEAP-NAME))
$\wedge$ (type (fetch (add-addr (cadr (*temp-stk*),
identity (LR-REF-COUNT-OFFSET)),
data-seg))
= **`'nat`**))

THEOREM: p-run-subr-preserves-lr-proper-heapp2

(good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\wedge$ lr-programs-properp (*l*, *table*)
$\wedge$ lr-programs-properp (*new-l*, *table*)
$\wedge$ listp (lr-expr (*l*))
$\wedge$ proper-p-statep (lr->p (*new-l*))
$\wedge$ lr-proper-free-listp (p-data-segment (*new-l*))
$\wedge$ adpp (untag (lr-max-node (p-data-segment (*new-l*))), p-data-segment (*new-l*))
$\wedge$ lr-boundary-nodep (lr-max-node (p-data-segment (*new-l*)))
$\wedge$ (p-psw (*new-l*) = **`'run`**)
$\wedge$ (p-psw (p-run-subr (car (lr-expr (*l*)),
p-set-pc (lr->p (*new-l*), lr-return-pc (*l*))))
= **`'run`**)
$\wedge$ lr-proper-heapp2 (*addr*, p-data-segment (*new-l*))
$\wedge$ lr-nodep (*addr*, p-data-segment (*new-l*))
$\wedge$ (p-prog-segment (*l*) = p-prog-segment (*new-l*))
$\wedge$ lr-check-result (**`'list`**,
value,
p-temp-stk (*new-l*),
p-data-segment (*new-l*),
p-temp-stk (*l*))
$\wedge$ (length (*value*) = length (cdr (lr-expr (*l*)))))
$\rightarrow$ lr-proper-heapp2 (*addr*,
p-data-segment (p-run-subr (car (lr-expr (*l*)),
p-set-pc (lr->p (*new-l*),
lr-return-pc (*l*)))))

THEOREM: lr-apply-subr-preserves-lr-proper-heapp2

let *new-l* **be** lr-eval (**`'list`**, lr-set-pos (*l*, *pos*), *c*)
in
(good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))

$\wedge$ good-posp ('**list**, *pos*, program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ listp (lr-expr (l))
$\wedge$ subrp (car (lr-expr (l)))
$\wedge$ (car (lr-expr (l)) $\neq$ '**if**)
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ lr-proper-free-listp (p-data-segment (l))
$\wedge$ lr-proper-heapp2 (*addr*, p-data-segment (*new-l*))
$\wedge$ adpp (untag (lr-max-node (p-data-segment (l))), p-data-segment (l))
$\wedge$ lr-boundary-nodep (lr-max-node (p-data-segment (l)))
$\wedge$ lr-nodep (*addr*, p-data-segment (l))
$\wedge$ lr-check-result ('**list**,
value,
p-temp-stk (*new-l*),
p-data-segment (*new-l*),
p-temp-stk (l))
$\wedge$ (length (*value*) = length (cdr (lr-expr (l))))
$\wedge$ (p-psw (*new-l*) = '**run**)
$\wedge$ (p-psw (lr-apply-subr (l, *new-l*)) = '**run**))
$\rightarrow$ lr-proper-heapp2 (*addr*, p-data-segment (lr-apply-subr (l, *new-l*))) **endlet**

DEFINITION:
induct-hint-15 (s, c)
= **if** listp (s-pos (s))
then if listp (s-expr-list (s))
then induct-hint-15 (s-set-expr (s-eval (**t**, s, c), s, nx (s-pos (s))), c)
else t endif
else t endif

THEOREM: length-s-eval-list
(listp (s-pos (s)) $\wedge$ (s-err-flag (s-eval ('**list**, s, c)) = '**run**))
$\rightarrow$ (length (s-ans (s-eval ('**list**, s, c))) = length (s-expr-list (s)))

THEOREM: plistp-lr-compile-body
listp (*body*) $\rightarrow$ plistp (lr-compile-body (*flag*, *body*, *temp-alist*, *const-alist*))

THEOREM: plistp-lr-expr-s->lr1
(good-posp1 (s-pos (s), s-body (s-prog (s))) $\wedge$ listp (s-expr (s)))
$\rightarrow$ plistp (lr-expr (s->lr1 (s, l, *table*)))

THEOREM: length-cdr-lr-expr-funcall-s->lr1
(good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\wedge$ listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $\neq$ '**quote**)

$\wedge$ (subrp (car (s-expr (s))) $\vee$ litatom (car (s-expr (s)))))
$\rightarrow$ (length (cdr (lr-expr (s->lr1 (s, l, *table*)))) $=$ length (cdr (s-expr (s))))

THEOREM: adpp-same-signature-lr-apply-subr
same-signature (p-data-segment (*new-l*), p-data-segment (lr-apply-subr (l, *new-l*)))
$\rightarrow$ (adpp (*adp*, p-data-segment (lr-apply-subr (l, *new-l*)))
$=$ adpp (*adp*, p-data-segment (*new-l*)))

THEOREM: lr-apply-subr-preserves-lr-proper-heapp
let *new-l* **be** lr-eval (`'list`, lr-set-pos (s->lr1 (s, l, *table*), *pos*), c)
in
(good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ definedp (s-pname (s), s-progs (s))
$\wedge$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\wedge$ listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $\neq$ S-TEMP-EVAL)
$\wedge$ (car (s-expr (s)) $\neq$ S-TEMP-TEST)
$\wedge$ (car (s-expr (s)) $\neq$ S-TEMP-FETCH)
$\wedge$ (car (s-expr (s)) $\neq$ `'quote`)
$\wedge$ subrp (car (s-expr (s)))
$\wedge$ (car (s-expr (s)) $\neq$ `'if`)
$\wedge$ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\wedge$ lr-check-result (`'list`,
s-ans (s-eval (`'list`, s-set-pos (s, *pos*), c)),
p-temp-stk (*new-l*),
p-data-segment (*new-l*),
p-temp-stk (l))
$\wedge$ (p-psw (*new-l*) $=$ `'run`)
$\wedge$ (p-psw (lr-apply-subr (s->lr1 (s, l, *table*), *new-l*)) $=$ `'run`)
$\wedge$ lr-proper-heapp (p-data-segment (*new-l*))
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ (s-err-flag (s-eval (`'list`, s-set-pos (s, *pos*), c)) $=$ `'run`)
$\wedge$ (*pos* $=$ dv (s-pos (s), `1`)))
$\rightarrow$ lr-proper-heapp (p-data-segment (lr-apply-subr (s->lr1 (s, l, *table*),
new-l))) **endlet**

THEOREM: lr-s-similar-params-lr-apply-subr
let *new-l* **be** lr-eval (`'list`, lr-set-pos (l, *pos*), c)
in
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ listp (lr-expr (l))
$\wedge$ subrp (car (lr-expr (l)))
$\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
$\wedge$ proper-p-statep (lr->p (*new-l*))

$\land$ lr-proper-heapp (p-data-segment (*new-l*))
$\land$ (p-psw (*new-l*) = `'run`)
$\land$ (p-psw (lr-apply-subr (*l*, *new-l*)) = `'run`)
$\land$ lr-s-similar-params (*s-params*, *lr-params*, p-data-segment (*new-l*)))
$\rightarrow$ lr-s-similar-params (*s-params*,
lr-params,
p-data-segment (lr-apply-subr (*l*, *new-l*))) **endlet**

THEOREM: lr-params-lr-apply-subr
((area-name (p-pc (*new-l*)) = area-name (p-pc (*l*)))
$\land$ (p-prog-segment (*new-l*) = p-prog-segment (*l*)))
$\rightarrow$ (lr-params (*frame*, lr-apply-subr (*l*, *new-l*)) = lr-params (*frame*, *l*))

THEOREM: lr-temps-lr-apply-subr
((area-name (p-pc (*new-l*)) = area-name (p-pc (*l*)))
$\land$ (p-prog-segment (*new-l*) = p-prog-segment (*l*)))
$\rightarrow$ (lr-temps (*frame*, lr-apply-subr (*l*, *new-l*)) = lr-temps (*frame*, *l*))

THEOREM: lr-s-similar-temps-lr-apply-subr
let *new-l* **be** lr-eval (`'list`, lr-set-pos (*l*, *pos*), *c*)
in
(good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\land$ lr-programs-properp (*l*, *table*)
$\land$ listp (lr-expr (*l*))
$\land$ subrp (car (lr-expr (*l*)))
$\land$ (car (lr-expr (*l*)) $\neq$ `'if`)
$\land$ proper-p-statep (lr->p (*new-l*))
$\land$ lr-proper-heapp (p-data-segment (*new-l*))
$\land$ (p-psw (*new-l*) = `'run`)
$\land$ (p-psw (lr-apply-subr (*l*, *new-l*)) = `'run`)
$\land$ lr-s-similar-temps (*s-temps*, *lr-temps*, p-data-segment (*new-l*)))
$\rightarrow$ lr-s-similar-temps (*s-temps*,
lr-temps,
p-data-segment (lr-apply-subr (*l*, *new-l*))) **endlet**

THEOREM: lr-s-similar-const-table-lr-apply-subr
let *new-l* **be** lr-eval (`'list`, lr-set-pos (*l*, *pos*), *c*)
in
(good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\land$ lr-programs-properp (*l*, *table1*)
$\land$ listp (lr-expr (*l*))
$\land$ subrp (car (lr-expr (*l*)))
$\land$ (car (lr-expr (*l*)) $\neq$ `'if`)
$\land$ proper-p-statep (lr->p (*new-l*))
$\land$ lr-proper-heapp (p-data-segment (*new-l*))

$\wedge$ (p-psw (*new-l*) = `'run`)
$\wedge$ (p-psw (lr-apply-subr (*l*, *new-l*)) = `'run`)
$\wedge$ lr-s-similar-const-table (*table2*, p-data-segment (*new-l*)))
$\rightarrow$ lr-s-similar-const-table (*table2*,
p-data-segment (lr-apply-subr (*l*, *new-l*))) **endlet**

THEOREM: lr-s-similar-statesp-lr-apply-subr
let *new-l* **be** lr-eval (`'list`, lr-set-pos (*l*, *pos*), *c*)
in
(good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\wedge$ lr-programs-properp (*l*, *table*)
$\wedge$ listp (lr-expr (*l*))
$\wedge$ subrp (car (lr-expr (*l*)))
$\wedge$ (car (lr-expr (*l*)) $\neq$ `'if`)
$\wedge$ proper-p-statep (lr->p (*new-l*))
$\wedge$ lr-proper-heapp (p-data-segment (*new-l*))
$\wedge$ (p-psw (*new-l*) = `'run`)
$\wedge$ (p-psw (lr-apply-subr (*l*, *new-l*)) = `'run`)
$\wedge$ lr-s-similar-statesp (*s-params*, *s-temps*, *new-l*, *table*))
$\rightarrow$ lr-s-similar-statesp (*s-params*,
s-temps,
lr-apply-subr (*l*, *new-l*),
table) **endlet**

THEOREM: proper-p-statep-lr->p-lr-eval-list-helper
let *cur-expr* **be** cur-expr (offset (p-pc (*l*)),
program-body (p-current-program (*l*)))
in
((length (*cur-expr*) < `1`)
$\wedge$ good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\wedge$ listp (*cur-expr*)
$\wedge$ (car (*cur-expr*) $\neq$ `'if`)
$\wedge$ (car (*cur-expr*) $\neq$ `'quote`)
$\wedge$ (litatom (car (*cur-expr*)) $\vee$ subrp (car (*cur-expr*)))
$\wedge$ proper-p-statep (lr->p (*l*))
$\wedge$ lr-programs-properp (*l*, *table*)
$\wedge$ (p-psw (lr-eval (`'list`, lr-set-pos (*l*, *pos*), *c*)) = `'run`)
$\wedge$ (*pos* = dv (offset (p-pc (*l*)), `1`)))
$\rightarrow$ proper-p-statep (lr->p (lr-eval (`'list`, lr-set-pos (*l*, *pos*), *c*))) **endlet**

THEOREM: proper-p-statep-lr->p-lr-eval-list
(good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\wedge$ listp (lr-expr (*l*))
$\wedge$ (car (lr-expr (*l*)) $\neq$ `'if`)
$\wedge$ (car (lr-expr (*l*)) $\neq$ `'quote`)

$\wedge$ (subrp (car (lr-expr (l))) $\vee$ litatom (car (lr-expr (l))))
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (lr-eval (`'list`, lr-set-pos (l, *pos*), c)) = `'run`)
$\wedge$ (*pos* = dv (offset (p-pc (l)), `1`))
$\to$ proper-p-statep (lr->p (lr-eval (`'list`, lr-set-pos (l, *pos*), c)))

Event: Disable proper-p-statep-lr->p-lr-eval-list-helper.

Theorem: not-listp-p-temp-stk-not-lr-check-result1
lr-check-result1 (*value*, *temp-stk*, *data-seg*)
$\to$ (length (*temp-stk*) $\not<$ length (*value*))

Theorem: restn-add1-opener-alt
restn (1 + n, *list*)
= **if** listp (*list*) **then** restn (n, cdr (*list*))
else *list* **endif**

Theorem: cdr-p-temp-stk-p-run-subr
let *new-l* **be** lr-eval (`'list`, lr-set-pos (l, *pos*), c)
in
(listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
$\wedge$ subrp (car (lr-expr (l)))
$\wedge$ (p-temp-stk (l)
= restn (length (cdr (lr-expr (l))), p-temp-stk (*new-l*)))
$\wedge$ lr-check-result1 (*value*,
p-temp-stk (*new-l*),
p-data-segment (*new-l*))
$\wedge$ (length (*value*) = length (cdr (lr-expr (l))))
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (p-run-subr (car (lr-expr (l)),
p-set-pc (lr->p (*new-l*), lr-return-pc (l))))
= `'run`)
$\wedge$ (p-psw (*new-l*) = `'run`)
$\wedge$ (*pos* = dv (offset (p-pc (l)), `1`)))
$\to$ (cdr (p-temp-stk (p-run-subr (car (lr-expr (l)),
p-set-pc (lr->p (*new-l*),
lr-return-pc (l)))))
= p-temp-stk (l)) **endlet**

Event: Disable restn-add1-opener-alt.

THEOREM: cdr-p-temp-stk-lr-apply-subr
let *new-l* **be** lr-eval ('`list`, lr-set-pos (*l*, *pos*), *c*)
in
(listp (lr-expr (*l*))
$\land$ (car (lr-expr (*l*)) $\neq$ '`if`)
$\land$ subrp (car (lr-expr (*l*)))
$\land$ (p-temp-stk (*l*)
    $=$ restn (length (cdr (lr-expr (*l*))), p-temp-stk (*new-l*)))
$\land$ lr-check-result1 (*value*,
        p-temp-stk (*new-l*),
        p-data-segment (*new-l*))
$\land$ (length (*value*) $=$ length (cdr (lr-expr (*l*))))
$\land$ proper-p-statep (lr->p (*l*))
$\land$ good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\land$ lr-programs-properp (*l*, *table*)
$\land$ (p-psw (*new-l*) $=$ '`run`)
$\land$ (p-psw (lr-apply-subr (*l*, *new-l*)) $=$ '`run`)
$\land$ (*pos* $=$ dv (offset (p-pc (*l*)), `1`)))
$\rightarrow$ (cdr (p-temp-stk (lr-apply-subr (*l*, *new-l*))) $=$ p-temp-stk (*l*)) **endlet**

THEOREM: lr-check-result1-reverse-length-1-opener
(length (*values*) $=$ `1`)
$\rightarrow$ (lr-check-result1 (reverse (*values*), *temp-stk*, *data-seg*)
    $=$ (lr-valp (car (*values*), car (*temp-stk*), *data-seg*)
        $\land$ lr-good-pointerp (car (*temp-stk*), *data-seg*)))

THEOREM: lr-check-result1-reverse-length-2-opener
(length (*values*) $=$ `2`)
$\rightarrow$ (lr-check-result1 (reverse (*values*), *temp-stk*, *data-seg*)
    $=$ (lr-valp (cadr (*values*), car (*temp-stk*), *data-seg*)
        $\land$ lr-good-pointerp (cadr (*temp-stk*), *data-seg*)
        $\land$ lr-valp (car (*values*), cadr (*temp-stk*), *data-seg*)
        $\land$ lr-good-pointerp (car (*temp-stk*), *data-seg*)))

THEOREM: lr-valp-fetch-tag-cons-lr-valp-car-cdr
(lr-valp (*value*, *addr*, *data-seg*)
$\land$ (fetch (*addr*, *data-seg*) $=$ tag ('`nat`, LR-CONS-TAG)))
$\rightarrow$ (lr-valp (car (*value*),
        fetch (add-addr (*addr*, identity (LR-CAR-OFFSET)), *data-seg*),
        data-seg)
    $\land$ lr-valp (cdr (*value*),
            fetch (add-addr (*addr*, identity (LR-CDR-OFFSET)), *data-seg*),
            data-seg))

THEOREM: lr-good-pointerp-type-tag-nat

(lr-proper-heapp (*data-seg*) ∧ lr-good-pointerp (*addr*, *data-seg*))
→ (type (fetch (*addr*, *data-seg*)) = **'nat**)

THEOREM: lr-proper-heapp-lr-valp-f-helper
(lr-proper-heapp-nodep (LR-F-ADDR, *data-seg*)
∧ lr-proper-heapp-nodep (*addr*, *data-seg*)
∧ lr-nodep (*addr*, *data-seg*)
∧ lr-minimum-heapp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*))
→ (lr-valp (**f**, *addr*, *data-seg*) = (*addr* = LR-F-ADDR))

THEOREM: lr-proper-heapp-lr-valp-f
(lr-proper-heapp (*data-seg*) ∧ lr-proper-p-areasp (*data-seg*))
→ (lr-valp (**f**, *addr*, *data-seg*) = (*addr* = LR-F-ADDR))

THEOREM: lr-valp-equal-value-fact
(lr-valp (*value1*, *addr*, *data-seg*) ∧ lr-valp (*value2*, *addr*, *data-seg*))
→ (*value1* = *value2*)

THEOREM: lr-proper-heapp-lr-valp-0
lr-proper-heapp (*data-seg*)
→ (lr-valp (*value*, identity (LR-0-ADDR), *data-seg*) = (*value* = 0))

THEOREM: lr-proper-heapp-lr-valp-lr-f-addr
(lr-proper-heapp (*data-seg*) ∧ lr-proper-p-areasp (*data-seg*))
→ (lr-valp (*value*, identity (LR-F-ADDR), *data-seg*) = (*value* = **f**))

THEOREM: lr-proper-heapp-lr-valp-lr-t-addr
lr-proper-heapp (*data-seg*)
→ (lr-valp (*value*, identity (LR-T-ADDR), *data-seg*) = (*value* = **t**))

THEOREM: lr-valp-fetch-tag-not-cons-lr-valp-car-cdr-0
(proper-p-statep (lr->p (*l*))
∧ lr-proper-heapp (p-data-segment (*l*))
∧ (fetch (car (p-temp-stk (*l*)), p-data-segment (*l*))
≠ tag (**'nat**, LR-CONS-TAG))
∧ ((car (*value*) ≠ 0) ∨ (cdr (*value*) ≠ 0)))
→ (¬ lr-valp (*value*, car (p-temp-stk (*l*)), p-data-segment (*l*)))

THEOREM: lr-valp-not-tag-cons-not-listp
((¬ listp (*value*)) ∧ (fetch (*addr*, *data-seg*) = tag (**'nat**, LR-CONS-TAG)))
→ (¬ lr-valp (*value*, *addr*, *data-seg*))

THEOREM: lr-valp-fetch-tag-not-cons-lr-valp-listp
(proper-p-statep (lr->p (*l*))
∧ lr-proper-heapp (p-data-segment (*l*))

$\land$ (fetch (car (p-temp-stk (l)), p-data-segment (l))
  $\neq$ tag ('**nat**, LR-CONS-TAG))
$\land$ listp (*value*))
$\rightarrow$ ($\neg$ lr-valp (*value*, car (p-temp-stk (l)), p-data-segment (l)))

THEOREM: lr-valp-cons
lr-valp (cons (x, y), *addr*, *data-seg*)
= **if** lr-good-pointerp (*addr*, *data-seg*)
  then (untag (fetch (*addr*, *data-seg*)) = LR-CONS-TAG)
    $\land$ lr-valp (x,
      fetch (add-addr (*addr*, LR-CAR-OFFSET), *data-seg*),
      data-seg)
    $\land$ lr-valp (y,
      fetch (add-addr (*addr*, LR-CDR-OFFSET), *data-seg*),
      data-seg)
  else f endif

THEOREM: lr-valp-deposit-a-list-cons
(lr-proper-free-listp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ lr-nodep (lr-max-node (*data-seg*), *data-seg*)
$\land$ lr-minimum-heapp (*data-seg*)
$\land$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\land$ lr-valp (*value*, *addr*, *data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ (*fp-addr* = fetch (identity (LR-FP-ADDR), *data-seg*)))
$\rightarrow$ lr-valp (*value*, *addr*, deposit-a-list (list (*x0*, *x1*, *x2*, *x3*), *fp-addr*, *data-seg*))

THEOREM: lr-valp-car-p-temp-stk-p-run-subr-cons-helper
(lr-proper-heapp (*data-seg*)
$\land$ lr-valp (*car*, *car-addr*, *data-seg*)
$\land$ lr-valp (*cdr*, *cdr-addr*, *data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ (type (*ref-count*) = '**nat**))
$\rightarrow$ lr-valp (cons (*car*, *cdr*),
    fetch (identity (LR-FP-ADDR), *data-seg*),
    deposit-a-list (list (identity (tag ('**nat**, LR-CONS-TAG)),
        ref-count,
        car-addr,
        cdr-addr),
      fetch (identity (LR-FP-ADDR), *data-seg*),
      data-seg))

EVENT: Disable lr-valp-cons.

THEOREM: lr-valp-not-tag-true-not-listp
((¬ truep (*value*)) ∧ (fetch (*addr*, *data-seg*) = tag ('**nat**, LR-TRUE-TAG)))
→ (¬ lr-valp (*value*, *addr*, *data-seg*))

THEOREM: lr-valp-fetch-tag-not-true-lr-valp-listp
(proper-p-statep (lr->p (*l*))
∧ lr-proper-heapp (p-data-segment (*l*))
∧ (fetch (car (p-temp-stk (*l*)), p-data-segment (*l*))
≠ tag ('**nat**, LR-TRUE-TAG)))
→ (¬ lr-valp (**t**, car (p-temp-stk (*l*)), p-data-segment (*l*)))

THEOREM: lr-valp-car-p-temp-stk-p-run-subr
(lr-proper-heapp (p-data-segment (*l*))
∧ lr-check-result1 (reverse (*values*), p-temp-stk (*l*), p-data-segment (*l*))
∧ (length (*values*) = arity (*subr*))
∧ proper-p-statep (lr->p (*l*))
∧ lr-programs-properp (*l*, *table*)
∧ (p-psw (p-run-subr (*subr*, p-set-pc (lr->p (*l*), *pc*))) = '**run**)
∧ (p-psw (*l*) = '**run**)
∧ (area-name (*pc*) = area-name (p-pc (*l*)))
∧ (unlabel (get (offset (*pc*),
program-body (assoc (area-name (*pc*),
p-prog-segment (lr->p (*l*))))))
= list ('**call**, *subr*)))
→ lr-valp (apply-subr (*subr*, *values*),
car (p-temp-stk (p-run-subr (*subr*, p-set-pc (lr->p (*l*), *pc*)))),
p-data-segment (p-run-subr (*subr*, p-set-pc (lr->p (*l*), *pc*))))

THEOREM: lr-programs-properp-not-definedp-subrp-runtime-support
(subrp (car (s-expr (*s*)))
∧ listp (s-expr (*s*))
∧ (car (s-expr (*s*)) ≠ '**if**)
∧ (¬ definedp (car (s-expr (*s*)), P-RUNTIME-SUPPORT-PROGRAMS))
∧ good-posp1 (s-pos (*s*), s-body (s-prog (*s*))))
→ (¬ lr-programs-properp (s->lr1 (*s*, *l*, *table*), *table*))

THEOREM: lr-valp-apply-subr-lr-apply-subr
let *new-l* **be** lr-eval ('**list**, lr-set-pos (s->lr1 (*s*, *l*, *table*), *pos*), *c*)
in
(listp (s-expr (*s*))
∧ (car (s-expr (*s*)) ≠ '**if**)
∧ (car (s-expr (*s*)) ≠ '**quote**)
∧ (car (s-expr (*s*)) ≠ S-TEMP-EVAL)
∧ (car (s-expr (*s*)) ≠ S-TEMP-TEST)
∧ (car (s-expr (*s*)) ≠ S-TEMP-FETCH)

```
 ∧   subrp (car (s-expr (s)))
 ∧   lr-proper-heapp (p-data-segment (new-l))
 ∧   lr-check-result1 (reverse (values),
                       p-temp-stk (new-l),
                       p-data-segment (new-l))
 ∧   (length (values) = length (cdr (lr-expr (s->lr1 (s, l, table)))))
 ∧   proper-p-statep (lr->p (s->lr1 (s, l, table)))
 ∧   good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧   lr-programs-properp (s->lr1 (s, l, table), table)
 ∧   s-good-statep (s, c)
 ∧   (p-psw (new-l) = 'run)
 ∧   (p-psw (lr-apply-subr (s->lr1 (s, l, table), new-l)) = 'run)
 ∧   (pos = dv (s-pos (s), 1))
 →   lr-valp (apply-subr (car (s-expr (s)), values),
              car (p-temp-stk (lr-apply-subr (s->lr1 (s, l, table), new-l))),
              p-data-segment (lr-apply-subr (s->lr1 (s, l, table), new-l))) endlet
```

EVENT: Disable lr-programs-properp-not-definedp-subrp-runtime-support.

THEOREM: lr-check-result-lr-apply-subr

```
let new-l be lr-eval ('list,
                      lr-set-pos (s->lr1 (s, l, table), dv (s-pos (s), 1)),
                      c),
    pos be dv (s-pos (s), 1)
in
(listp (s-expr (s))
 ∧   (car (s-expr (s)) ≠ 'if)
 ∧   (car (s-expr (s)) ≠ 'quote)
 ∧   (car (s-expr (s)) ≠ S-TEMP-EVAL)
 ∧   (car (s-expr (s)) ≠ S-TEMP-TEST)
 ∧   (car (s-expr (s)) ≠ S-TEMP-FETCH)
 ∧   subrp (car (s-expr (s)))
 ∧   lr-check-result ('list,
                      s-ans (s-eval ('list, s-set-pos (s, pos), c)),
                      p-temp-stk (new-l),
                      p-data-segment (new-l),
                      p-temp-stk (l))
 ∧   proper-p-statep (lr->p (s->lr1 (s, l, table)))
 ∧   lr-proper-heapp (p-data-segment (l))
 ∧   good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧   lr-programs-properp (s->lr1 (s, l, table), table)
 ∧   s-good-statep (s, c)
 ∧   (p-psw (new-l) = 'run)
 ∧   (p-psw (lr-apply-subr (s->lr1 (s, l, table), new-l)) = 'run)
```

```
 ∧   (s-err-flag (s-eval ('list, s-set-pos (s, dv (s-pos (s), 1)), c))
       =   'run))
 →   lr-check-result (t,
                       apply-subr (car (s-expr (s)),
                                   s-ans (s-eval ('list,
                                                  s-set-pos (s, pos),
                                                  c))),
                       p-temp-stk (lr-apply-subr (s->lr1 (s, l, table),
                                                   new-l)),
                       p-data-segment (lr-apply-subr (s->lr1 (s, l, table),
                                                       new-l)),
                       p-temp-stk (l)) endlet
```

THEOREM: s->lr1-lr-funcall-s-fun-call-state

```
(listp (s-expr (s))
 ∧   (¬ subrp (car (s-expr (s))))
 ∧   (car (s-expr (s)) ≠ 'quote)
 ∧   (car (s-expr (s)) ≠ 'if)
 ∧   litatom (car (s-expr (s)))
 ∧   good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧   (p-psw (lr-funcall (s->lr1 (s, l, table), lr-eval)) = 'run)
 ∧   (s-progs (s-eval) = s-progs (s))
 ∧   (p-prog-segment (lr-eval) = p-prog-segment (s->lr1 (s, l, table))))
 →   (s->lr1 (s-fun-call-state (s-eval, car (s-expr (s))),
              lr-funcall (s->lr1 (s, l, table), lr-eval),
              table)
       =   lr-funcall (s->lr1 (s, l, table), lr-eval))
```

THEOREM: lr-params-lr-funcall

```
((p-psw (lr-funcall (l1, l2)) = 'run)
 ∧   (p-prog-segment (l1) = p-prog-segment (l2)))
 →   (lr-params (car (p-ctrl-stk (lr-funcall (l1, l2))), lr-funcall (l1, l2))
       =   pair-formal-vars-with-actuals (formal-vars (assoc (user-fname (car (lr-expr (l1))),
                                                              p-prog-segment (l1))),
                                          p-temp-stk (l2)))
```

THEOREM: lr-temps-lr-funcall

```
((p-psw (lr-funcall (l1, l2)) = 'run)
 ∧   (p-prog-segment (l1) = p-prog-segment (l2)))
 →   (lr-temps (car (p-ctrl-stk (lr-funcall (l1, l2))), lr-funcall (l1, l2))
       =   pair-temps-with-initial-values (temp-var-dcls (assoc (user-fname (car (lr-expr (l1))),
                                                                p-prog-segment (l1)))))
```

THEOREM: listp-pairlist

```
listp (pairlist (x, y)) = listp (x)
```

THEOREM: car-reverse-last
car (reverse (*list*)) = car (last (*list*))

THEOREM: get-sub1-length-car-last
(listp (*list*) $\wedge$ (n = (length (*list*) $-$ 1)))
$\rightarrow$ (get (n, *list*) = car (last (*list*)))

THEOREM: car-last-first-n-add1-get
car (last (first-n (1 + n, *list*))) = get (n, *list*)

THEOREM: length-butlast
length (butlast (x)) = (length (x) $-$ 1)

DEFINITION:
induct-hint-1 (x, y, z)
= **if** listp (x)
  then if listp (y)
    then if listp (z)
      then induct-hint-1 (cdr (x), butlast (y), butlast (z))
      else t endif
    else t endif
  else t endif

THEOREM: lr-check-result1-append-2
(length (*values*) = length (*temp-stk1*))
$\rightarrow$ (lr-check-result1 (*values*, append (*temp-stk1*, *temp-stk2*), *data-seg*)
  = lr-check-result1 (*values*, *temp-stk1*, *data-seg*))

THEOREM: lr-check-result1-butlast
(lr-check-result1 (*values*, *temp-stk*, *data-seg*)
$\wedge$ (length (*temp-stk*) = length (*values*))
$\wedge$ listp (*temp-stk*)
$\wedge$ listp (*values*))
$\rightarrow$ lr-check-result1 (butlast (*values*), butlast (*temp-stk*), *data-seg*)

THEOREM: reverse-butlast
listp (x) $\rightarrow$ (reverse (butlast (x)) = cdr (reverse (x)))

THEOREM: lr-s-similar-params-lr-valp-get
((n < length (*s-params*))
$\wedge$ (strip-cars (*s-params*) = strip-cars (*lr-params*))
$\wedge$ lr-s-similar-params (*s-params*, *lr-params*, *data-seg*))
$\rightarrow$ lr-valp (cdr (get (n, *s-params*)), cdr (get (n, *lr-params*)), *data-seg*)

THEOREM: lr-s-similar-params-lr-funcall-helper-1

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-s-similar-params}\,(\text{pairlist}\,(\text{cdr}\,(\mathit{formals}),\ \text{cdr}\,(\text{reverse}\,(\mathit{values}))),} \\
& \quad \text{pairlist}\,(\text{cdr}\,(\mathit{formals}),\ \text{cdr}\,(\text{reverse}\,(\mathit{temp\text{-}stk}))), \\
& \quad \mathit{data\text{-}seg}) \\
\land & \text{listp}\,(\mathit{formals}) \\
\land & \text{listp}\,(\mathit{values}) \\
\land & \text{listp}\,(\mathit{temp\text{-}stk}) \\
\land & \text{lr-check-result1}\,(\mathit{values},\ \mathit{temp\text{-}stk},\ \mathit{data\text{-}seg}) \\
\land & ((1 + \text{length}\,(\text{cdr}\,(\mathit{formals}))) = \text{length}\,(\mathit{temp\text{-}stk})) \\
\land & (\text{length}\,(\mathit{temp\text{-}stk}) = \text{length}\,(\mathit{values}))) \\
\to & \text{lr-valp}\,(\text{car}\,(\text{last}\,(\mathit{values})),\ \text{car}\,(\text{last}\,(\mathit{temp\text{-}stk})),\ \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: lr-s-similar-params-lr-funcall

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-check-result1}\,(\mathit{values},\ \mathit{temp\text{-}stk},\ \mathit{data\text{-}seg})} \\
\land & (\text{length}\,(\mathit{temp\text{-}stk}) = \text{length}\,(\mathit{values})) \\
\land & (\text{length}\,(\mathit{temp\text{-}stk}) = \text{length}\,(\mathit{formals}))) \\
\to & \text{lr-s-similar-params}\,(\text{pairlist}\,(\mathit{formals},\ \text{reverse}\,(\mathit{values})), \\
& \quad \text{pairlist}\,(\mathit{formals},\ \text{reverse}\,(\mathit{temp\text{-}stk})), \\
& \quad \mathit{data\text{-}seg})
\end{array}
$$

THEOREM: append-first-n-restn

$$(\text{length}\,(l) \not< i) \to (\text{append}\,(\text{first-n}\,(i,\ l),\ \text{restn}\,(i,\ l)) = l)$$

THEOREM: lr-check-result1-first-n-temp-stk

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{length}\,(\text{p-temp-stk}\,(l)) \not< \text{length}\,(\mathit{values}))} \\
\to & (\text{lr-check-result1}\,(\mathit{values},\ \text{p-temp-stk}\,(l),\ \mathit{data\text{-}seg}) \\
& \quad = \ \text{lr-check-result1}\,(\mathit{values}, \\
& \qquad\qquad \text{first-n}\,(\text{length}\,(\mathit{values}),\ \text{p-temp-stk}\,(l)), \\
& \qquad\qquad \mathit{data\text{-}seg}))
\end{array}
$$

THEOREM: lr-push-tstk-length

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{p-psw}\,(\text{lr-push-tstk}\,(l,\ \mathit{object})) = \texttt{'run})} \\
\to & (\text{length}\,(\text{p-temp-stk}\,(\text{lr-push-tstk}\,(l,\ \mathit{object}))) \\
& \quad = \ (1 + \text{length}\,(\text{p-temp-stk}\,(l))))
\end{array}
$$

THEOREM: length-add1-add1-cddr-fact

$$(\text{length}\,(x) = (1 + (1 + \text{length}\,(y)))) \to (\text{length}\,(\text{cddr}\,(x)) = \text{length}\,(y))$$

THEOREM: length-p-temp-stk-p-run-subr-helper-1

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{length}\,(\text{p-temp-stk}\,(\text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ \mathit{pos}),\ c)))} \\
= & (1 + (1 + \text{length}\,(\text{p-temp-stk}\,(l))))) \\
\to & (\text{length}\,(\text{cddr}\,(\text{p-temp-stk}\,(\text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ \mathit{pos}),\ c)))) \\
& \quad = \ \text{length}\,(\text{p-temp-stk}\,(l)))
\end{array}
$$

THEOREM: length-p-temp-stk-p-run-subr

let *new-l* **be** lr-eval ($'$`list`, lr-set-pos (l, *pos*), c)
in
(listp (lr-expr (l))
 $\wedge$ (car (lr-expr (l)) $\neq$ $'$`if`)
 $\wedge$ subrp (car (lr-expr (l)))
 $\wedge$ (length (p-temp-stk (*new-l*))
  $=$ (length (p-temp-stk (l)) $+$ arity (car (lr-expr (l)))))
 $\wedge$ proper-p-statep (lr->p (l))
 $\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 $\wedge$ lr-programs-properp (l, *table*)
 $\wedge$ (p-psw (p-run-subr (car (lr-expr (l)),
   p-set-pc (lr->p (*new-l*), lr-return-pc (l))))
  $=$ $'$`run`)
 $\wedge$ (p-psw (*new-l*) $=$ $'$`run`)
 $\wedge$ (*pos* $=$ dv (offset (p-pc (l)), `1`)))
$\rightarrow$ (length (p-temp-stk (p-run-subr (car (lr-expr (l)),
   p-set-pc (lr->p (*new-l*),
    lr-return-pc (l)))))
  $=$ (1 $+$ length (p-temp-stk (l)))) **endlet**

THEOREM: length-p-temp-stk-lr-apply-subr
let *new-l* **be** lr-eval ($'$`list`, lr-set-pos (l, *pos*), c)
in
(listp (lr-expr (l))
 $\wedge$ (car (lr-expr (l)) $\neq$ $'$`if`)
 $\wedge$ subrp (car (lr-expr (l)))
 $\wedge$ (length (p-temp-stk (*new-l*))
  $=$ (length (p-temp-stk (l)) $+$ arity (car (lr-expr (l)))))
 $\wedge$ proper-p-statep (lr->p (l))
 $\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 $\wedge$ lr-programs-properp (l, *table*)
 $\wedge$ (p-psw (*new-l*) $=$ $'$`run`)
 $\wedge$ (p-psw (lr-apply-subr (l, *new-l*)) $=$ $'$`run`)
 $\wedge$ (*pos* $=$ dv (offset (p-pc (l)), `1`)))
$\rightarrow$ (length (p-temp-stk (lr-apply-subr (l, *new-l*)))
  $=$ (1 $+$ length (p-temp-stk (l)))) **endlet**

DEFINITION:
lr-proper-formalsp (*programs*)
$=$ **if** listp (*programs*)
 then ((logic-fname (name (car (*programs*))) $=$ $'$`quote`)
   $\vee$ (length (formal-vars (car (*programs*)))
    $=$ arity (logic-fname (name (car (*programs*))))))
  $\wedge$ lr-proper-formalsp (cdr (*programs*))
 else t endif

THEOREM: length-formal-vars-lr-proper-formalsp-arity
(definedp ($name$, $programs$)
 ∧ (logic-fname ($name$) ≠ `'quote`)
 ∧ lr-proper-formalsp ($programs$))
→ (length (formal-vars (assoc ($name$, $programs$)))
   = arity (logic-fname ($name$)))

THEOREM: arity-formals-not-quote
(formals ($name$) ∧ ($name$ ≠ `'quote`))
→ (arity ($name$) = length (formals ($name$)))

THEOREM: lr-proper-formalsp-lr-compile-programs
s-programs-okp ($programs$)
→ lr-proper-formalsp (lr-compile-programs ($programs$, $table$))

EVENT: Disable arity-formals-not-quote.

EVENT: Disable lr-proper-formalsp.

THEOREM: lr-programs-properp-funcall-not-caar-prog-seg
(listp (lr-expr (l))
 ∧ (car (lr-expr (l)) ≠ `'if`)
 ∧ (car (lr-expr (l)) ≠ `'quote`)
 ∧ (¬ subrp (car (lr-expr (l))))
 ∧ litatom (car (lr-expr (l)))
 ∧ listp (p-prog-segment (l))
 ∧ (user-fname (car (lr-expr (l))) = caar (p-prog-segment (l)))
 ∧ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l))))
→ (¬ lr-programs-properp (l, $table$))

THEOREM: length-p-temp-stk-lr-funcall
(listp (lr-expr (l))
 ∧ (car (lr-expr (l)) ≠ `'if`)
 ∧ (car (lr-expr (l)) ≠ `'quote`)
 ∧ (p-psw ($new\text{-}l$) = `'run`)
 ∧ (¬ subrp (car (lr-expr (l))))
 ∧ litatom (car (lr-expr (l)))
 ∧ (length (p-temp-stk (lr-eval (**t**, lr-funcall (l, $new\text{-}l$), $c - 1$)))
   = (1 + length (p-temp-stk (lr-funcall (l, $new\text{-}l$)))))
 ∧ (length (p-temp-stk ($new\text{-}l$))
   = (length (p-temp-stk (l)) + arity (car (lr-expr (l)))))
 ∧ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, $table$)
 ∧ lr-proper-formalsp (cdr (p-prog-segment (l)))

$\wedge$ (p-psw (lr-funcall (l, *new-l*)) = `'run`))
$\rightarrow$ (length (p-temp-stk (lr-funcall (l, *new-l*))) = length (p-temp-stk (l)))

THEOREM: length-p-temp-stk-lr-eval
(proper-p-statep (lr->p (l))
$\wedge$ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ lr-proper-formalsp (cdr (p-prog-segment (l)))
$\wedge$ (p-psw (lr-eval (*flag*, l, c)) = `'run`))
$\rightarrow$ (length (p-temp-stk (lr-eval (*flag*, l, c)))
= **if** *flag* = `'list`
then length (lr-expr-list (l)) + length (p-temp-stk (l))
else `1` + length (p-temp-stk (l)) **endif**)

THEOREM: length-p-temp-stk-lr-eval-flag-list
(proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\wedge$ good-posp (`'list`, *pos*, s-body (s-prog (s)))
$\wedge$ s-good-statep (s, c)
$\wedge$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\wedge$ (p-psw (lr-eval (`'list`, lr-set-pos (s->lr1 (s, l, *table*), *pos*), c))
= `'run`))
$\rightarrow$ (length (p-temp-stk (lr-eval (`'list`,
lr-set-pos (s->lr1 (s, l, *table*), *pos*),
c)))
= (length (s-expr-list (s-set-pos (s, *pos*)))
+ length (p-temp-stk (l))))

THEOREM: reverse-reverse-alt
reverse (reverse (l)) = plist (l)

THEOREM: pairlist-plist-1
pairlist (x, plist (y)) = pairlist (x, y)

THEOREM: s-good-statep-length-cdr-s-expr-funcall
(s-good-statep (s, c)
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $\neq$ `'quote`)
$\wedge$ (car (s-expr (s)) $\neq$ `'if`)
$\wedge$ (litatom (car (s-expr (s))) $\vee$ subrp (car (s-expr (s)))))
$\rightarrow$ (length (cdr (s-expr (s))) = arity (car (s-expr (s))))

THEOREM: lr-s-similar-temps-make-temps-pair-temps
lr-s-similar-temps (make-temps-entries (*temp-list*),
pair-temps-with-initial-values (lr-make-temp-var-dcls (lr-make-temp-name-alist-1 (*initial*,

num-list,
temp-list
formals))
data-seg)

THEOREM: lr-s-similar-temps-lr-funcall
lr-s-similar-temps (make-temps-entries (s-temp-list (assoc (*name*, *progs*))),
pair-temps-with-initial-values (temp-var-dcls (assoc (*name*,
lr-compile-programs (*progs*,
table)))),
data-seg)

THEOREM: lr-eval-preserves-lr-s-similar-const-table
(proper-p-statep (lr->p (l))
$\land$ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table1*)
$\land$ (p-psw (lr-eval (*flag*, l, c)) = `'run`)
$\land$ lr-proper-heapp (p-data-segment (l))
$\land$ lr-s-similar-const-table (*table2*, p-data-segment (l)))
$\rightarrow$ lr-s-similar-const-table (*table2*, p-data-segment (lr-eval (*flag*, l, c)))

THEOREM: lr-s-similar-statesp-lr-funcall
let *pos* **be** dv (s-pos (s), `1`)
in
(listp (s-expr (s))
$\land$ ($\neg$ subrp (car (s-expr (s))))
$\land$ (car (s-expr (s)) $\neq$ `'quote`)
$\land$ (car (s-expr (s)) $\neq$ `'if`)
$\land$ litatom (car (s-expr (s)))
$\land$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\land$ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\land$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\land$ s-good-statep (s, c)
$\land$ (p-psw (lr-funcall (s->lr1 (s, l, *table*), *lr-eval*)) = `'run`)
$\land$ (p-psw (*lr-eval*) = `'run`)
$\land$ lr-check-result (`'list`,
values,
p-temp-stk (*lr-eval*),
p-data-segment (*lr-eval*),
p-temp-stk (l))
$\land$ lr-proper-heapp (p-data-segment (l))
$\land$ (s-err-flag (s-eval (`'list`, s-set-pos (s, *pos*), c)) = `'run`)
$\land$ (*formals* = s-formals (assoc (user-fname (car (s-expr (s))),
s-progs (s))))
$\land$ (*lr-eval* = lr-eval (`'list`,

lr-set-pos (s->lr1 (s, l, *table*), *pos*),
c))
∧ (*values* = s-ans (s-eval (`'list`, s-set-pos (s, *pos*), c)))
∧ lr-s-similar-statesp (s-params (s),
s-temps (s),
s->lr1 (s, l, *table*),
table))
→ lr-s-similar-statesp (pairlist (*formals*, *values*),
make-temps-entries (s-temp-list (assoc (user-fname (car (s-expr (s))),
s-progs (s)))),
lr-funcall (s->lr1 (s, l, *table*), *lr-eval*),
table) **endlet**

THEOREM: lr-params-lr-set-expr-lr-pop-cstk
((area-name (p-pc (l)) = area-name (p-pc (*new-l*)))
∧ (p-prog-segment (l) = p-prog-segment (*new-l*)))
→ (lr-params (*frame*,
lr-set-expr (lr-pop-cstk (lr-eval (**t**, lr-funcall (l, *new-l*), c)),
l,
pos))
= lr-params (*frame*, l))

THEOREM: lr-temps-lr-set-expr-lr-pop-cstk
((area-name (p-pc (l)) = area-name (p-pc (*new-l*)))
∧ (p-prog-segment (l) = p-prog-segment (*new-l*)))
→ (lr-temps (*frame*,
lr-set-expr (lr-pop-cstk (lr-eval (**t**, lr-funcall (l, *new-l*), c)),
l,
pos))
= lr-temps (*frame*, l))

THEOREM: lr-eval-preserves-lr-s-similar-params
(proper-p-statep (lr->p (l))
∧ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
∧ lr-programs-properp (l, *table*)
∧ (p-psw (lr-eval (*flag*, l, c)) = `'run`)
∧ lr-proper-heapp (p-data-segment (l))
∧ lr-s-similar-params (*s-params*, *lr-params*, p-data-segment (l)))
→ lr-s-similar-params (*s-params*,
lr-params,
p-data-segment (lr-eval (*flag*, l, c)))

THEOREM: lr-eval-preserves-lr-s-similar-temps
(proper-p-statep (lr->p (l))
∧ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))

$\land$ lr-programs-properp (l, *table*)
$\land$ (p-psw (lr-eval (*flag*, l, c)) = `'run`)
$\land$ lr-proper-heapp (p-data-segment (l))
$\land$ lr-s-similar-temps (*s-temps*, *lr-temps*, p-data-segment (l)))
$\rightarrow$ lr-s-similar-temps (*s-temps*, *lr-temps*, p-data-segment (lr-eval (*flag*, l, c)))

THEOREM: lr-s-similar-statesp-lr-set-expr-lr-pop-cstk
let *funcall* **be** lr-funcall (s->lr1 (s, l, *table*),
    lr-eval (`'list`,
        lr-set-pos (s->lr1 (s, l, *table*), *pos*),
        c)),
  lr-eval **be** lr-eval (`'list`, lr-set-pos (s->lr1 (s, l, *table*), *pos*), c)
in
(lr-s-similar-statesp (s-params (s), s-temps (*s-eval*), *lr-eval*, *table*)
$\land$ listp (s-expr (s))
$\land$ ($\neg$ subrp (car (s-expr (s))))
$\land$ (car (s-expr (s)) $\neq$ `'quote`)
$\land$ (car (s-expr (s)) $\neq$ `'if`)
$\land$ litatom (car (s-expr (s)))
$\land$ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\land$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\land$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\land$ lr-proper-heapp (p-data-segment (*lr-eval*))
$\land$ (*pos* = dv (s-pos (s), `1`))
$\land$ (p-psw (lr-eval (`t`, *funcall*, $c - 1$)) = `'run`)
$\land$ (p-psw (*lr-eval*) = `'run`))
$\rightarrow$ lr-s-similar-statesp (s-params (s),
    s-temps (*s-eval*),
    lr-set-expr (lr-pop-cstk (lr-eval (`t`,
            funcall,
            $c - 1$)),
        s->lr1 (s, l, *table*),
        s-pos (s)),
    table) **endlet**

THEOREM: popn-restn
(length (*list*) $\not<$ n) $\rightarrow$ (popn (n, *list*) = restn (n, *list*))

THEOREM: lr-check-result-lr-funcall
let *new-l* **be** lr-eval (`'list`,
    lr-set-pos (s->lr1 (s, l, *table*), dv (s-pos (s), `1`)),
    c),
  pos **be** dv (s-pos (s), `1`),
  s-eval **be** s-eval (`'list`, s-set-pos (s, dv (s-pos (s), `1`)), c)
in

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\text{s-expr}\,(s))} \\
\land & (\text{car}\,(\text{s-expr}\,(s)) \neq \texttt{'if}) \\
\land & (\text{car}\,(\text{s-expr}\,(s)) \neq \texttt{'quote}) \\
\land & (\neg\ \text{subrp}\,(\text{car}\,(\text{s-expr}\,(s)))) \\
\land & \text{litatom}\,(\text{car}\,(\text{s-expr}\,(s))) \\
\land & \text{lr-check-result}\,(\texttt{'list}, \\
& \qquad\qquad\quad \text{s-ans}\,(\mathit{s\text{-}eval}), \\
& \qquad\qquad\quad \text{p-temp-stk}\,(\mathit{new\text{-}l}), \\
& \qquad\qquad\quad \text{p-data-segment}\,(\mathit{new\text{-}l}), \\
& \qquad\qquad\quad \text{p-temp-stk}\,(l)) \\
\land & \text{proper-p-statep}\,(\text{lr->p}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}))) \\
\land & \text{good-posp1}\,(\text{s-pos}\,(s),\ \text{s-body}\,(\text{s-prog}\,(s))) \\
\land & \text{lr-programs-properp}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table}) \\
\land & \text{s-good-statep}\,(s,\ c) \\
\land & (\text{p-psw}\,(\mathit{new\text{-}l}) = \texttt{'run}) \\
\land & (\text{p-psw}\,(\text{lr-funcall}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{new\text{-}l})) = \texttt{'run}) \\
\land & (\text{s-err-flag}\,(\mathit{s\text{-}eval}) = \texttt{'run}) \\
\land & \text{lr-check-result}\,(\mathbf{t}, \\
& \qquad\qquad\quad \text{s-ans}\,(\text{s-eval}\,(\mathbf{t}, \\
& \qquad\qquad\qquad\qquad \text{s-fun-call-state}\,(\mathit{s\text{-}eval}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{car}\,(\text{s-expr}\,(s))), \\
& \qquad\qquad\qquad\qquad c - 1)), \\
& \qquad\qquad\quad \text{p-temp-stk}\,(\text{lr-eval}\,(\mathbf{t}, \\
& \qquad\qquad\qquad\qquad\qquad \text{lr-funcall}\,(\text{s->lr1}\,(s, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad l, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{table}), \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{new\text{-}l}), \\
& \qquad\qquad\qquad\qquad\qquad c - 1)), \\
& \qquad\qquad\quad \text{p-data-segment}\,(\text{lr-eval}\,(\mathbf{t}, \\
& \qquad\qquad\qquad\qquad\qquad\quad \text{lr-funcall}\,(\text{s->lr1}\,(s, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad l, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{table}), \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{new\text{-}l}), \\
& \qquad\qquad\qquad\qquad\qquad\quad c - 1)), \\
& \qquad\qquad\quad \text{p-temp-stk}\,(\text{lr-funcall}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{new\text{-}l}))) \\
\rightarrow & \text{lr-check-result}\,(\mathbf{t}, \\
& \qquad\qquad\quad \text{s-ans}\,(\text{s-eval}\,(\mathbf{t}, \\
& \qquad\qquad\qquad\qquad \text{s-fun-call-state}\,(\mathit{s\text{-}eval}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad \text{car}\,(\text{s-expr}\,(s))), \\
& \qquad\qquad\qquad\qquad c - 1)), \\
& \qquad\qquad\quad \text{p-temp-stk}\,(\text{lr-eval}\,(\mathbf{t}, \\
& \qquad\qquad\qquad\qquad\qquad \text{lr-funcall}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}), \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{new\text{-}l}), \\
& \qquad\qquad\qquad\qquad\qquad c - 1)),
\end{array}
$$

p-data-segment (lr-eval (**t**,
lr-funcall (s->lr1 (*s*,
l,
table),
new-l),
$c - 1$)),
p-temp-stk (*l*)) **endlet**

EVENT: Disable popn-restn.

THEOREM: lr-eval-s->lr1-flag-list-opener-1
(good-posp (`'list`, s-pos (*s*), s-body (s-prog (*s*)))
$\land$ listp (s-expr-list (*s*))
$\land$ listp (s-pos (*s*))
$\land$ (s-err-flag (*s*) = `'run`))
$\rightarrow$ (lr-eval (`'list`, s->lr1 (*s*, *l*, *table*), *c*)
= lr-eval (`'list`,
lr-set-expr (lr-eval (**t**, s->lr1 (*s*, *l*, *table*), *c*),
s->lr1 (*s*, *l*, *table*),
nx (s-pos (*s*))),
c))

THEOREM: lr-eval-s->lr1-flag-list-opener-2
(good-posp (`'list`, s-pos (*s*), s-body (s-prog (*s*)))
$\land$ ($\neg$ listp (s-expr-list (*s*)))
$\land$ listp (s-pos (*s*))
$\land$ (s-err-flag (*s*) = `'run`))
$\rightarrow$ (lr-eval (`'list`, s->lr1 (*s*, *l*, *table*), *c*) = s->lr1 (*s*, *l*, *table*))

THEOREM: lr-check-result-lr-proper-heapp
lr-check-result (*flag*, *value*, *temp-stk*, *data-seg*, *orig-temp-stk*)
$\rightarrow$ lr-proper-heapp (*data-seg*)

THEOREM: lr-programs-properp-lr-set-error
lr-programs-properp (lr-set-error (*l*, *error*), *table*)
= lr-programs-properp (*l*, *table*)

THEOREM: p-psw-lr-pop-tstk-lr-eval-flag-t
(proper-p-statep (lr->p (*l*))
$\land$ good-posp1 (*pos*, program-body (p-current-program (*l*)))
$\land$ lr-programs-properp (*l*, *table*)
$\land$ (p-psw (lr-if-ok (lr-eval (**t**, lr-set-pos (*l*, *pos*), *c*))) = `'run`))
$\rightarrow$ (p-psw (lr-pop-tstk (lr-if-ok (lr-eval (**t**, lr-set-pos (*l*, *pos*), *c*))))
= `'run`)

THEOREM: lr-eval-leaves-listp-p-temp-stk-lr-set-pos
(proper-p-statep (lr->p (l))
∧ good-posp1 (pos, program-body (p-current-program (l)))
∧ lr-programs-properp (l, $table$)
∧ (p-psw (lr-eval (**t**, lr-set-pos (l, pos), c)) = **'run**))
→ listp (p-temp-stk (lr-eval (**t**, lr-set-pos (l, pos), c)))

THEOREM: p-psw-run-lr-if-ok-p-psw-run
(p-psw (lr-if-ok (l)) = **'run**) → (p-psw (l) = **'run**)

THEOREM: lr-s-similar-statesp-lr-if-ok
lr-s-similar-statesp (s-$params$, s-$temps$, lr-if-ok (l), $table$)
= lr-s-similar-statesp (s-$params$, s-$temps$, l, $table$)

THEOREM: lr-eval-s-eval-equivalence
(proper-p-statep (lr->p (s->lr1 (s, l, $table$)))
∧ lr-proper-heapp (p-data-segment (l))
∧ good-posp ($flag$, s-pos (s), s-body (s-prog (s)))
∧ lr-programs-properp (s->lr1 (s, l, $table$), $table$)
∧ lr-s-similar-statesp (s-params (s), s-temps (s), s->lr1 (s, l, $table$), $table$)
∧ s-good-statep (s, c)
∧ (p-psw (lr-eval ($flag$, s->lr1 (s, l, $table$), c)) = **'run**))
→ (lr-s-similar-statesp (s-params (s),
s-temps (s-eval ($flag$, s, c)),
lr-eval ($flag$, s->lr1 (s, l, $table$), c),
$table$)
∧ lr-check-result (**if** $flag$ = **'list then 'list**
else t endif,
s-ans (s-eval ($flag$, s, c)),
p-temp-stk (lr-eval ($flag$, s->lr1 (s, l, $table$), c)),
p-data-segment (lr-eval ($flag$,
s->lr1 (s, l, $table$),
c)),
p-temp-stk (l))
∧ (s-err-flag (s-eval ($flag$, s, c)) = **'run**))

EVENT: Disable p-psw-run-lr-if-ok-p-psw-run.

```
; ----------------------------------------------------------
; was lr-eval5.events
; ----------------------------------------------------------


;; The following define functions for each SUBR that tell how many
```

```
;; resources are used.  In the computations of the maximum control
;; stack size we break out the parts needed for formals and
;; temporaries and building a new control-stack frame.  For example in
;; CONS we have (plus 2 0 1 ...), the 2 is for building a new frame,
;; the 0 is for the formals (CONS leaves its args on the temp stack)
;; and 1 for temporaries.
```

DEFINITION:
s-apply-car-r (s) = list $(1,\ 2 + 1 + 0 + 0,\ 0,\ 0)$

DEFINITION:
s-apply-cdr-r (s) = list $(1,\ 2 + 1 + 0 + 0,\ 0,\ 0)$

```
;; CONS takes two implicit args
```

DEFINITION:
s-apply-cons-r (s) = list $(2,\ 2 + 0 + 1 + 0,\ 0,\ 1)$

DEFINITION:
s-apply-false-r (s) = list $(1,\ 2 + 0 + 0 + 0,\ 0,\ 0)$

```
;; FALSEP takes one implicit arg on stack.
```

DEFINITION:
s-apply-falsep-r (s) = list $(1,\ 2 + 0 + 0 + 0,\ 0,\ 0)$

```
;; LISTP takes an implicit arg
```

DEFINITION:
s-apply-listp-r (s) = list $(1,\ 2 + 0 + 0 + 0,\ 0,\ 0)$

```
;; NLISTP takes an implicit arg
```

DEFINITION:
s-apply-nlistp-r (s) = list $(1,\ 2 + 0 + 0 + 0,\ 0,\ 0)$

DEFINITION:
s-apply-true-r (s) = list $(1,\ 2 + 0 + 0 + 0,\ 0,\ 0)$

DEFINITION:
s-apply-truep-r (s) = list $(1,\ 2 + 0 + 0 + 0,\ 0,\ 0)$

DEFINITION:
s-apply-subr-r $(subr,\ s)$
= **case on** *subr*:
  case = *car*
  then s-apply-car-r (s)

case = *cdr*
then s-apply-cdr-r (s)
case = *cons*
then s-apply-cons-r (s)
case = *false*
then s-apply-false-r (s)
case = *falsep*
then s-apply-falsep-r (s)
case = *listp*
then s-apply-listp-r (s)
case = *nlistp*
then s-apply-nlistp-r (s)
case = *true*
then s-apply-true-r (s)
case = *truep*
then s-apply-truep-r (s)
otherwise list (0, 0, 0, 0) **endcase**

DEFINITION:
max-r (*list1*, *list2*)
= list (max (car (*list1*), car (*list2*)),
max (cadr (*list1*), cadr (*list2*)),
max (caddr (*list1*), caddr (*list2*)),
cadddr (*list1*) + cadddr (*list2*))

EVENT: Disable max-r.

DEFINITION:
s-add-temp-r (*list*, n)
= list (n + car (*list*), cadr (*list*), caddr (*list*), cadddr (*list*))

```
;; S-EVAL-R is somewhat similar to S-EVAL.  It returns a list of four
;; numbers representing.  the maximum temp stack size, maximum ctrl stack
;; size, maximum word size and number of free heap nodes respectively needed
;; to execute the compilation of the S-STATE s in Piton without getting an
;; error.
```

DEFINITION:
s-eval-r (*flag*, s, c)
= **if** s-err-flag $(s) \neq$ **'run then** list (0, 0, 0, 0)
elseif *flag* = **'list**
then if s-pos $(s) \simeq$ **nil then** list (0, 0, 0, 0)
elseif listp (s-expr-list (s))
then max-r (s-eval-r (**t**, s, c),

```
                 s-add-temp-r (s-eval-r ('list,
                                         s-set-expr (s-eval (t, s, c),
                                                     s,
                                                     nx (s-pos (s))),
                                         c),
                               1))
      else list (0, 0, 0, 0) endif
elseif c ≃ 0  then list (0, 0, 0, 0)
elseif litatom (s-expr (s))  then list (1, 0, 0, 0)
elseif s-expr (s) ≃ nil  then list (0, 0, 0, 0)
elseif car (s-expr (s)) = 'if
then let test  be  s-eval (t, s-set-pos (s, dv (s-pos (s), 1)), c)
      in
      if s-err-flag (test) = 'run
      then if s-ans (test)
            then max-r (s-add-temp-r (s-eval-r (t,
                                                 s-set-pos (s,
                                                            dv (s-pos (s),
                                                                1)),
                                                 c),
                                      1),
                        s-eval-r (t,
                                  s-set-expr (test,
                                              s,
                                              dv (s-pos (s), 2)),
                                  c))
            else max-r (s-add-temp-r (s-eval-r (t,
                                                 s-set-pos (s,
                                                            dv (s-pos (s),
                                                                1)),
                                                 c),
                                      1),
                        s-eval-r (t,
                                  s-set-expr (test,
                                              s,
                                              dv (s-pos (s), 3)),
                                  c)) endif
      else s-eval-r (t, s-set-pos (s, dv (s-pos (s), 1)), c) endif endlet
elseif car (s-expr (s)) = S-TEMP-EVAL
then s-eval-r (t, s-set-pos (s, dv (s-pos (s), 1)), c)
elseif car (s-expr (s)) = S-TEMP-TEST
then if s-temp-setp (cadr (s-expr (s)), s-temps (s))  then list (2, 0, 0, 0)
      else max-r (list (2, 0, 0, 0),
                  s-eval-r (t, s-set-pos (s, dv (s-pos (s), 1)), c)) endif
```

```
    elseif car (s-expr (s)) = S-TEMP-FETCH  then list (1, 0, 0, 0)
    elseif car (s-expr (s)) = 'quote  then list (1, 0, 0, 0)
    elseif s-err-flag (s-eval ('list, s-set-pos (s, dv (s-pos (s), 1)), c))
           ≠   'run
    then s-eval-r ('list, s-set-pos (s, dv (s-pos (s), 1)), c)
    elseif subrp (car (s-expr (s)))
    then max-r (s-eval-r ('list, s-set-pos (s, dv (s-pos (s), 1)), c),
                s-add-temp-r (s-apply-subr-r (car (s-expr (s)),
                                              s-eval ('list,
                                                      s-set-pos (s,
                                                                 dv (s-pos (s),
                                                                     1)),
                                                      c)),
                              arity (car (s-expr (s)))))
    elseif litatom (car (s-expr (s)))
    then let arg-s  be  s-eval ('list, s-set-pos (s, dv (s-pos (s), 1)), c)
         in
         let fstate  be  s-fun-call-state (arg-s, car (s-expr (s)))
         in
         let arg-r  be  s-eval-r (t, fstate, c − 1)
         in
         max-r (s-eval-r ('list,
                          s-set-pos (s, dv (s-pos (s), 1)),
                          c),
                list (car (arg-r),
                      2
                      +   length (s-params (fstate))
                      +   length (s-temps (fstate))
                      +   cadr (arg-r),
                      caddr (arg-r),
                      cadddr (arg-r))) endlet endlet endlet
    else list (0, 0, 0, 0) endif
```

DEFINITION: s-eval-temp-r (*flag*, s, c) = car (s-eval-r (*flag*, s, c))

DEFINITION: s-eval-ctrl-r (*flag*, s, c) = cadr (s-eval-r (*flag*, s, c))

DEFINITION: s-eval-ws-r (*flag*, s, c) = caddr (s-eval-r (*flag*, s, c))

DEFINITION:
s-eval-heap-r (*flag*, s, c) = cadddr (s-eval-r (*flag*, s, c))

DEFINITION:
S-MAX-SUBR-REQS
= max (log (2, LR-CONS-TAG),

max (log (2, LR-TRUE-TAG),
max (log (2, LR-CDR-OFFSET), log (2, LR-CAR-OFFSET))))

EVENT: Disable s-max-subr-reqs.

THEOREM: numberp-car-cadr-caddr-cadddr-s-apply-subr-r
(car (s-apply-subr-r (*subr*, *s*)) $\in$ **N**)
$\wedge$ (cadr (s-apply-subr-r (*subr*, *s*)) $\in$ **N**)
$\wedge$ (caddr (s-apply-subr-r (*subr*, *s*)) $\in$ **N**)
$\wedge$ (cadddr (s-apply-subr-r (*subr*, *s*)) $\in$ **N**)

EVENT: Disable s-apply-subr-r.

THEOREM: numberp-max-r
(car (max-r (*list1*, *list2*)) $\in$ **N**)
$\wedge$ (cadr (max-r (*list1*, *list2*)) $\in$ **N**)
$\wedge$ (caddr (max-r (*list1*, *list2*)) $\in$ **N**)
$\wedge$ (cadddr (max-r (*list1*, *list2*)) $\in$ **N**)

THEOREM: numberp-s-eval-temp-ctrl-ws-heap-r
(s-eval-temp-r (*flag*, *s*, *c*) $\in$ **N**)
$\wedge$ (s-eval-ctrl-r (*flag*, *s*, *c*) $\in$ **N**)
$\wedge$ (s-eval-ws-r (*flag*, *s*, *c*) $\in$ **N**)
$\wedge$ (s-eval-heap-r (*flag*, *s*, *c*) $\in$ **N**)

EVENT: Disable s-eval-temp-r.

EVENT: Disable s-eval-ctrl-r.

EVENT: Disable s-eval-ws-r.

EVENT: Disable s-eval-heap-r.

DEFINITION:
lr-count-free-nodes (*addr*, *node-list*, *data-seg*)
= **if** *addr* $\in$ *node-list*
then 1 + lr-count-free-nodes (fetch (add-addr (*addr*, LR-REF-COUNT-OFFSET),
data-seg),
delete (*addr*, *node-list*),
data-seg)
else 0 **endif**

DEFINITION:
lr-check-resourcesp (*flag*, s, l, c)
= ((p-max-temp-stk-size (l)
$\not<$ (length (p-temp-stk (l)) + s-eval-temp-r (*flag*, s, c)))
$\land$ (p-max-ctrl-stk-size (l)
$\not<$ (p-ctrl-stk-size (p-ctrl-stk (l))
\+ s-eval-ctrl-r (*flag*, s, c)))
$\land$ (p-word-size (l) $\not<$ s-eval-ws-r (*flag*, s, c))
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (l)),
lr-free-list-nodes (lr-max-node (p-data-segment (l)),
p-data-segment (l)),
p-data-segment (l))
$\not<$ s-eval-heap-r (*flag*, s, c)))

EVENT: Disable lr-check-resourcesp.

THEOREM: not-lessp-max-r-car
(car (max-r (*list1*, *list2*)) $\not<$ car (*list1*))
$\land$ (car (max-r (*list1*, *list2*)) $\not<$ car (*list2*))

THEOREM: not-lessp-max-r-cadr
(cadr (max-r (*list1*, *list2*)) $\not<$ cadr (*list1*))
$\land$ (cadr (max-r (*list1*, *list2*)) $\not<$ cadr (*list2*))

THEOREM: not-lessp-max-r-caddr
(caddr (max-r (*list1*, *list2*)) $\not<$ caddr (*list1*))
$\land$ (caddr (max-r (*list1*, *list2*)) $\not<$ caddr (*list2*))

THEOREM: not-lessp-max-r-cadddr
(cadddr (max-r (*list1*, *list2*)) $\not<$ cadddr (*list1*))
$\land$ (cadddr (max-r (*list1*, *list2*)) $\not<$ cadddr (*list2*))

THEOREM: lr-check-resourcesp-listp-s-expr-list
((s-err-flag (s) = `'run`)
$\land$ listp (s-pos (s))
$\land$ listp (s-expr-list (s))
$\land$ good-posp (`'list`, s-pos (s), s-body (s-prog (s)))
$\land$ lr-check-resourcesp (`'list`, s, l, c))
$\rightarrow$ lr-check-resourcesp (**t**, s, l, c)

THEOREM: lr-eval-preserves-lr-proper-heapp
(proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\land$ lr-proper-heapp (p-data-segment (l))
$\land$ good-posp (*flag*, s-pos (s), s-body (s-prog (s)))
$\land$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)

$\wedge$ lr-s-similar-statesp (s-params (s), s-temps (s), s->lr1 (s, l, *table*), *table*)
$\wedge$ s-good-statep (s, c)
$\wedge$ (p-psw (lr-eval (*flag*, s->lr1 (s, l, *table*), c)) = **'run**))
$\rightarrow$ lr-proper-heapp (p-data-segment (lr-eval (*flag*, s->lr1 (s, l, *table*), c)))

THEOREM: lr-eval-preserves-lr-s-similar-statesp
(proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ good-posp (*flag*, s-pos (s), s-body (s-prog (s)))
$\wedge$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\wedge$ lr-s-similar-statesp (s-params (s), s-temps (s), s->lr1 (s, l, *table*), *table*)
$\wedge$ s-good-statep (s, c)
$\wedge$ (p-psw (lr-eval (*flag*, s->lr1 (s, l, *table*), c)) = **'run**))
$\rightarrow$ lr-s-similar-statesp (s-params (s),
s-temps (s-eval (*flag*, s, c)),
lr-eval (*flag*, s->lr1 (s, l, *table*), c),
table)

THEOREM: s-eval-flag-run-flag-t-subsetp-s-collect-all-temps
(s-good-statep (s, c)
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ s-all-temps-setp (*flag*, s-expr (s), temp-alist-to-set (s-temps (s)))
$\wedge$ s-all-progs-temps-setp (s-progs (s))
$\wedge$ (s-err-flag (s-eval (*flag*, s, c)) = **'run**)
$\wedge$ s-check-temps-setp (s-temps (s))
$\wedge$ (*flag* $\neq$ **'list**))
$\rightarrow$ subsetp (s-collect-all-temps (*flag*, s-expr (s)),
temp-alist-to-set (s-temps (s-eval (*flag*, s, c))))

THEOREM: s-eval-flag-run-flag-t-s-check-temps-setp
(s-good-statep (s, c)
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ s-all-temps-setp (*flag*, s-expr (s), temp-alist-to-set (s-temps (s)))
$\wedge$ s-all-progs-temps-setp (s-progs (s))
$\wedge$ (s-err-flag (s-eval (*flag*, s, c)) = **'run**)
$\wedge$ s-check-temps-setp (s-temps (s))
$\wedge$ (*flag* $\neq$ **'list**))
$\rightarrow$ s-check-temps-setp (s-temps (s-eval (*flag*, s, c)))

THEOREM: lr-eval-preserves-length-bindings-car-p-ctrl-stk
(proper-p-statep (lr->p (l))
$\wedge$ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (lr-eval (*flag*, l, c)) = **'run**))
$\rightarrow$ (length (bindings (car (p-ctrl-stk (lr-eval (*flag*, l, c)))))
= length (bindings (car (p-ctrl-stk (l)))))

THEOREM: lr-eval-s->lr1-preserves-p-ctrl-stk-size
(proper-p-statep (lr->p (*l*))
∧ good-posp (*flag*, offset (p-pc (*l*)), program-body (p-current-program (*l*)))
∧ lr-programs-properp (*l*, *table*)
∧ (p-psw (lr-eval (*flag*, *l*, *c*)) = **'run**))
→ (p-ctrl-stk-size (p-ctrl-stk (lr-eval (*flag*, *l*, *c*)))
= p-ctrl-stk-size (p-ctrl-stk (*l*)))

THEOREM: length-p-temp-stk-lr-eval-flag-not-list
(proper-p-statep (lr->p (s->lr1 (*s*, *l*, *table*)))
∧ good-posp (*flag*, s-pos (*s*), s-body (s-prog (*s*)))
∧ s-good-statep (*s*, *c*)
∧ lr-programs-properp (s->lr1 (*s*, *l*, *table*), *table*)
∧ (p-psw (lr-eval (*flag*, s->lr1 (*s*, *l*, *table*), *c*)) = **'run**)
∧ (*flag* ≠ **'list**))
→ (length (p-temp-stk (lr-eval (*flag*, s->lr1 (*s*, *l*, *table*), *c*)))
= (1 + length (p-temp-stk (*l*))))

THEOREM: lr-eval-preserves-lr-proper-heapp-lr-set-pos
(proper-p-statep (lr->p (s->lr1 (*s*, *l*, *table*)))
∧ lr-proper-heapp (p-data-segment (*l*))
∧ good-posp (*flag*, *pos*, s-body (s-prog (*s*)))
∧ lr-programs-properp (s->lr1 (*s*, *l*, *table*), *table*)
∧ lr-s-similar-statesp (s-params (*s*), s-temps (*s*), s->lr1 (*s*, *l*, *table*), *table*)
∧ s-good-statep (*s*, *c*)
∧ (p-psw (lr-eval (*flag*, lr-set-pos (s->lr1 (*s*, *l*, *table*), *pos*), *c*)) = **'run**))
→ lr-proper-heapp (p-data-segment (lr-eval (*flag*,
lr-set-pos (s->lr1 (*s*, *l*, *table*),
pos),
c)))

THEOREM: lr-eval-preserves-lr-s-similar-statesp-lr-set-pos
(proper-p-statep (lr->p (s->lr1 (*s*, *l*, *table*)))
∧ lr-proper-heapp (p-data-segment (*l*))
∧ good-posp (*flag*, *pos*, s-body (s-prog (*s*)))
∧ lr-programs-properp (s->lr1 (*s*, *l*, *table*), *table*)
∧ lr-s-similar-statesp (s-params (*s*), s-temps (*s*), s->lr1 (*s*, *l*, *table*), *table*)
∧ s-good-statep (*s*, *c*)
∧ (p-psw (lr-eval (*flag*, lr-set-pos (s->lr1 (*s*, *l*, *table*), *pos*), *c*)) = **'run**))
→ lr-s-similar-statesp (s-params (*s*),
s-temps (s-eval (*flag*, s-set-pos (*s*, *pos*), *c*)),
lr-eval (*flag*, lr-set-pos (s->lr1 (*s*, *l*, *table*), *pos*), *c*),
table)

THEOREM: lr-eval-s-eval-flag-t-s-ans-f-lr-set-pos

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{proper-p-statep}\,(\text{lr->p}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}))))}\\
\ \wedge & \text{lr-proper-heapp}\,(\text{p-data-segment}\,(l))\\
\ \wedge & \text{good-posp1}\,(\mathit{pos},\ \text{s-body}\,(\text{s-prog}\,(s)))\\
\ \wedge & \text{lr-programs-properp}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table})\\
\ \wedge & \text{lr-s-similar-statesp}\,(\text{s-params}\,(s),\ \text{s-temps}\,(s),\ \text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table})\\
\ \wedge & \text{s-good-statep}\,(s,\ c)\\
\ \wedge & (\text{p-psw}\,(\text{lr-eval}\,(\mathbf{t},\ \text{lr-set-pos}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{pos}),\ c)) = \texttt{'run})\\
\ \wedge & (\neg\ \text{s-ans}\,(\text{s-eval}\,(\mathbf{t},\ \text{s-set-pos}\,(s,\ \mathit{pos}),\ c))))\\
\rightarrow & (\text{car}\,(\text{p-temp-stk}\,(\text{lr-eval}\,(\mathbf{t},\ \text{lr-set-pos}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{pos}),\ c)))\\
 & \quad = \ \ \textsc{lr-f-addr})
\end{array}
$$

THEOREM: lr-eval-s-eval-flag-t-s-ans-non-f-lr-set-pos

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{proper-p-statep}\,(\text{lr->p}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}))))}\\
\ \wedge & \text{lr-proper-heapp}\,(\text{p-data-segment}\,(l))\\
\ \wedge & \text{good-posp1}\,(\mathit{pos},\ \text{s-body}\,(\text{s-prog}\,(s)))\\
\ \wedge & \text{lr-programs-properp}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table})\\
\ \wedge & \text{lr-s-similar-statesp}\,(\text{s-params}\,(s),\ \text{s-temps}\,(s),\ \text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table})\\
\ \wedge & \text{s-good-statep}\,(s,\ c)\\
\ \wedge & (\text{p-psw}\,(\text{lr-eval}\,(\mathbf{t},\ \text{lr-set-pos}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{pos}),\ c)) = \texttt{'run})\\
\ \wedge & \text{s-ans}\,(\text{s-eval}\,(\mathbf{t},\ \text{s-set-pos}\,(s,\ \mathit{pos}),\ c)))\\
\rightarrow & (\text{car}\,(\text{p-temp-stk}\,(\text{lr-eval}\,(\mathbf{t},\ \text{lr-set-pos}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{pos}),\ c)))\\
 & \quad \neq \ \ \text{identity}\,(\textsc{lr-f-addr}))
\end{array}
$$

THEOREM: subsetp-not-member-both

$$((\mathit{addr} \notin \mathit{set2}) \wedge \text{subsetp}\,(\mathit{set1},\ \mathit{set2})) \rightarrow (\mathit{addr} \notin \mathit{set1})$$

THEOREM: lr-count-free-nodes-deposit-free-ptr

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{adpp}\,(\texttt{'(free-ptr . 0)},\ \mathit{data\text{-}seg}) \wedge \text{lr-node-listp}\,(\mathit{node\text{-}list},\ \mathit{data\text{-}seg}))}\\
\rightarrow & (\text{lr-count-free-nodes}\,(\mathit{addr},\\
 & \qquad\qquad \mathit{node\text{-}list},\\
 & \qquad\qquad \text{deposit}\,(\mathit{anything},\ \text{identity}\,(\textsc{lr-fp-addr}),\ \mathit{data\text{-}seg}))\\
 & \quad = \ \ \text{lr-count-free-nodes}\,(\mathit{addr},\ \mathit{node\text{-}list},\ \mathit{data\text{-}seg}))
\end{array}
$$

THEOREM: lr-count-free-nodes-deposit-non-ref-count

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-nodep}\,(\mathit{addr2},\ \mathit{data\text{-}seg})}\\
\ \wedge & (\mathit{offset} \neq \textsc{lr-ref-count-offset})\\
\ \wedge & (\mathit{offset} \in \mathbf{N})\\
\ \wedge & (\mathit{offset} < \textsc{lr-node-size})\\
\ \wedge & \text{lr-node-listp}\,(\mathit{node\text{-}list},\ \mathit{data\text{-}seg}))\\
\rightarrow & (\text{lr-count-free-nodes}\,(\mathit{addr1},\\
 & \qquad\qquad \mathit{node\text{-}list},\\
 & \qquad\qquad \text{deposit}\,(\mathit{anything},\ \text{add-addr}\,(\mathit{addr2},\ \mathit{offset}),\ \mathit{data\text{-}seg}))\\
 & \quad = \ \ \text{lr-count-free-nodes}\,(\mathit{addr1},\ \mathit{node\text{-}list},\ \mathit{data\text{-}seg}))
\end{array}
$$

THEOREM: lr-count-free-nodes-deposit-lr-nodep

(lr-nodep (*addr2*, *data-seg*) $\land$ lr-node-listp (*node-list*, *data-seg*))
$\rightarrow$ (lr-count-free-nodes (*addr1*, *node-list*, deposit (*anything*, *addr2*, *data-seg*))
$=$ lr-count-free-nodes (*addr1*, *node-list*, *data-seg*))

THEOREM: lr-count-free-nodes-delete-deposit
((*addr1* $\notin$ *node-list*)
$\land$ lr-nodep (*addr1*, *data-seg*)
$\land$ lr-node-listp (*node-list*, *data-seg*)
$\land$ (type (*ref-count*) $=$ `'nat`)
$\land$ (untag (*ref-count*) $\in \mathbf{N}$))
$\rightarrow$ (lr-count-free-nodes (*addr2*,
node-list,
deposit (*ref-count*,
add-addr (*addr1*,
identity (LR-REF-COUNT-OFFSET)),
data-seg))
$=$ lr-count-free-nodes (*addr2*, *node-list*, *data-seg*))

THEOREM: lr-count-free-nodes-max-addr-lr-free-list-nodes
lr-count-free-nodes (*max-addr*,
lr-free-list-nodes (*max-addr*, *data-seg1*),
data-seg2)
$=$ `0`

THEOREM: lr-count-lr-free-list-nodes-p-run-cons
let *dds* **be** deposit-a-list (list (identity (tag (`'nat`, `5`)),
ref-count,
any1,
any2),
fetch (identity (LR-FP-ADDR), *data-seg*),
data-seg)
in
(lr-proper-heapp (*data-seg*)
$\land$ (*max-addr* $=$ lr-max-node (*data-seg*))
$\land$ (type (*ref-count*) $=$ `'nat`)
$\land$ (untag (*ref-count*) $\in \mathbf{N}$)
$\land$ (fetch (identity (LR-FP-ADDR), *data-seg*) $\neq$ *max-addr*))
$\rightarrow$ ((1 $+$ lr-count-free-nodes (fetch (add-addr (fetch (identity (LR-FP-ADDR),
data-seg),
identity (LR-REF-COUNT-OFFSET)),
data-seg),
lr-free-list-nodes (*max-addr*, *dds*),
dds))
$=$ lr-count-free-nodes (fetch (identity (LR-FP-ADDR),
data-seg),

lr-free-list-nodes (*max-addr*,
data-seg),
data-seg)) **endlet**

THEOREM: not-p-max-node-fetch-fp-addr-not-errorp-p-run-cons
((p-psw (p (p-set-pc (lr->p (*new-l*), *pc*), p-cons-clock (p-set-pc (lr->p (*new-l*), *pc*))))
= `'run`)
$\land$ proper-p-statep (lr->p (*new-l*))
$\land$ proper-p-statep (p-set-pc (lr->p (*new-l*), *pc*))
$\land$ (*max-node* = lr-max-node (p-data-segment (*new-l*)))
$\land$ (p-psw (*new-l*) = `'run`)
$\land$ lr-programs-properp (*new-l*, *table*)
$\land$ lr-proper-heapp (p-data-segment (*new-l*))
$\land$ (unlabel (get (offset (*pc*),
program-body (assoc (area-name (*pc*),
p-prog-segment (lr->p (*new-l*))))))
= `'(call cons)`))
$\rightarrow$ (fetch (identity (LR-FP-ADDR), p-data-segment (*new-l*)) $\neq$ *max-node*)

THEOREM: get-comp-body-lr-compile-programs
(good-posp1 (s-pos (*s*), s-body (s-prog (*s*)))
$\land$ lr-programs-properp (s->lr1 (*s*, *l*, *table*), *table*)
$\land$ (car (s-expr (*s*)) $\neq$ `'if`)
$\land$ (car (s-expr (*s*)) $\neq$ S-TEMP-EVAL)
$\land$ (car (s-expr (*s*)) $\neq$ S-TEMP-TEST)
$\land$ (car (s-expr (*s*)) $\neq$ S-TEMP-FETCH)
$\land$ (car (s-expr (*s*)) $\neq$ `'quote`)
$\land$ listp (s-expr (*s*)))
$\rightarrow$ (get (offset (lr-return-pc (s->lr1 (*s*, *l*, *table*))),
program-body (assoc (s-pname (*s*),
comp-programs (lr-compile-programs (s-progs (*s*),
table)))))
= list (`'dl`,
lr-make-label (offset (lr-return-pc (s->lr1 (*s*, *l*, *table*)))),
nil,
if definedp (car (s-expr (*s*)), P-RUNTIME-SUPPORT-PROGRAMS)
then list (`'call`, car (s-expr (*s*)))
else list (`'call`, user-fname (car (s-expr (*s*)))) **endif**))

THEOREM: lr-count-lr-free-list-nodes-p-run-subr
let *p* **be** p-set-pc (lr->p (lr-eval (`'list`,
lr-set-pos (s->lr1 (*s*, *l*, *table*), *pos*),
c)),
lr-return-pc (s->lr1 (*s*, *l*, *table*))),
new-l **be** lr-eval (`'list`, lr-set-pos (s->lr1 (*s*, *l*, *table*), *pos*), *c*)

in
(listp (s-expr (s))
 ∧ (car (s-expr (s)) ≠ `'if`)
 ∧ subrp (car (s-expr (s)))
 ∧ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
 ∧ good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
 ∧ s-good-statep (s, c)
 ∧ (p-psw (p-run-subr (car (s-expr (s)), p)) = `'run`)
 ∧ (p-psw (*new-l*) = `'run`)
 ∧ (*pos* = dv (s-pos (s), `1`))
 ∧ lr-proper-heapp (p-data-segment (l))
 ∧ lr-s-similar-statesp (s-params (s),
  s-temps (s),
  s->lr1 (s, l, *table*),
  table))
→ (lr-count-free-nodes (fetch (identity (LR-FP-ADDR),
  p-data-segment (*new-l*)),
  lr-free-list-nodes (lr-max-node (p-data-segment (l)),
  p-data-segment (*new-l*)),
  p-data-segment (*new-l*))
 = (lr-count-free-nodes (fetch (identity (LR-FP-ADDR),
  p-data-segment (p-run-subr (car (s-expr (s)),
  p))),
  lr-free-list-nodes (lr-max-node (p-data-segment (l)),
  p-data-segment (p-run-subr (car (s-expr (s)),
  p))),
  p-data-segment (p-run-subr (car (s-expr (s)),
  p)))
 + cadddr (s-apply-subr-r (car (s-expr (s)),
  s-eval (`'list`,
  s-set-pos (s, *pos*),
  c))))) **endlet**

THEOREM: lr-count-lr-free-list-nodes-lr-apply-subr
let *new-l* **be** lr-eval (`'list`, lr-set-pos (s->lr1 (s, l, *table*), *pos*), c)
in
(listp (s-expr (s))
 ∧ (car (s-expr (s)) ≠ `'if`)
 ∧ subrp (car (s-expr (s)))
 ∧ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
 ∧ good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
 ∧ (p-psw (*new-l*) = `'run`)

$$
\begin{array}{ll}
\land & (\text{p-psw}\,(\text{lr-apply-subr}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{new\text{-}l})) = \texttt{'run})\\
\land & \text{s-good-statep}\,(s,\ c)\\
\land & \text{lr-proper-heapp}\,(\text{p-data-segment}\,(l))\\
\land & \text{lr-s-similar-statesp}\,(\text{s-params}\,(s),\\
& \qquad \text{s-temps}\,(s),\\
& \qquad \text{s->lr1}\,(s,\ l,\ \mathit{table}),\\
& \qquad \mathit{table})\\
\land & (\mathit{pos} = \text{dv}\,(\text{s-pos}\,(s),\ \texttt{1}))\\
\land & (\mathit{max\text{-}addr} = \text{lr-max-node}\,(\text{p-data-segment}\,(l)))\\
\land & (\mathit{s\text{-}eval\text{-}size} = \text{s-eval-heap-r}\,(\texttt{'list},\ \text{s-set-pos}\,(s,\ \mathit{pos}),\ c))\\
\land & ((\text{lr-count-free-nodes}\,(\text{fetch}\,(\textsc{lr-fp-addr},\ \text{p-data-segment}\,(\mathit{new\text{-}l})),\\
& \qquad \text{lr-free-list-nodes}\,(\mathit{max\text{-}addr},\\
& \qquad\qquad \text{p-data-segment}\,(\mathit{new\text{-}l})),\\
& \qquad \text{p-data-segment}\,(\mathit{new\text{-}l}))\\
& \quad + \quad \mathit{s\text{-}eval\text{-}size})\\
& \quad = \quad \text{lr-count-free-nodes}\,(\text{fetch}\,(\textsc{lr-fp-addr},\\
& \qquad\qquad \text{p-data-segment}\,(l)),\\
& \qquad \text{lr-free-list-nodes}\,(\mathit{max\text{-}addr},\\
& \qquad\qquad \text{p-data-segment}\,(l)),\\
& \qquad \text{p-data-segment}\,(l))))\\
\to & ((\mathit{s\text{-}eval\text{-}size}\\
& \quad + \quad \text{cadddr}\,(\text{s-apply-subr-r}\,(\text{car}\,(\text{s-expr}\,(s)),\\
& \qquad\qquad \text{s-eval}\,(\texttt{'list},\\
& \qquad\qquad\qquad \text{s-set-pos}\,(s,\ \mathit{pos}),\\
& \qquad\qquad\qquad c)))\\
& \quad + \quad \text{lr-count-free-nodes}\,(\text{fetch}\,(\text{identity}\,(\textsc{lr-fp-addr}),\\
& \qquad\qquad \text{p-data-segment}\,(\text{lr-apply-subr}\,(\text{s->lr1}\,(s,\\
& \qquad\qquad\qquad l,\\
& \qquad\qquad\qquad \mathit{table}),\\
& \qquad\qquad\qquad \mathit{new\text{-}l}))),\\
& \qquad \text{lr-free-list-nodes}\,(\mathit{max\text{-}addr},\\
& \qquad\qquad \text{p-data-segment}\,(\text{lr-apply-subr}\,(\text{s->lr1}\,(s,\\
& \qquad\qquad\qquad l,\\
& \qquad\qquad\qquad \mathit{table}),\\
& \qquad\qquad\qquad \mathit{new\text{-}l}))),\\
& \qquad \text{p-data-segment}\,(\text{lr-apply-subr}\,(\text{s->lr1}\,(s,\\
& \qquad\qquad l,\\
& \qquad\qquad \mathit{table}),\\
& \qquad\qquad \mathit{new\text{-}l}))))\\
& \quad = \quad \text{lr-count-free-nodes}\,(\text{fetch}\,(\text{identity}\,(\textsc{lr-fp-addr}),\\
& \qquad\qquad \text{p-data-segment}\,(l)),\\
& \qquad \text{lr-free-list-nodes}\,(\mathit{max\text{-}addr},\\
& \qquad\qquad \text{p-data-segment}\,(l)),\\
& \qquad \text{p-data-segment}\,(l)))\ \textbf{endlet}
\end{array}
$$

THEOREM: lr-eval-s-eval-equivalence-lr-check-result-flag-list

let *lr-eval* **be** lr-eval ('`list`,
    lr-set-pos (s->lr1 (s, l, *table*), dv (s-pos (s), `1`)),
    c)
in
(proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
 $\land$ lr-proper-heapp (p-data-segment (l))
 $\land$ good-posp1 (s-pos (s), s-body (s-prog (s)))
 $\land$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
 $\land$ lr-s-similar-statesp (s-params (s),
    s-temps (s),
    s->lr1 (s, l, *table*),
    table)
 $\land$ s-good-statep (s, c)
 $\land$ (p-psw (*lr-eval*) = '`run`)
 $\land$ listp (s-expr (s))
 $\land$ (car (s-expr (s)) $\neq$ '`if`)
 $\land$ (car (s-expr (s)) $\neq$ S-TEMP-EVAL)
 $\land$ (car (s-expr (s)) $\neq$ S-TEMP-TEST)
 $\land$ (car (s-expr (s)) $\neq$ S-TEMP-FETCH)
 $\land$ (car (s-expr (s)) $\neq$ '`quote`))
$\rightarrow$ lr-check-result ('`list`,
    s-ans (s-eval ('`list`,
        s-set-pos (s, dv (s-pos (s), `1`)),
        c)),
    p-temp-stk (*lr-eval*),
    p-data-segment (*lr-eval*),
    p-temp-stk (l)) **endlet**

THEOREM: cadddr-max-r

cadddr (max-r (*list1*, *list2*)) = (cadddr (*list1*) + cadddr (*list2*))

THEOREM: lr-eval-s-eval-heap-r-lr-count-lr-free-list-nodes

(proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
 $\land$ lr-proper-heapp (p-data-segment (l))
 $\land$ good-posp (*flag*, s-pos (s), s-body (s-prog (s)))
 $\land$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
 $\land$ lr-s-similar-statesp (s-params (s), s-temps (s), s->lr1 (s, l, *table*), *table*)
 $\land$ s-good-statep (s, c)
 $\land$ (p-psw (lr-eval (*flag*, s->lr1 (s, l, *table*), c)) = '`run`)
 $\land$ (s-err-flag (s-eval (*flag*, s, c)) = '`run`))
$\rightarrow$ ((lr-count-free-nodes (fetch (LR-FP-ADDR,
    p-data-segment (lr-eval (*flag*,
        s->lr1 (s, l, *table*),

c))),
lr-free-list-nodes (lr-max-node (p-data-segment (*l*)),
p-data-segment (lr-eval (*flag*,
s->lr1 (*s*,
l,
table),
c))),
p-data-segment (lr-eval (*flag*, s->lr1 (*s*, *l*, *table*), *c*)))
+ s-eval-heap-r (*flag*, *s*, *c*))
= lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (*l*)),
lr-free-list-nodes (lr-max-node (p-data-segment (*l*)),
p-data-segment (*l*)),
p-data-segment (*l*)))

THEOREM: lr-check-resourcesp-list-set-expr-nx
(listp (s-pos (*s*))
∧ listp (s-expr-list (*s*))
∧ good-posp (**'list**, s-pos (*s*), s-body (s-prog (*s*)))
∧ proper-p-statep (lr->p (s->lr1 (*s*, *l*, *table*)))
∧ lr-proper-heapp (p-data-segment (*l*))
∧ lr-programs-properp (s->lr1 (*s*, *l*, *table*), *table*)
∧ lr-s-similar-statesp (s-params (*s*), s-temps (*s*), s->lr1 (*s*, *l*, *table*), *table*)
∧ s-good-statep (*s*, *c*)
∧ (p-psw (lr-eval (**t**, s->lr1 (*s*, *l*, *table*), *c*)) = **'run**)
∧ (s-err-flag (s-eval (**t**, *s*, *c*)) = **'run**)
∧ lr-check-resourcesp (**'list**, *s*, *l*, *c*))
→ lr-check-resourcesp (**'list**,
s-set-expr (s-eval (**t**, *s*, *c*), *s*, nx (s-pos (*s*))),
lr-eval (**t**, s->lr1 (*s*, *l*, *table*), *c*),
c)

THEOREM: lr-check-resourcesp-lr-push-tstk-flag-run
(lr-check-resourcesp (*flag*, *s*, *l*, *c*)
∧ (*flag* ≠ **'list**)
∧ litatom (s-expr (*s*))
∧ (*c* ≄ **0**)
∧ (s-err-flag (*s*) = **'run**))
→ (p-psw (lr-push-tstk (s->lr1 (*s*, *l*, *table*),
cdr (assoc (s-expr (*s*), bindings (car (p-ctrl-stk (*l*)))))))
= **'run**)

THEOREM: lr-check-resourcesp-s-set-pos-if-cadr
(lr-check-resourcesp (*flag*, *s*, *l*, *c*)
∧ s-good-statep (*s*, *c*)
∧ (*flag* ≠ **'list**)

$\wedge$ $(c \not\simeq \texttt{0})$
$\wedge$ listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $=$ **'if**))
$\rightarrow$ lr-check-resourcesp (**t**, s-set-pos (s, dv (s-pos (s), 1)), l, c)

THEOREM: s-eval-subsetp-s-collect-temp-alist-s-set-pos-if
(listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $=$ **'if**)
$\wedge$ s-good-statep (s, c)
$\wedge$ good-posp1 (dv (s-pos (s), 1), s-body (s-prog (s)))
$\wedge$ s-check-temps-setp (s-temps (s))
$\wedge$ s-all-temps-setp (**t**, cadr (s-expr (s)), temp-alist-to-set (s-temps (s)))
$\wedge$ s-all-progs-temps-setp (s-progs (s))
$\wedge$ (s-err-flag (s-eval (**t**, s-set-pos (s, dv (s-pos (s), 1)), c)) $=$ **'run**))
$\rightarrow$ subsetp (s-collect-all-temps (**t**, cadr (s-expr (s))),
temp-alist-to-set (s-temps (s-eval (**t**,
s-set-pos (s, dv (s-pos (s), 1)),
c))))

THEOREM: length-p-temp-stk-lr-pop-tstk-lr-eval-flag-t
(proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\wedge$ good-posp1 (*pos*, s-body (s-prog (s)))
$\wedge$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\wedge$ s-good-statep (s, c)
$\wedge$ (p-psw (lr-if-ok (lr-eval (**t**, lr-set-pos (s->lr1 (s, l, *table*), *pos*), c)))
$=$ **'run**))
$\rightarrow$ (length (p-temp-stk (lr-pop-tstk (lr-if-ok (lr-eval (**t**,
lr-set-pos (s->lr1 (s,
l,
table),
pos),
c)))))
$=$ length (p-temp-stk (l)))

THEOREM: lr-eval-s->lr1-preserves-p-ctrl-stk-size-lr-set-pos
(proper-p-statep (lr->p (l))
$\wedge$ good-posp (*flag*, *pos*, program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (lr-eval (*flag*, lr-set-pos (l, *pos*), c)) $=$ **'run**))
$\rightarrow$ (p-ctrl-stk-size (p-ctrl-stk (lr-eval (*flag*, lr-set-pos (l, *pos*), c)))
$=$ p-ctrl-stk-size (p-ctrl-stk (l)))

THEOREM: lr-check-resourcesp-lr-pop-tstk-lr-eval-1
let *lr-eval* **be** lr-if-ok (lr-eval (**t**, lr-set-pos (s->lr1 (s, l, *table*), *pos*), c))
in

$((c \not\simeq \mathtt{0})$
$\land$ listp (s-expr (s))
$\land$ (car (s-expr (s)) $=$ **'if**)
$\land$ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\land$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\land$ lr-proper-heapp (p-data-segment (l))
$\land$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\land$ lr-s-similar-statesp (s-params (s),
s-temps (s),
s->lr1 (s, l, *table*),
table)
$\land$ s-good-statep (s, c)
$\land$ (p-psw (*lr-eval*) $=$ **'run**)
$\land$ (pos $=$ dv (s-pos (s), $\mathtt{1}$))
$\land$ (s-err-flag (s-eval (**t**, s-set-pos (s, pos), c)) $=$ **'run**)
$\land$ s-ans (s-eval (**t**, s-set-pos (s, dv (s-pos (s), $\mathtt{1}$)), c))
$\land$ lr-check-resourcesp (*flag*, s, l, c)
$\land$ (*flag* $\neq$ **'list**))
$\rightarrow$ lr-check-resourcesp (**t**,
s-set-expr (s-eval (**t**, s-set-pos (s, pos), c),
s,
dv (s-pos (s), $\mathtt{2}$)),
lr-pop-tstk (*lr-eval*),
c) **endlet**

THEOREM: lr-check-resourcesp-lr-pop-tstk-lr-eval-2
let *lr-eval* **be** lr-if-ok (lr-eval (**t**, lr-set-pos (s->lr1 (s, l, *table*), pos), c))
in
$((c \not\simeq \mathtt{0})$
$\land$ listp (s-expr (s))
$\land$ (car (s-expr (s)) $=$ **'if**)
$\land$ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\land$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\land$ lr-proper-heapp (p-data-segment (l))
$\land$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\land$ lr-s-similar-statesp (s-params (s),
s-temps (s),
s->lr1 (s, l, *table*),
table)
$\land$ s-good-statep (s, c)
$\land$ (p-psw (*lr-eval*) $=$ **'run**)
$\land$ (pos $=$ dv (s-pos (s), $\mathtt{1}$))
$\land$ (s-err-flag (s-eval (**t**, s-set-pos (s, pos), c)) $=$ **'run**)
$\land$ ($\neg$ s-ans (s-eval (**t**, s-set-pos (s, dv (s-pos (s), $\mathtt{1}$)), c)))

$\wedge$ lr-check-resourcesp (*flag*, s, l, c)
$\wedge$ (*flag* $\neq$ `'list`))
$\rightarrow$ lr-check-resourcesp (**t**,
s-set-expr (s-eval (**t**, s-set-pos (s, *pos*), c),
s,
dv (s-pos (s), 3)),
lr-pop-tstk (*lr-eval*),
c) **endlet**

THEOREM: lr-check-resourcesp-s-temp-eval
(($c \not\simeq 0$)
$\wedge$ listp (s-expr (s))
$\wedge$ (car (s-expr (s)) = S-TEMP-EVAL)
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ (*pos* = dv (s-pos (s), 1))
$\wedge$ lr-check-resourcesp (*flag*, s, l, c)
$\wedge$ (*flag* $\neq$ `'list`))
$\rightarrow$ lr-check-resourcesp (**t**, s-set-pos (s, *pos*), l, c)

THEOREM: lr-check-resourcesp-s-temp-test
(($c \not\simeq 0$)
$\wedge$ listp (s-expr (s))
$\wedge$ (car (s-expr (s)) = S-TEMP-TEST)
$\wedge$ ($\neg$ s-temp-setp (cadr (s-expr (s)), s-temps (s)))
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ (*pos* = dv (s-pos (s), 1))
$\wedge$ lr-check-resourcesp (*flag*, s, l, c)
$\wedge$ (*flag* $\neq$ `'list`))
$\rightarrow$ lr-check-resourcesp (**t**, s-set-pos (s, *pos*), l, c)

THEOREM: lr-do-temp-fetch-lr-check-resourcesp-temp-test
(lr-check-resourcesp (*flag*, s, l, c)
$\wedge$ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\wedge$ listp (s-expr (s))
$\wedge$ ((car (s-expr (s)) = S-TEMP-TEST)
$\vee$ (car (s-expr (s)) = S-TEMP-FETCH))
$\wedge$ s-good-statep (s, c)
$\wedge$ lr-s-similar-statesp (s-params (s), s-temps (s), s->lr1 (s, l, *table*), *table*)
$\wedge$ ($c \not\simeq 0$)
$\wedge$ s-temp-setp (cadr (s-expr (s)), s-temps (s))
$\wedge$ (*flag* $\neq$ `'list`))
$\rightarrow$ (p-psw (lr-do-temp-fetch (s->lr1 (s, l, *table*))) = `'run`)

THEOREM: lr-push-tstk-lr-check-resourcesp-quote

$$
\begin{array}{ll}
 & (\text{lr-check-resourcesp}\,(\mathit{flag},\ s,\ l,\ c) \\
\land & \text{good-posp1}\,(\text{s-pos}\,(s),\ \text{s-body}\,(\text{s-prog}\,(s))) \\
\land & \text{listp}\,(\text{s-expr}\,(s)) \\
\land & (\text{car}\,(\text{s-expr}\,(s)) = \texttt{'quote}) \\
\land & \text{s-good-statep}\,(s,\ c) \\
\land & (c \not\simeq \texttt{0}) \\
\land & (\mathit{flag} \neq \texttt{'list})) \\
\to & (\text{p-psw}\,(\text{lr-push-tstk}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}), \\
 & \qquad\qquad\qquad \text{cadr}\,(\text{lr-expr}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}))))) \\
 & \quad = \ \texttt{'run})
\end{array}
$$

THEOREM: lr-check-resourcesp-funcall

$$
\begin{array}{ll}
 & ((c \not\simeq \texttt{0}) \\
\land & \text{listp}\,(\text{s-expr}\,(s)) \\
\land & (\text{car}\,(\text{s-expr}\,(s)) \neq \texttt{'if}) \\
\land & (\text{car}\,(\text{s-expr}\,(s)) \neq \text{S-TEMP-EVAL}) \\
\land & (\text{car}\,(\text{s-expr}\,(s)) \neq \text{S-TEMP-TEST}) \\
\land & (\text{car}\,(\text{s-expr}\,(s)) \neq \text{S-TEMP-FETCH}) \\
\land & (\text{car}\,(\text{s-expr}\,(s)) \neq \texttt{'quote}) \\
\land & (\text{s-err-flag}\,(\text{s-eval}\,(\texttt{'list},\ \text{s-set-pos}\,(s,\ \text{dv}\,(\text{s-pos}\,(s),\ \texttt{1})),\ c)) = \texttt{'run}) \\
\land & \text{proper-p-statep}\,(\text{lr->p}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}))) \\
\land & \text{lr-proper-heapp}\,(\text{p-data-segment}\,(l)) \\
\land & \text{good-posp1}\,(\text{s-pos}\,(s),\ \text{s-body}\,(\text{s-prog}\,(s))) \\
\land & \text{lr-programs-properp}\,(\text{s->lr1}\,(s,\ l,\ \mathit{table}),\ \mathit{table}) \\
\land & \text{s-good-statep}\,(s,\ c) \\
\land & \text{lr-check-resourcesp}\,(\mathit{flag},\ s,\ l,\ c) \\
\land & (\mathit{flag} \neq \texttt{'list})) \\
\to & \text{lr-check-resourcesp}\,(\texttt{'list},\ \text{s-set-pos}\,(s,\ \text{dv}\,(\text{s-pos}\,(s),\ \texttt{1})),\ l,\ c)
\end{array}
$$

THEOREM: numberp-s-eval-temp-ctrl-ws-heap-r-opened

$$
\begin{array}{ll}
 & (\text{car}\,(\text{s-eval-r}\,(\mathit{flag},\ s,\ c)) \in \mathbf{N}) \\
\land & (\text{cadr}\,(\text{s-eval-r}\,(\mathit{flag},\ s,\ c)) \in \mathbf{N}) \\
\land & (\text{caddr}\,(\text{s-eval-r}\,(\mathit{flag},\ s,\ c)) \in \mathbf{N}) \\
\land & (\text{cadddr}\,(\text{s-eval-r}\,(\mathit{flag},\ s,\ c)) \in \mathbf{N})
\end{array}
$$

THEOREM: lessp-1-not-zerop-exp

$$((m \not\simeq \texttt{0}) \land (\texttt{1} < n)) \to (\texttt{1} < \text{exp}\,(n,\ m))$$

THEOREM: lessp-1-not-zerop-log

$$((\texttt{1} < c) \land (n \in \mathbf{N})) \to ((\text{log}\,(c,\ n) < \texttt{1}) = (n < \texttt{1}))$$

DEFINITION:

$$
\begin{array}{l}
\text{induct-hint-18}\,(c,\ n,\ m) \\
= \quad \textbf{if } c < 2 \ \textbf{ then t}
\end{array}
$$

elseif $n \simeq 0$ **then t**
elseif $m \simeq 0$ **then t**
else induct-hint-18 (c, $n \div c$, $m - 1$) **endif**

THEOREM: times-quotient-lessp-fact-1
$((c \not\simeq 0) \wedge (n \in \mathbf{N}) \wedge (m \in \mathbf{N}))$
$\rightarrow \quad ((n < (c * m)) = ((n \div c) < m))$

THEOREM: exp-log-lessp-fact-1
$((1 < c) \wedge (n \in \mathbf{N}) \wedge (m \in \mathbf{N}))$
$\rightarrow \quad ((n < \exp(c, m)) = (\log(c, n) < (1 + m)))$

EVENT: Disable times-quotient-lessp-fact-1.

THEOREM: adpp-untag-add-addr-offset-car
(lr-good-pointerp (*addr*, *data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ (untag (fetch (*addr*, *data-seg*)) = LR-CONS-TAG)
$\wedge$ lr-proper-heapp (*data-seg*))
$\rightarrow$ adpp (untag (add-addr (*addr*, identity (LR-CAR-OFFSET))), *data-seg*)

THEOREM: adpp-untag-add-addr-offset-cdr
(lr-good-pointerp (*addr*, *data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ (untag (fetch (*addr*, *data-seg*)) = LR-CONS-TAG)
$\wedge$ lr-proper-heapp (*data-seg*))
$\rightarrow$ adpp (untag (add-addr (*addr*, identity (LR-CDR-OFFSET))), *data-seg*)

THEOREM: exp-log-2-lessp-add1-fact-1
$((1 + n) < \exp(2, m)) = (\log(2, 1 + n) < (1 + m))$

```
;; The P-TEST-BOOL-AND-JUMP cause a lot of case splits after being opened
;; and the result rewritten with P-OBJECTP-TYPE, so we prove two simple
;; lemmas and disable it, this should hopefully speed up the proof.
```

THEOREM: p-test-bool-and-jump-okp-t-cons-bool-t
p-test-bool-and-jump-okp (list (*ins-name*, `'t`, *label*),
p-state (*pc*,
ctrl-stk,
cons (`'(bool t)`, *temp-stk*),
prog-seg,
data-seg,
max-ctrl,
max-temp,
word-size,
psw))
= **t**

THEOREM: p-test-bool-and-jump-okp-f-cons-bool-t
p-test-bool-and-jump-okp (list (*ins-name*, `'f`, *label*),
p-state (*pc*,
ctrl-stk,
cons (`'(bool t)`, *temp-stk*),
prog-seg,
data-seg,
max-ctrl,
max-temp,
word-size,
psw))
= **t**

THEOREM: p-test-bool-and-jump-okp-t-cons-bool-f
p-test-bool-and-jump-okp (list (*ins-name*, `'t`, *label*),
p-state (*pc*,
ctrl-stk,
cons (`'(bool f)`, *temp-stk*),
prog-seg,
data-seg,
max-ctrl,
max-temp,
word-size,
psw))
= **t**

THEOREM: p-test-bool-and-jump-okp-f-cons-bool-f
p-test-bool-and-jump-okp (list (*ins-name*, `'f`, *label*),
p-state (*pc*,
ctrl-stk,
cons (`'(bool f)`, *temp-stk*),
prog-seg,
data-seg,
max-ctrl,
max-temp,
word-size,
psw))
= **t**

EVENT: Disable p-test-bool-and-jump-okp.

THEOREM: p-psw-run-run-car-lr-check-resourcesp
(lr-check-result (`'list`,
s-ans (*new-s*),

```
                    p-temp-stk (new-l),
                    p-data-segment (new-l),
                    orig-temp-stk)
  ∧  proper-p-statep (lr->p (new-l))
  ∧  (p-psw (new-l) = 'run)
  ∧  lr-programs-properp (new-l, table)
  ∧  (unlabel (get (offset (pc),
                      program-body (assoc (area-name (pc),
                                           p-prog-segment (lr->p (new-l))))))
       =   '(call car))
  ∧  (p-max-temp-stk-size (new-l)
       ≮   (length (p-temp-stk (new-l))
              +   car (s-apply-subr-r ('car, new-s))))
  ∧  (p-max-ctrl-stk-size (new-l)
       ≮   (p-ctrl-stk-size (p-ctrl-stk (new-l))
              +   cadr (s-apply-subr-r ('car, new-s))))
  ∧  (p-word-size (new-l)
       ≮   max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r ('car, new-s))))
  ∧  (lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (new-l)),
                            lr-free-list-nodes (lr-max-node (p-data-segment (new-l)),
                                                p-data-segment (new-l)),
                            p-data-segment (new-l))
       ≮   cadddr (s-apply-subr-r ('car, new-s)))
  ∧  (length (p-temp-stk (new-l)) ≮ arity ('car))
  ∧  (length (s-ans (new-s)) = arity ('car)))
→   (p-psw (p (p-set-pc (lr->p (new-l), pc),
                p-car-clock (p-set-pc (lr->p (new-l), pc))))
       =   'run)
```

```
THEOREM: p-psw-run-run-cdr-lr-check-resourcesp
(lr-check-result ('list,
                    s-ans (new-s),
                    p-temp-stk (new-l),
                    p-data-segment (new-l),
                    orig-temp-stk)
  ∧  proper-p-statep (lr->p (new-l))
  ∧  (p-psw (new-l) = 'run)
  ∧  lr-programs-properp (new-l, table)
  ∧  (unlabel (get (offset (pc),
                      program-body (assoc (area-name (pc),
                                           p-prog-segment (lr->p (new-l))))))
       =   '(call cdr))
  ∧  (p-max-temp-stk-size (new-l)
       ≮   (length (p-temp-stk (new-l))
```

$$
\begin{array}{lll}
 & & \quad + \quad \text{car}\,(\text{s-apply-subr-r}\,(\texttt{'cdr},\ \textit{new-s})))) \\
\wedge & (\text{p-max-ctrl-stk-size}\,(\textit{new-l}) & \\
 & \quad \not< \quad (\text{p-ctrl-stk-size}\,(\text{p-ctrl-stk}\,(\textit{new-l})) & \\
 & & \quad + \quad \text{cadr}\,(\text{s-apply-subr-r}\,(\texttt{'cdr},\ \textit{new-s})))) \\
\wedge & (\text{p-word-size}\,(\textit{new-l}) & \\
 & \quad \not< \quad \max\,(\text{S-MAX-SUBR-REQS},\ \text{caddr}\,(\text{s-apply-subr-r}\,(\texttt{'cdr},\ \textit{new-s})))) & \\
\wedge & (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \text{p-data-segment}\,(\textit{new-l})), & \\
 & \qquad \text{lr-free-list-nodes}\,(\text{lr-max-node}\,(\text{p-data-segment}\,(\textit{new-l})), & \\
 & \qquad\qquad \text{p-data-segment}\,(\textit{new-l})), & \\
 & \qquad \text{p-data-segment}\,(\textit{new-l})) & \\
 & \quad \not< \quad \text{cadddr}\,(\text{s-apply-subr-r}\,(\texttt{'cdr},\ \textit{new-s}))) & \\
\wedge & (\text{length}\,(\text{p-temp-stk}\,(\textit{new-l})) \not< \text{arity}\,(\texttt{'cdr})) & \\
\wedge & (\text{length}\,(\text{s-ans}\,(\textit{new-s})) = \text{arity}\,(\texttt{'cdr}))) & \\
\rightarrow & (\text{p-psw}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(\textit{new-l}),\ \textit{pc}), & \\
 & \qquad \text{p-cdr-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(\textit{new-l}),\ \textit{pc})))) & \\
 & \quad = \quad \texttt{'run}) &
\end{array}
$$

THEOREM: lessp-plus-remainder-0-fact

$$
\begin{array}{ll}
 & (((\textit{offset1}\ \textbf{mod}\ \textit{max}) = 0) \\
\wedge & ((\textit{offset2}\ \textbf{mod}\ \textit{max}) = 0) \\
\wedge & (n < \textit{max}) \\
\wedge & (\textit{offset1} \in \mathbf{N}) \\
\wedge & (\textit{offset2} \in \mathbf{N})) \\
\rightarrow & (((n + \textit{offset1}) < \textit{offset2}) = (\textit{offset1} < \textit{offset2}))
\end{array}
$$

THEOREM: lr-boundary-nodep-lessp-plus-fact

$$
\begin{array}{ll}
 & (\text{lr-boundary-nodep}\,(\textit{addr1}) \\
\wedge & \text{lr-boundary-nodep}\,(\textit{addr2}) \\
\wedge & (n < \text{LR-NODE-SIZE}) \\
\wedge & (\text{offset}\,(\textit{addr1}) \in \mathbf{N}) \\
\wedge & (\text{offset}\,(\textit{addr2}) \in \mathbf{N})) \\
\rightarrow & (((n + \text{offset}\,(\textit{addr1})) < \text{offset}\,(\textit{addr2})) \\
 & \quad = \quad (\text{offset}\,(\textit{addr1}) < \text{offset}\,(\textit{addr2})))
\end{array}
$$

THEOREM: adpp-untag-add-addr-lr-nodep-not-max-addr

$$
\begin{array}{ll}
 & (\text{adpp}\,(\text{untag}\,(\textit{addr}),\ \textit{data-seg}) \\
\wedge & \text{lr-boundary-nodep}\,(\textit{addr}) \\
\wedge & (\text{area-name}\,(\textit{addr}) = \texttt{'heap}) \\
\wedge & \text{definedp}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}) \\
\wedge & \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\textit{data-seg})) \\
\wedge & (\text{offset}\,(\textit{addr}) < \text{offset}\,(\text{lr-max-node}\,(\textit{data-seg}))) \\
\wedge & (n < \text{LR-NODE-SIZE})) \\
\rightarrow & \text{adpp}\,(\text{untag}\,(\text{add-addr}\,(\textit{addr},\ n)),\ \textit{data-seg})
\end{array}
$$

THEOREM: adpp-untag-add-addr-offset-on-free-listp

(lr-proper-p-areasp (*data-seg*)
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ `1`)
$\wedge$ lr-proper-heapp (*data-seg*)
$\wedge$ (n $<$ LR-NODE-SIZE))
$\rightarrow$ adpp (untag (add-addr (fetch (identity (LR-FP-ADDR), *data-seg*), n)), *data-seg*)

THEOREM: p-psw-run-run-cons-lr-check-resourcesp
(lr-check-result (**'list**,
s-ans (*new-s*),
p-temp-stk (*new-l*),
p-data-segment (*new-l*),
orig-temp-stk)
$\wedge$ proper-p-statep (lr->p (*new-l*))
$\wedge$ (p-psw (*new-l*) $=$ **'run**)
$\wedge$ lr-programs-properp (*new-l*, *table*)
$\wedge$ (unlabel (get (offset (*pc*),
program-body (assoc (area-name (*pc*),
p-prog-segment (lr->p (*new-l*))))))
$=$ **'(call cons)**)
$\wedge$ (p-max-temp-stk-size (*new-l*)
$\not<$ (length (p-temp-stk (*new-l*))
$+$ car (s-apply-subr-r (**'cons**, *new-s*))))
$\wedge$ (p-max-ctrl-stk-size (*new-l*)
$\not<$ (p-ctrl-stk-size (p-ctrl-stk (*new-l*))
$+$ cadr (s-apply-subr-r (**'cons**, *new-s*))))
$\wedge$ (p-word-size (*new-l*)
$\not<$ max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r (**'cons**, *new-s*))))
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (*new-l*)),
lr-free-list-nodes (lr-max-node (p-data-segment (*new-l*)),
p-data-segment (*new-l*)),
p-data-segment (*new-l*))
$\not<$ cadddr (s-apply-subr-r (**'cons**, *new-s*)))
$\wedge$ (length (p-temp-stk (*new-l*)) $\not<$ arity (**'cons**))
$\wedge$ (length (s-ans (*new-s*)) $=$ arity (**'cons**)))
$\rightarrow$ (p-psw (p (p-set-pc (lr->p (*new-l*), *pc*),
p-cons-clock (p-set-pc (lr->p (*new-l*), *pc*))))
$=$ **'run**)

THEOREM: p-psw-run-run-false-lr-check-resourcesp
(lr-check-result (**'list**,

```
                   s-ans (new-s),
                   p-temp-stk (new-l),
                   p-data-segment (new-l),
                   orig-temp-stk)
 ∧   proper-p-statep (lr->p (new-l))
 ∧   (p-psw (new-l) = 'run)
 ∧   lr-programs-properp (new-l, table)
 ∧   (unlabel (get (offset (pc),
                    program-body (assoc (area-name (pc),
                                         p-prog-segment (lr->p (new-l))))))
       =  '(call false))
 ∧   (p-max-temp-stk-size (new-l)
       ≮  (length (p-temp-stk (new-l))
            +  car (s-apply-subr-r ('false, new-s))))
 ∧   (p-max-ctrl-stk-size (new-l)
       ≮  (p-ctrl-stk-size (p-ctrl-stk (new-l))
            +  cadr (s-apply-subr-r ('false, new-s))))
 ∧   (p-word-size (new-l)
       ≮  max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r ('false, new-s))))
 ∧   (lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (new-l)),
                           lr-free-list-nodes (lr-max-node (p-data-segment (new-l)),
                                               p-data-segment (new-l)),
                           p-data-segment (new-l))
       ≮  cadddr (s-apply-subr-r ('false, new-s)))
 ∧   (length (p-temp-stk (new-l)) ≮ arity ('false))
 ∧   (length (s-ans (new-s)) = arity ('false)))
→    (p-psw (p (p-set-pc (lr->p (new-l), pc),
                p-false-clock (p-set-pc (lr->p (new-l), pc))))
       =  'run)
```

THEOREM: p-psw-run-run-falsep-lr-check-resourcesp

```
 (lr-check-result ('list,
                   s-ans (new-s),
                   p-temp-stk (new-l),
                   p-data-segment (new-l),
                   orig-temp-stk)
 ∧   proper-p-statep (lr->p (new-l))
 ∧   (p-psw (new-l) = 'run)
 ∧   lr-programs-properp (new-l, table)
 ∧   (unlabel (get (offset (pc),
                    program-body (assoc (area-name (pc),
                                         p-prog-segment (lr->p (new-l))))))
       =  '(call falsep))
 ∧   (p-max-temp-stk-size (new-l)
```

```
      ≮  (length (p-temp-stk (new-l))
             +  car (s-apply-subr-r ('falsep, new-s))))
  ∧  (p-max-ctrl-stk-size (new-l)
      ≮  (p-ctrl-stk-size (p-ctrl-stk (new-l))
             +  cadr (s-apply-subr-r ('falsep, new-s))))
  ∧  (p-word-size (new-l)
      ≮  max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r ('falsep, new-s))))
  ∧  (lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (new-l)),
                           lr-free-list-nodes (lr-max-node (p-data-segment (new-l)),
                                               p-data-segment (new-l)),
                           p-data-segment (new-l))
      ≮  cadddr (s-apply-subr-r ('falsep, new-s)))
  ∧  (length (p-temp-stk (new-l)) ≮ arity ('falsep))
  ∧  (length (s-ans (new-s)) = arity ('falsep)))
→  (p-psw (p (p-set-pc (lr->p (new-l), pc),
              p-falsep-clock (p-set-pc (lr->p (new-l), pc))))
      =  'run)
```

THEOREM: p-psw-run-run-listp-lr-check-resourcesp

```
 (lr-check-result ('list,
                   s-ans (new-s),
                   p-temp-stk (new-l),
                   p-data-segment (new-l),
                   orig-temp-stk)
  ∧  proper-p-statep (lr->p (new-l))
  ∧  (p-psw (new-l) = 'run)
  ∧  lr-programs-properp (new-l, table)
  ∧  (unlabel (get (offset (pc),
                    program-body (assoc (area-name (pc),
                                         p-prog-segment (lr->p (new-l))))))
      =  '(call listp))
  ∧  (p-max-temp-stk-size (new-l)
      ≮  (length (p-temp-stk (new-l))
             +  car (s-apply-subr-r ('listp, new-s))))
  ∧  (p-max-ctrl-stk-size (new-l)
      ≮  (p-ctrl-stk-size (p-ctrl-stk (new-l))
             +  cadr (s-apply-subr-r ('listp, new-s))))
  ∧  (p-word-size (new-l)
      ≮  max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r ('listp, new-s))))
  ∧  (lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (new-l)),
                           lr-free-list-nodes (lr-max-node (p-data-segment (new-l)),
                                               p-data-segment (new-l)),
                           p-data-segment (new-l))
      ≮  cadddr (s-apply-subr-r ('listp, new-s)))
```

$\land$ (length (p-temp-stk (*new-l*)) $\not<$ arity (`'listp`))
$\land$ (length (s-ans (*new-s*)) = arity (`'listp`)))
$\rightarrow$ (p-psw (p (p-set-pc (lr->p (*new-l*), *pc*),
p-listp-clock (p-set-pc (lr->p (*new-l*), *pc*))))
= `'run`)

THEOREM: p-psw-run-run-nlistp-lr-check-resourcesp
(lr-check-result (`'list`,
s-ans (*new-s*),
p-temp-stk (*new-l*),
p-data-segment (*new-l*),
orig-temp-stk)
$\land$ proper-p-statep (lr->p (*new-l*))
$\land$ (p-psw (*new-l*) = `'run`)
$\land$ lr-programs-properp (*new-l*, *table*)
$\land$ (unlabel (get (offset (*pc*),
program-body (assoc (area-name (*pc*),
p-prog-segment (lr->p (*new-l*))))))
= `'(call nlistp)`)
$\land$ (p-max-temp-stk-size (*new-l*)
$\not<$ (length (p-temp-stk (*new-l*))
+ car (s-apply-subr-r (`'nlistp`, *new-s*))))
$\land$ (p-max-ctrl-stk-size (*new-l*)
$\not<$ (p-ctrl-stk-size (p-ctrl-stk (*new-l*))
+ cadr (s-apply-subr-r (`'nlistp`, *new-s*))))
$\land$ (p-word-size (*new-l*)
$\not<$ max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r (`'nlistp`, *new-s*))))
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (*new-l*)),
lr-free-list-nodes (lr-max-node (p-data-segment (*new-l*)),
p-data-segment (*new-l*)),
p-data-segment (*new-l*))
$\not<$ cadddr (s-apply-subr-r (`'nlistp`, *new-s*)))
$\land$ (length (p-temp-stk (*new-l*)) $\not<$ arity (`'nlistp`))
$\land$ (length (s-ans (*new-s*)) = arity (`'nlistp`)))
$\rightarrow$ (p-psw (p (p-set-pc (lr->p (*new-l*), *pc*),
p-nlistp-clock (p-set-pc (lr->p (*new-l*), *pc*))))
= `'run`)

THEOREM: p-psw-run-run-true-lr-check-resourcesp
(lr-check-result (`'list`,
s-ans (*new-s*),
p-temp-stk (*new-l*),
p-data-segment (*new-l*),
orig-temp-stk)

```
 ∧   proper-p-statep (lr->p (new-l))
 ∧   (p-psw (new-l) = 'run)
 ∧   lr-programs-properp (new-l, table)
 ∧   (unlabel (get (offset (pc),
                   program-body (assoc (area-name (pc),
                                        p-prog-segment (lr->p (new-l))))))
      =   '(call true))
 ∧   (p-max-temp-stk-size (new-l)
      ≮   (length (p-temp-stk (new-l))
             +   car (s-apply-subr-r ('true, new-s))))
 ∧   (p-max-ctrl-stk-size (new-l)
      ≮   (p-ctrl-stk-size (p-ctrl-stk (new-l))
             +   cadr (s-apply-subr-r ('true, new-s))))
 ∧   (p-word-size (new-l)
      ≮   max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r ('true, new-s))))
 ∧   (lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (new-l)),
                           lr-free-list-nodes (lr-max-node (p-data-segment (new-l)),
                                               p-data-segment (new-l)),
                           p-data-segment (new-l))
      ≮   cadddr (s-apply-subr-r ('true, new-s)))
 ∧   (length (p-temp-stk (new-l)) ≮ arity ('true))
 ∧   (length (s-ans (new-s)) = arity ('true)))
→    (p-psw (p (p-set-pc (lr->p (new-l), pc),
                p-true-clock (p-set-pc (lr->p (new-l), pc))))
      =   'run)
```

THEOREM: p-psw-run-run-truep-lr-check-resourcesp

```
(lr-check-result ('list,
                  s-ans (new-s),
                  p-temp-stk (new-l),
                  p-data-segment (new-l),
                  orig-temp-stk)
 ∧   proper-p-statep (lr->p (new-l))
 ∧   (p-psw (new-l) = 'run)
 ∧   lr-programs-properp (new-l, table)
 ∧   (unlabel (get (offset (pc),
                   program-body (assoc (area-name (pc),
                                        p-prog-segment (lr->p (new-l))))))
      =   '(call truep))
 ∧   (p-max-temp-stk-size (new-l)
      ≮   (length (p-temp-stk (new-l))
             +   car (s-apply-subr-r ('truep, new-s))))
 ∧   (p-max-ctrl-stk-size (new-l)
      ≮   (p-ctrl-stk-size (p-ctrl-stk (new-l))
```

$+$ cadr (s-apply-subr-r (`'truep`, *new-s*))))
$\land$ (p-word-size (*new-l*)
$\not<$ max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r (`'truep`, *new-s*))))
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (*new-l*)),
lr-free-list-nodes (lr-max-node (p-data-segment (*new-l*)),
p-data-segment (*new-l*)),
p-data-segment (*new-l*))
$\not<$ cadddr (s-apply-subr-r (`'truep`, *new-s*)))
$\land$ (length (p-temp-stk (*new-l*)) $\not<$ arity (`'truep`))
$\land$ (length (s-ans (*new-s*)) $=$ arity (`'truep`)))
$\rightarrow$ (p-psw (p (p-set-pc (lr->p (*new-l*), *pc*),
p-truep-clock (p-set-pc (lr->p (*new-l*), *pc*))))
$=$ `'run`)

THEOREM: length-last
listp (l) $\rightarrow$ (length (last (l)) $=$ `1`)

THEOREM: equal-plus-lessp-fact
$((x + z) = y) \rightarrow ((y < (n + x)) = (z < n))$

THEOREM: not-lessp-lr-count-free-nodes-lr-eval-list-lr-set-pos
let *new-l* **be** lr-eval (`'list`, lr-set-pos (s->lr1 (s, l, *table*), *pos*), c)
in
(proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\land$ lr-proper-heapp (p-data-segment (l))
$\land$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\land$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\land$ lr-s-similar-statesp (s-params (s),
s-temps (s),
s->lr1 (s, l, *table*),
table)
$\land$ s-good-statep (s, c)
$\land$ (p-psw (*new-l*) $=$ `'run`)
$\land$ (s-err-flag (s-eval (`'list`, s-set-pos (s, *pos*), c)) $=$ `'run`)
$\land$ listp (s-expr (s))
$\land$ (car (s-expr (s)) $\neq$ `'if`)
$\land$ (car (s-expr (s)) $\neq$ S-TEMP-EVAL)
$\land$ (car (s-expr (s)) $\neq$ S-TEMP-TEST)
$\land$ (car (s-expr (s)) $\neq$ S-TEMP-FETCH)
$\land$ (car (s-expr (s)) $\neq$ `'quote`)
$\land$ (*pos* $=$ dv (s-pos (s), `1`))
$\land$ (*max-addr* $=$ lr-max-node (p-data-segment (l))))
$\rightarrow$ ((lr-count-free-nodes (fetch (identity (LR-FP-ADDR),
p-data-segment (*new-l*)),
lr-free-list-nodes (*max-addr*,

p-data-segment (*new-l*)),
p-data-segment (*new-l*))
$<$ n)
$=$ (lr-count-free-nodes (fetch (identity (LR-FP-ADDR),
p-data-segment (l)),
lr-free-list-nodes (*max-addr*,
p-data-segment (l)),
p-data-segment (l))
$<$ (s-eval-heap-r (`'list`,
s-set-pos (s, dv (s-pos (s), `1`)),
c)
$+$ n))) **endlet**

EVENT: Disable equal-plus-lessp-fact.

THEOREM: lr-programs-properp-definedp-subrp-runtime-support
(($\neg$ definedp (car (lr-expr (l)), P-RUNTIME-SUPPORT-PROGRAMS))
$\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
$\wedge$ subrp (car (lr-expr (l)))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l))))
$\rightarrow$ ($\neg$ lr-programs-properp (l, *table*))

THEOREM: p-psw-run-p-run-subr-lr-check-resourcesp
let *new-l* **be** lr-eval (`'list`, lr-set-pos (s->lr1 (s, l, *table*), *pos*), c),
new-s **be** s-eval (`'list`, s-set-pos (s, *pos*), c),
pc **be** lr-return-pc (s->lr1 (s, l, *table*)),
r **be** s-apply-subr-r (car (s-expr (s)), s-eval (`'list`, s-set-pos (s, *pos*), c))
in
(proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\wedge$ ($c \not\simeq$ `0`)
$\wedge$ listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $\neq$ `'if`)
$\wedge$ (car (s-expr (s)) $\neq$ `'quote`)
$\wedge$ (s-err-flag (*new-s*) $=$ `'run`)
$\wedge$ subrp (car (s-expr (s)))
$\wedge$ (p-psw (*new-l*) $=$ `'run`)
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\wedge$ lr-s-similar-statesp (s-params (s),
s-temps (s),
s->lr1 (s, l, *table*),
table)
$\wedge$ s-good-statep (s, c)

$\land$ (p-max-temp-stk-size (l)
  $\not<$ (length (p-temp-stk (l))
    $+$ arity (car (s-expr (s)))
    $+$ car (r)))
$\land$ (p-max-ctrl-stk-size (l)
  $\not<$ (p-ctrl-stk-size (p-ctrl-stk (l)) $+$ cadr (r)))
$\land$ (p-word-size (l) $\not<$ max (S-MAX-SUBR-REQS, caddr (r)))
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (l)),
        lr-free-list-nodes (lr-max-node (p-data-segment (l)),
          p-data-segment (l)),
        p-data-segment (l))
  $\not<$ (cadddr (r)
    $+$ s-eval-heap-r (`'list`, s-set-pos (s, pos), c)))
$\land$ (pos $=$ dv (s-pos (s), `1`)))
$\rightarrow$ (p-psw (p-run-subr (car (s-expr (s)), p-set-pc (lr->p ($new\text{-}l$), pc)))
  $=$ `'run`) **endlet**

EVENT: Disable lr-programs-properp-definedp-subrp-runtime-support.

THEOREM: not-lessp-help-fact
$((x \not< y) \land (x \not< z)) \rightarrow ((x < \max(y, z)) = \mathbf{f})$

THEOREM: p-psw-run-lr-apply-subr-lr-check-resourcesp
(($c \not\simeq$ `0`)
$\land$ listp (s-expr (s))
$\land$ (car (s-expr (s)) $\neq$ `'if`)
$\land$ (car (s-expr (s)) $\neq$ `'quote`)
$\land$ (s-err-flag (s-eval (`'list`, s-set-pos (s, dv (s-pos (s), `1`)), c)) $=$ `'run`)
$\land$ subrp (car (s-expr (s)))
$\land$ (p-psw (lr-eval (`'list`,
      lr-set-pos (s->lr1 (s, l, $table$), dv (s-pos (s), `1`)),
      c))
  $=$ `'run`)
$\land$ proper-p-statep (lr->p (s->lr1 (s, l, $table$)))
$\land$ lr-proper-heapp (p-data-segment (l))
$\land$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\land$ lr-programs-properp (s->lr1 (s, l, $table$), $table$)
$\land$ lr-s-similar-statesp (s-params (s), s-temps (s), s->lr1 (s, l, $table$), $table$)
$\land$ s-good-statep (s, c)
$\land$ lr-check-resourcesp ($flag$, s, l, c)
$\land$ (p-word-size (l) $\not<$ S-MAX-SUBR-REQS)
$\land$ ($flag$ $\neq$ `'list`))
$\rightarrow$ (p-psw (lr-apply-subr (s->lr1 (s, l, $table$),
      lr-eval (`'list`,

lr-set-pos (s->lr1 (s, l, *table*),
dv (s-pos (s), `1`)),
c)))
= `'run`)

THEOREM: strip-logic-fnames-lr-compile-programs
strip-logic-fnames (lr-compile-programs (*programs*, *const-table*))
= strip-logic-fnames (*programs*)

THEOREM: strip-logic-fnames-cdr-lr-compile-programs
strip-logic-fnames (cdr (lr-compile-programs (*programs*, *const-table*)))
= strip-logic-fnames (cdr (*programs*))

THEOREM: lr-programs-properp-s->lr1-definedp-cdr-s-progs
(lr-programs-properp (s->lr1 (s, l, *table*), *table*)
∧ listp (s-expr (s))
∧ (car (s-expr (s)) ≠ `'if`)
∧ (car (s-expr (s)) ≠ `'quote`)
∧ (¬ subrp (car (s-expr (s))))
∧ litatom (car (s-expr (s)))
∧ good-posp1 (s-pos (s), s-body (s-prog (s)))
∧ s-programs-okp (cdr (s-progs (s))))
→ definedp (user-fname (car (s-expr (s))), cdr (s-progs (s)))

THEOREM: s-programs-okp-formals-not-f
(s-programs-okp (*progs*) ∧ (*prog* ∈ *progs*)) → (s-formals (*prog*) ≠ **f**)

THEOREM: not-lessp-plus-arity-length-formals
(listp (s-expr (s))
∧ (car (s-expr (s)) ≠ `'if`)
∧ (car (s-expr (s)) ≠ `'quote`)
∧ (¬ subrp (car (s-expr (s))))
∧ litatom (car (s-expr (s)))
∧ good-posp1 (s-pos (s), s-body (s-prog (s)))
∧ s-good-statep (s, c))
→ (((arity (car (s-expr (s))) + x) < length (formals (car (s-expr (s)))))
= **f**)

THEOREM: length-lr-make-temp-var-dcls
length (lr-make-temp-var-dcls (*temp-alist*)) = length (*temp-alist*)

THEOREM: length-lr-make-temp-name-alist-1
length (lr-make-temp-name-alist-1 (*initial*, *num-list*, *temp-list*, *formals*))
= length (*temp-list*)

THEOREM: length-lr-make-temp-name-alist
$\text{length}\,(\text{lr-make-temp-name-alist}\,(\textit{temp-list},\ \textit{formals})) = \text{length}\,(\textit{temp-list})$

THEOREM: p-ctrl-stk-size-0
$(\text{p-ctrl-stk-size}\,(\textit{ctrl-stk}) = \texttt{0}) = (\neg\ \text{listp}\,(\textit{ctrl-stk}))$

THEOREM: length-make-temps-entries
$\text{length}\,(\text{make-temps-entries}\,(\textit{list})) = \text{length}\,(\textit{list})$

THEOREM: s-eval-ctrl-r-funcall-opener

$$
\begin{array}{l}
\textbf{let}\ \textit{arg-s}\ \ \textbf{be}\ \ \text{s-eval}\,(\texttt{'list},\ \text{s-set-pos}\,(s,\ \text{dv}\,(\text{s-pos}\,(s),\ \texttt{1})),\ c) \\
\textbf{in} \\
((c \not\simeq \texttt{0}) \\
\ \wedge\quad \text{s-good-statep}\,(s,\ c) \\
\ \wedge\quad \text{listp}\,(\text{s-expr}\,(s)) \\
\ \wedge\quad (\text{car}\,(\text{s-expr}\,(s)) \neq \texttt{'if}) \\
\ \wedge\quad (\text{car}\,(\text{s-expr}\,(s)) \neq \texttt{'quote}) \\
\ \wedge\quad (\neg\ \text{subrp}\,(\text{car}\,(\text{s-expr}\,(s)))) \\
\ \wedge\quad \text{litatom}\,(\text{car}\,(\text{s-expr}\,(s))) \\
\ \wedge\quad (\textit{flag} \neq \texttt{'list}) \\
\ \wedge\quad \text{good-posp1}\,(\text{s-pos}\,(s),\ \text{s-body}\,(\text{s-prog}\,(s))) \\
\ \wedge\quad (\text{s-err-flag}\,(\textit{arg-s}) = \texttt{'run})) \\
\rightarrow\quad (\text{s-eval-ctrl-r}\,(\textit{flag},\ s,\ c) \\
\qquad = \quad \max(\text{s-eval-ctrl-r}\,(\texttt{'list}, \\
\qquad\qquad\qquad\qquad\qquad \text{s-set-pos}\,(s,\ \text{dv}\,(\text{s-pos}\,(s),\ \texttt{1})), \\
\qquad\qquad\qquad\qquad\qquad c), \\
\qquad\qquad\quad 1 + (1 + (\text{length}\,(\text{formals}\,(\text{car}\,(\text{s-expr}\,(s)))) \\
\qquad\qquad\qquad\qquad + \quad \text{length}\,(\text{s-temp-list}\,(\text{assoc}\,(\text{user-fname}\,(\text{car}\,(\text{s-expr}\,(s))), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{s-progs}\,(s)))) \\
\qquad\qquad\qquad\qquad + \quad \text{s-eval-ctrl-r}\,(\mathbf{t}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{s-fun-call-state}\,(\textit{arg-s}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{car}\,(\text{s-expr}\,(s))), \\
\qquad\qquad\qquad\qquad\qquad\qquad c - 1))))) \ \textbf{endlet}
\end{array}
$$

THEOREM: s-good-statep-formals-assoc-cdr-s-progs

$$
\begin{array}{l}
(\text{s-good-statep}\,(s,\ c) \\
\ \wedge\quad \text{listp}\,(\text{s-expr}\,(s)) \\
\ \wedge\quad (\text{car}\,(\text{s-expr}\,(s)) \neq \texttt{'if}) \\
\ \wedge\quad (\text{car}\,(\text{s-expr}\,(s)) \neq \texttt{'quote}) \\
\ \wedge\quad (\neg\ \text{subrp}\,(\text{car}\,(\text{s-expr}\,(s)))) \\
\ \wedge\quad \text{good-posp1}\,(\text{s-pos}\,(s),\ \text{s-body}\,(\text{s-prog}\,(s))) \\
\ \wedge\quad \text{litatom}\,(\text{car}\,(\text{s-expr}\,(s))) \\
\ \wedge\quad (\textit{progs} = \text{cdr}\,(\text{s-progs}\,(s)))) \\
\rightarrow\quad (\text{s-formals}\,(\text{assoc}\,(\text{user-fname}\,(\text{car}\,(\text{s-expr}\,(s))),\ \textit{progs})) \\
\qquad = \quad \text{formals}\,(\text{car}\,(\text{s-expr}\,(s))))
\end{array}
$$

THEOREM: not-lessp-p-ctrl-stk-size-make-p-call-frame

let *s-prog* **be** assoc (user-fname (car (s-expr (s))), cdr (s-progs (s))),
lr-eval **be** lr-eval ('`list`,
lr-set-pos (s->lr1 (s, l, *table*), dv (s-pos (s), `1`)),
c)
in
(listp (s-expr (s))
$\land$ (car (s-expr (s)) $\neq$ '`if`)
$\land$ (car (s-expr (s)) $\neq$ '`quote`)
$\land$ ($\neg$ subrp (car (s-expr (s))))
$\land$ litatom (car (s-expr (s)))
$\land$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\land$ s-good-statep (s, c)
$\land$ ($c \not\simeq$ `0`)
$\land$ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\land$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\land$ (length (*temp-list*) = length (s-temp-list (*s-prog*)))
$\land$ (p-psw (*lr-eval*) = '`run`)
$\land$ (s-err-flag (s-eval ('`list`, s-set-pos (s, dv (s-pos (s), `1`)), c))
= '`run`)
$\land$ (p-max-ctrl-stk-size (l)
$\not<$ (p-ctrl-stk-size (p-ctrl-stk (l))
+ s-eval-ctrl-r (*flag*, s, c)))
$\land$ (*flag* $\neq$ '`list`))
$\rightarrow$ (p-max-ctrl-stk-size (l)
$\not<$ p-ctrl-stk-size (cons (make-p-call-frame (formals (car (s-expr (s))),
temp-stk,
temp-list,
pc),
p-ctrl-stk (*lr-eval*)))) **endlet**

THEOREM: definedp-0
definedp (x, `0`) = **f**

THEOREM: not-definedp-user-fname-p-runtime-support-programs
$\neg$ definedp (user-fname (*name*), P-RUNTIME-SUPPORT-PROGRAMS)

THEOREM: comp-programs-assoc-cons-opener
(user-fname (*name*) $\neq$ *prog1-name*)
$\rightarrow$ (assoc (user-fname (*name*),
comp-programs (cons (cons (*prog1-name*, *prog1*), *progs*)))
= assoc (user-fname (*name*), comp-programs-1 (*progs*)))

THEOREM: lr-check-resourcesp-lr-funcall-p-psw-run
(($c \not\simeq$ `0`)

$\wedge$ listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $\neq$ `'if`)
$\wedge$ (car (s-expr (s)) $\neq$ `'quote`)
$\wedge$ ($\neg$ subrp (car (s-expr (s))))
$\wedge$ litatom (car (s-expr (s)))
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ lr-programs-properp (s->lr1 (s, l, $table$), $table$)
$\wedge$ (p-psw (lr-eval (`'list`, lr-set-pos (s->lr1 (s, l, $table$), pos), c))
= `'run`)
$\wedge$ (s-err-flag (s-eval (`'list`, s-set-pos (s, pos), c)) = `'run`)
$\wedge$ (pos = dv (s-pos (s), `1`))
$\wedge$ proper-p-statep (lr->p (s->lr1 (s, l, $table$)))
$\wedge$ s-good-statep (s, c)
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-check-resourcesp ($flag$, s, l, c)
$\wedge$ ($flag$ $\neq$ `'list`))
$\rightarrow$ (p-psw (lr-funcall (s->lr1 (s, l, $table$),
lr-eval (`'list`, lr-set-pos (s->lr1 (s, l, $table$), pos), c)))
= `'run`)

THEOREM: lessp-max-arg2
max (x, y) $\not<$ y

THEOREM: not-lessp-plus-arity-length-formals-alt
(listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $\neq$ `'if`)
$\wedge$ (car (s-expr (s)) $\neq$ `'quote`)
$\wedge$ ($\neg$ subrp (car (s-expr (s))))
$\wedge$ litatom (car (s-expr (s)))
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ s-good-statep (s, c))
$\rightarrow$ (((x + arity (car (s-expr (s)))) < length (formals (car (s-expr (s)))))
= **f**)

THEOREM: listp-lr-compile-programs
listp (lr-compile-programs ($progs$, $table$)) = listp ($progs$)

THEOREM: caar-lr-compile-programs
listp ($progs$) $\rightarrow$ (caar (lr-compile-programs ($progs$, $table$)) = caar ($progs$))

THEOREM: length-p-temp-stk-lr-eval-lr-funcall
let *lr-eval* **be** lr-eval (`'list`, lr-set-pos (s->lr1 (s, l, $table$), pos), c)
in
(listp (s-expr (s))
$\wedge$ ($\neg$ subrp (car (s-expr (s))))

$\wedge$ (car (s-expr (s)) $\neq$ `'quote`)
$\wedge$ (car (s-expr (s)) $\neq$ `'if`)
$\wedge$ litatom (car (s-expr (s)))
$\wedge$ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ s-good-statep (s, c)
$\wedge$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\wedge$ (p-psw (*lr-eval*) = `'run`)
$\wedge$ (p-psw (lr-funcall (s->lr1 (s, l, *table*), *lr-eval*)) = `'run`)
$\wedge$ (s-err-flag (s-eval (`'list`, s-set-pos (s, *pos*), c)) = `'run`)
$\wedge$ (*pos* = dv (s-pos (s), `1`)))
$\rightarrow$ (length (p-temp-stk (lr-funcall (s->lr1 (s, l, *table*), *lr-eval*)))
= length (p-temp-stk (l))) **endlet**

THEOREM: p-ctrl-stk-size-p-ctrl-stk-lr-funcall
(p-psw (lr-funcall (l, *new-l*)) = `'run`)
$\rightarrow$ (p-ctrl-stk-size (p-ctrl-stk (lr-funcall (l, *new-l*)))
= (`2`
+ length (formal-vars (assoc (user-fname (car (lr-expr (l))),
p-prog-segment (l))))
+ length (temp-var-dcls (assoc (user-fname (car (lr-expr (l))),
p-prog-segment (l))))
+ p-ctrl-stk-size (p-ctrl-stk (*new-l*))))

THEOREM: lr-programs-properp-s->lr1-definedp-s-progs
(lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\wedge$ listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $\neq$ `'if`)
$\wedge$ (car (s-expr (s)) $\neq$ `'quote`)
$\wedge$ ($\neg$ subrp (car (s-expr (s))))
$\wedge$ litatom (car (s-expr (s)))
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ s-programs-okp (cdr (s-progs (s))))
$\rightarrow$ definedp (user-fname (car (s-expr (s))), s-progs (s))

THEOREM: s-eval-ctrl-heap-temp-ws-s-fun-call-state-opener
let *s-eval* **be** s-eval (`'list`, s-set-pos (s, dv (s-pos (s), `1`)), c)
in
(($c \not\simeq$ `0`)
$\wedge$ listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $\neq$ `'if`)
$\wedge$ (car (s-expr (s)) $\neq$ `'quote`)
$\wedge$ ($\neg$ subrp (car (s-expr (s))))
$\wedge$ litatom (car (s-expr (s)))
$\wedge$ s-good-statep (s, c)

$\wedge$ (s-err-flag (*s-eval*) = `'run`)
$\wedge$ (*flag* $\neq$ `'list`))
$\rightarrow$ ((s-eval-ctrl-r (*flag*, s, c)
= max (s-eval-ctrl-r (`'list`,
s-set-pos (s, dv (s-pos (s), `1`)),
c),
`2`
\+ length (s-formals (assoc (user-fname (car (s-expr (s))),
s-progs (s))))
\+ length (s-temp-list (assoc (user-fname (car (s-expr (s))),
s-progs (s))))
\+ s-eval-ctrl-r (**t**,
s-fun-call-state (*s-eval*,
car (s-expr (s))),
$c - 1$)))
$\wedge$ (s-eval-heap-r (*flag*, s, c)
= (s-eval-heap-r (`'list`,
s-set-pos (s, dv (s-pos (s), `1`)),
c)
\+ s-eval-heap-r (**t**,
s-fun-call-state (*s-eval*,
car (s-expr (s))),
$c - 1$)))
$\wedge$ (s-eval-temp-r (*flag*, s, c)
= max (s-eval-temp-r (`'list`,
s-set-pos (s, dv (s-pos (s), `1`)),
c),
s-eval-temp-r (**t**,
s-fun-call-state (*s-eval*,
car (s-expr (s))),
$c - 1$)))
$\wedge$ (s-eval-ws-r (*flag*, s, c)
= max (s-eval-ws-r (`'list`,
s-set-pos (s, dv (s-pos (s), `1`)),
c),
s-eval-ws-r (**t**,
s-fun-call-state (*s-eval*,
car (s-expr (s))),
$c - 1$)))) **endlet**

THEOREM: lr-check-resourcesp-lr-funcall-s-fun-call-state
(($c \not\simeq$ `0`)
$\wedge$ listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $\neq$ `'if`)

$\wedge$ (car (s-expr (s)) $\neq$ S-TEMP-EVAL)
$\wedge$ (car (s-expr (s)) $\neq$ S-TEMP-TEST)
$\wedge$ (car (s-expr (s)) $\neq$ S-TEMP-FETCH)
$\wedge$ (car (s-expr (s)) $\neq$ `'quote`)
$\wedge$ ($\neg$ subrp (car (s-expr (s))))
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ litatom (car (s-expr (s)))
$\wedge$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
$\wedge$ (p-psw (lr-eval (`'list`, lr-set-pos (s->lr1 (s, l, *table*), *pos*), c))
= `'run`)
$\wedge$ (s-err-flag (s-eval (`'list`, s-set-pos (s, *pos*), c)) = `'run`)
$\wedge$ (*pos* = dv (s-pos (s), `1`))
$\wedge$ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
$\wedge$ s-good-statep (s, c)
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-check-resourcesp (*flag*, s, l, c)
$\wedge$ (*flag* $\neq$ `'list`)
$\wedge$ lr-s-similar-statesp (s-params (s), s-temps (s), s->lr1 (s, l, *table*), *table*))
$\rightarrow$ lr-check-resourcesp (**t**,
s-fun-call-state (s-eval (`'list`,
s-set-pos (s, *pos*),
c),
car (s-expr (s))),
lr-funcall (s->lr1 (s, l, *table*),
lr-eval (`'list`,
lr-set-pos (s->lr1 (s, l, *table*),
pos),
c)),
$c - 1$)

EVENT: Disable s-eval-ctrl-heap-temp-ws-s-fun-call-state-opener.

THEOREM: s-eval-flag-run-car-s-apply-subr-r-not-zero
(listp (s-expr (s))
$\wedge$ (car (s-expr (s)) $\neq$ `'if`)
$\wedge$ (car (s-expr (s)) $\neq$ `'quote`)
$\wedge$ (s-err-flag (s-eval (`'list`, s-set-pos (s, dv (s-pos (s), `1`)), c)) = `'run`)
$\wedge$ subrp (car (s-expr (s)))
$\wedge$ (p-psw (lr-apply-subr (s->lr1 (s, l, *table*), *new-l*)) = `'run`)
$\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
$\wedge$ s-good-statep (s, c))
$\rightarrow$ (car (s-apply-subr-r (car (s-expr (s)),
s-eval (`'list`, s-set-pos (s, dv (s-pos (s), `1`)), c)))
$\not<$ `1`)

THEOREM: length-p-temp-stk-lr-eval-lr-set-pos-flag-t
(proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
 $\land$ good-posp1 (*pos*, s-body (s-prog (s)))
 $\land$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
 $\land$ s-good-statep (s, c)
 $\land$ (p-psw (lr-eval (**t**, lr-set-pos (s->lr1 (s, l, *table*), *pos*), c)) = **'run**))
$\rightarrow$ (length (p-temp-stk (lr-eval (**t**, lr-set-pos (s->lr1 (s, l, *table*), *pos*), c)))
 = (1 + length (p-temp-stk (l))))

THEOREM: s-eval-flag-run-s-eval-temp-r-not-zero
((p-psw (lr-eval (*flag*, s->lr1 (s, l, *table*), c)) = **'run**)
 $\land$ (s-err-flag (s-eval (*flag*, s, c)) = **'run**)
 $\land$ good-posp (*flag*, s-pos (s), s-body (s-prog (s)))
 $\land$ s-good-statep (s, c)
 $\land$ (*flag* $\neq$ **'list**))
$\rightarrow$ (s-eval-temp-r (*flag*, s, c) $\not<$ 1)

THEOREM: p-psw-run-p-psw-lr-if-ok-not-run-check-resourcesp
((*flag* $\neq$ **'list**)
 $\land$ (c $\not\simeq$ 0)
 $\land$ listp (s-expr (s))
 $\land$ (car (s-expr (s)) = **'if**)
 $\land$ s-good-statep (s, c)
 $\land$ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
 $\land$ proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
 $\land$ (s-err-flag (s-eval (**t**, s-set-pos (s, dv (s-pos (s), 1)), c)) = **'run**)
 $\land$ (p-psw (lr-if-ok (lr-eval (**t**,
 lr-set-pos (s->lr1 (s, l, *table*), dv (s-pos (s), 1)),
 c)))
 $\neq$ **'run**)
 $\land$ (p-psw (lr-eval (**t**, lr-set-pos (s->lr1 (s, l, *table*), dv (s-pos (s), 1)), c))
 = **'run**)
 $\land$ good-posp1 (s-pos (s), s-body (s-prog (s))))
$\rightarrow$ ($\neg$ lr-check-resourcesp (*flag*, s, l, c))

THEOREM: not-lr-check-resourcesp-temp-test-bad-max-temp-stk-size
((*flag* $\neq$ **'list**)
 $\land$ (c $\neq$ 0)
 $\land$ ($c \in \mathbf{N}$)
 $\land$ listp (s-expr (s))
 $\land$ (car (s-expr (s)) = S-TEMP-TEST)
 $\land$ (p-max-temp-stk-size (l) < (2 + length (p-temp-stk (l))))
 $\land$ s-good-statep (s, c))
$\rightarrow$ ($\neg$ lr-check-resourcesp (*flag*, s, l, c))

THEOREM: lr-eval-s-eval-flag-run
(proper-p-statep (lr->p (s->lr1 (s, l, *table*)))
∧ lr-proper-heapp (p-data-segment (l))
∧ good-posp (*flag*, s-pos (s), s-body (s-prog (s)))
∧ lr-programs-properp (s->lr1 (s, l, *table*), *table*)
∧ lr-s-similar-statesp (s-params (s), s-temps (s), s->lr1 (s, l, *table*), *table*)
∧ s-good-statep (s, c)
∧ s-all-temps-setp (*flag*,
if *flag* = `'list` **then** s-expr-list (s)
else s-expr (s) **endif**,
temp-alist-to-set (s-temps (s)))
∧ s-all-progs-temps-setp (s-progs (s))
∧ s-check-temps-setp (s-temps (s))
∧ (s-err-flag (s-eval (*flag*, s, c)) = `'run`)
∧ lr-check-resourcesp (*flag*, s, l, c)
∧ (p-word-size (l) $\not<$ S-MAX-SUBR-REQS))
→ (p-psw (lr-eval (*flag*, s->lr1 (s, l, *table*), c)) = `'run`)

THEOREM: plistp-lr-compile-body-1
plistp (*prog*) → plistp (lr-compile-body (*flag*, *prog*, *temp-alist*, *table*))

DEFINITION:
l-restrict-subrps (*flag*, *expr*)
= **if** *flag* = `'list`
then if listp (*expr*)
then l-restrict-subrps (**t**, car (*expr*))
∧ l-restrict-subrps (`'list`, cdr (*expr*))
else t endif
elseif listp (*expr*)
then if car (*expr*) = `'quote` **then t**
elseif car (*expr*) = `'if`
then l-restrict-subrps (`'list`, cdr (*expr*))
elseif subrp (car (*expr*))
then definedp (car (*expr*), P-RUNTIME-SUPPORT-PROGRAMS)
∧ l-restrict-subrps (`'list`, cdr (*expr*))
elseif body (car (*expr*)) **then** l-restrict-subrps (`'list`, cdr (*expr*))
else t endif
else t endif

DEFINITION:
l-restrict-subrps-progs (*pnames*)
= **if** listp (*pnames*)
then l-restrict-subrps (**t**, body (car (*pnames*)))
∧ l-restrict-subrps-progs (cdr (*pnames*))
else t endif

DEFINITION:
s-restrict-subrps (*flag*, *expr*)
= **if** *flag* = `'list`
  then if listp (*expr*)
    then s-restrict-subrps (**t**, car (*expr*))
      ∧ s-restrict-subrps (`'list`, cdr (*expr*))
    else t endif
  elseif listp (*expr*)
  then if car (*expr*) = `'quote` **then t**
    elseif (car (*expr*) = S-TEMP-FETCH)
      ∨ (car (*expr*) = S-TEMP-EVAL)
      ∨ (car (*expr*) = S-TEMP-TEST)
    then s-restrict-subrps (**t**, cadr (*expr*))
    elseif car (*expr*) = `'if`
    then s-restrict-subrps (`'list`, cdr (*expr*))
    elseif subrp (car (*expr*))
    then definedp (car (*expr*), P-RUNTIME-SUPPORT-PROGRAMS)
      ∧ s-restrict-subrps (`'list`, cdr (*expr*))
    elseif body (car (*expr*)) **then** s-restrict-subrps (`'list`, cdr (*expr*))
    else t endif
  else t endif

DEFINITION:
s-restrict-subrps-progs (*progs*)
= **if** listp (*progs*)
  then s-restrict-subrps (**t**, s-body (car (*progs*)))
    ∧ s-restrict-subrps-progs (cdr (*progs*))
  else t endif

THEOREM: s-proper-exprp-plist-temp-list
s-proper-exprp (*flag*, *expr*, *program-names*, *formals*, plist (*temp-list*))
= s-proper-exprp (*flag*, *expr*, *program-names*, *formals*, *temp-list*)

THEOREM: not-listp-s-progs-not-s-good-statep
(¬ listp (s-progs (*s*))) → (¬ s-good-statep (*s*, *c*))

THEOREM: length-lr-init-heap-contents
length (lr-init-heap-contents (*addr*, *size*)) = (1 + (*size* ∗ LR-NODE-SIZE))

THEOREM: fetch-cons
fetch (list (*x*, cons (*name1*, *n*)), cons (cons (*name2*, *contents*), *rest-data-seg*))
= **if** *name1* = *name2* **then** get (*n*, *contents*)
  else fetch (list (*x*, cons (*name1*, *n*)), *rest-data-seg*) **endif**

THEOREM: lr-s-similar-const-table-cons

lr-s-similar-const-table (cons (cons (*object*, *addr*), *table*), *data-seg*)
$=$ (lr-valp (*object*, *addr*, *data-seg*)
$\wedge$ lr-s-similar-const-table (*table*, *data-seg*))

THEOREM: lr-s-similar-const-table-nil
lr-s-similar-const-table (**nil**, *data-seg*)

THEOREM: lr-init-heap-contents-add1-opener
lr-init-heap-contents (*addr*, 1 + *size*)
$=$ append (lr-new-node (tag (`'nat`, LR-INIT-TAG),
add-addr (*addr*, LR-NODE-SIZE),
tag (`'nat`, `0`),
tag (`'nat`, `0`)),
lr-init-heap-contents (add-addr (*addr*, LR-NODE-SIZE), *size*))

THEOREM: deposit-cons
deposit (*object*,
list (*x*, cons (*name1*, *n*)),
cons (cons (*name2*, *contents*), *rest-data-seg*))
$=$ **if** *name1* = *name2*
then cons (cons (*name1*, put (*object*, *n*, *contents*)), *rest-data-seg*)
else cons (cons (*name2*, *contents*),
deposit (*object*, list (*x*, cons (*name1*, *n*)), *rest-data-seg*)) **endif**

THEOREM: adpp-cons-pack-opener
($n \in \mathbf{N}$)
$\rightarrow$ (adpp (cons (pack (*xxx*), *n*), cons (cons (pack (*yyy*), *contents*), *rest*))
$=$ **if** *xxx* = *yyy* **then** $n <$ length (*contents*)
else adpp (cons (pack (*xxx*), *n*), *rest*) **endif**)

THEOREM: fetch-deposit-a-list
((offset (*addr1*) $\in \mathbf{N}$) $\wedge$ (offset (*addr2*) $\in \mathbf{N}$) $\wedge$ listp (*list*))
$\rightarrow$ (fetch (*addr1*, deposit-a-list (*list*, *addr2*, *data-seg*))
$=$ **if** definedp (area-name (*addr2*), *data-seg*)
then if area-name (*addr1*) = area-name (*addr2*)
then if (offset (*addr1*) $\not<$ offset (*addr2*))
$\wedge$ (offset (*addr1*)
$<$ (offset (*addr2*) + length (*list*)))
then get (offset (*addr1*) $-$ offset (*addr2*), *list*)
else fetch (*addr1*, *data-seg*) **endif**
else fetch (*addr1*, *data-seg*) **endif**
else fetch (*addr1*, *data-seg*) **endif**)

THEOREM: lr-valp-0-lr-0-addr-opener
lr-valp (`0`, identity (LR-0-ADDR), *data-seg*)

$=\quad$ (adpp (identity (untag (LR-0-ADDR)), *data-seg*)
$\quad\wedge\quad$ (type (fetch (identity (add-addr (LR-0-ADDR, LR-REF-COUNT-OFFSET)), *data-seg*))
$\quad\quad=\quad$ 'nat)
$\quad\wedge\quad$ (untag (fetch (identity (LR-0-ADDR), *data-seg*)) $=$ LR-ADD1-TAG)
$\quad\wedge\quad$ (untag (fetch (identity (add-addr (LR-0-ADDR, LR-UNBOX-NAT-OFFSET)), *data-seg*))
$\quad\quad=\quad$ 0))

EVENT: Disable lr-valp-0-lr-0-addr-opener.

THEOREM: lr-valp-t-lr-t-addr-opener
lr-valp (**t**, identity (LR-T-ADDR), *data-seg*)
$=\quad$ (adpp (identity (untag (LR-T-ADDR)), *data-seg*)
$\quad\wedge\quad$ (type (fetch (identity (add-addr (LR-T-ADDR, LR-REF-COUNT-OFFSET)), *data-seg*))
$\quad\quad=\quad$ 'nat)
$\quad\wedge\quad$ (untag (fetch (identity (LR-T-ADDR), *data-seg*)) $=$ LR-TRUE-TAG))

EVENT: Disable lr-valp-t-lr-t-addr-opener.

THEOREM: lr-valp-f-lr-f-addr-opener
lr-valp (**f**, identity (LR-F-ADDR), *data-seg*)
$=\quad$ (adpp (identity (untag (LR-F-ADDR)), *data-seg*)
$\quad\wedge\quad$ (type (fetch (identity (add-addr (LR-F-ADDR, LR-REF-COUNT-OFFSET)), *data-seg*))
$\quad\quad=\quad$ 'nat)
$\quad\wedge\quad$ (untag (fetch (identity (LR-F-ADDR), *data-seg*)) $=$ LR-FALSE-TAG))

EVENT: Disable lr-valp-f-lr-f-addr-opener.

THEOREM: definedp-table-definedp-cdr-lr-compile-quote
definedp (x, *table*)
$\rightarrow\quad$ definedp (x, cdr (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))

THEOREM: definedp-car-lr-compile-quote
definedp (x, car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))
$=\quad$ definedp (x, *data-seg*)

THEOREM: lr-proper-p-areasp-car-lr-compile-quote
lr-proper-p-areasp (*data-seg*)
$\rightarrow\quad$ lr-proper-p-areasp (car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))

THEOREM: length-deposit-a-list
listp (*list*)
→ (length (cdr (assoc (*name*, deposit-a-list (*list*, *addr*, *data-seg*))))
  = **if** definedp (area-name (*addr*), *data-seg*)
    then if area-name (*addr*) = *name*
      then if length (cdr (assoc (*name*, *data-seg*)))
          < (offset (*addr*) + length (*list*))
        then offset (*addr*) + length (*list*)
        else length (cdr (assoc (*name*, *data-seg*))) **endif**
      else length (cdr (assoc (*name*, *data-seg*))) **endif**
    else length (cdr (assoc (*name*, *data-seg*))) **endif**)

THEOREM: adpp-lr-compile-quote
adpp (*addr*, *data-seg*)
→ adpp (*addr*, car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))

THEOREM: adpp-untag-definedp-area-name-free-ptr
adpp (untag (LR-FP-ADDR), *data-seg*)
→ definedp (identity (area-name (LR-FP-ADDR)), *data-seg*)

THEOREM: lr-max-node-deposit-a-list
(adpp (untag (*addr*), *data-seg*)
∧ listp (*list*)
∧ ((offset (*addr*) + length (*list*))
  < length (cdr (assoc (area-name (*addr*), *data-seg*)))))
→ (lr-max-node (deposit-a-list (*list*, *addr*, *data-seg*))
  = lr-max-node (*data-seg*))

DEFINITION:
all-p-objects-lookup (*list*, *table*, *p*)
= **if** listp (*list*)
  then p-objectp (cdr (assoc (car (*list*), *table*)), *p*)
    ∧ all-p-objects-lookup (cdr (*list*), *table*, *p*)
  else t endif

THEOREM: proper-p-alistp-all-litatoms-all-p-objectps-lookup
(all-litatoms (strip-cars (*params*))
∧ all-p-objects-lookup (strip-cdrs (*params*), *table*, *p*))
→ proper-p-alistp (pair-formals-with-addresses (*params*, *table*), *p*)

THEOREM: definedp-table-definedp-cdr-lr-data-seg-table-body
definedp (*object*, *table*)
→ definedp (*object*, cdr (lr-data-seg-table-body (*flag*, *expr*, *data-seg*, *table*)))

THEOREM: definedp-table-definedp-cdr-lr-data-seg-table-list
definedp (*object*, *table*)
→ definedp (*object*, cdr (lr-data-seg-table-list (*progs*, *data-seg*, *table*)))

THEOREM: definedp-table-definedp-cdr-lr-init-data-seg-table
definedp (*object*, *table*)
$\rightarrow$ definedp (*object*, cdr (lr-init-data-seg-table (*params*, *data-seg*, *table*)))

THEOREM: definedp-table-definedp-car-lr-data-seg-table-body
definedp (*name*, *data-seg*)
$\rightarrow$ definedp (*name*, car (lr-data-seg-table-body (*flag*, *expr*, *data-seg*, *table*)))

THEOREM: definedp-table-definedp-car-lr-data-seg-table-list
definedp (*name*, *data-seg*)
$\rightarrow$ definedp (*name*, car (lr-data-seg-table-list (*progs*, *data-seg*, *table*)))

THEOREM: equal-lengths-same-signature-car-lr-compile-quote
same-signature (*data-seg*, car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))
$\rightarrow$ (length (cdr (assoc (*name*,
car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))))
= length (cdr (assoc (*name*, *data-seg*))))

THEOREM: adpp-same-signature-car-lr-compile-quote
same-signature (*data-seg*, car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))
$\rightarrow$ (adpp (*adp*, car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))
= adpp (*adp*, *data-seg*))

THEOREM: same-signature-car-lr-compile-quote-helper
let *pair* **be** lr-compile-quote ('`list`,
list (car (*object*), cdr (*object*)),
data-seg,
table)
in
(lr-proper-free-listp (car (*pair*))
$\wedge$ same-signature (*data-seg*, car (*pair*))
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\wedge$ ((length (cdr (assoc (LR-HEAP-NAME, *data-seg*))) $-$ 1)
$\not<$ (offset (fetch (LR-FP-ADDR, car (*pair*))) $+$ length (*list*))))
$\rightarrow$ same-signature (*data-seg*,
deposit-a-list (*list*,
fetch (identity (LR-FP-ADDR),
car (*pair*)),
car (*pair*))) **endlet**

THEOREM: same-signature-car-lr-compile-quote-generalized
(lr-proper-free-listp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)

$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ (same-signature (*data-seg*,
car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))
$\wedge$ lr-proper-free-listp (car (lr-compile-quote (*flag*,
object,
data-seg,
table))))

EVENT: Disable equal-lengths-same-signature-car-lr-compile-quote.

EVENT: Disable adpp-same-signature-car-lr-compile-quote.

THEOREM: lr-proper-free-listp-car-lr-compile-quote
(lr-proper-free-listp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ lr-proper-free-listp (car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))

THEOREM: p-objectp-car-lr-compile-quote
(p-objectp (*object1*,
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
data-seg,
max-ctrl,
max-temp,
word-size,
psw))
$\wedge$ lr-proper-free-listp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ p-objectp (*object1*,
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
car (lr-compile-quote (*flag*, *object2*, *data-seg*, *table*)),
max-ctrl,
max-temp,

word-size,
psw))

THEOREM: lr-proper-p-areasp-car-lr-data-seg-table-body
lr-proper-p-areasp (*data-seg*)
$\rightarrow$ lr-proper-p-areasp (car (lr-data-seg-table-body (*flag*,
expr,
data-seg,
table)))

THEOREM: same-signature-car-lr-compile-quote
(lr-proper-free-listp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ definedp (LR-HEAP-NAME, *data-seg*)
$\land$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ same-signature (*data-seg*,
car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))

THEOREM: same-signature-car-lr-data-seg-table-body
(lr-proper-free-listp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ definedp (LR-HEAP-NAME, *data-seg*)
$\land$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ (same-signature (*data-seg*,
car (lr-data-seg-table-body (*flag*, *expr*, *data-seg*, *table*)))
$\land$ lr-proper-free-listp (car (lr-data-seg-table-body (*flag*,
expr,
data-seg,
table))))

EVENT: Disable same-signature-car-lr-compile-quote.

THEOREM: lr-max-node-car-lr-data-seg-table-body
(lr-proper-free-listp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ definedp (LR-HEAP-NAME, *data-seg*)
$\land$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ (lr-max-node (car (lr-data-seg-table-body (*flag*, *body*, *data-seg*, *table*)))
= lr-max-node (*data-seg*))

THEOREM: same-signature-car-lr-data-seg-table-list-helper
let *dst-body* **be** lr-data-seg-table-body (**t**, s-body (*prog*), *data-seg*, *table*)
in
(same-signature (car (*dst-body*),

car (lr-data-seg-table-list (*progs*,
car (*dst-body*),
cdr (*dst-body*))))
∧ lr-proper-free-listp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*)
∧ definedp (LR-HEAP-NAME, *data-seg*)
∧ lr-boundary-nodep (lr-max-node (*data-seg*)))
→ same-signature (*data-seg*,
car (lr-data-seg-table-list (*progs*,
car (*dst-body*),
cdr (*dst-body*)))) **endlet**

THEOREM: same-signature-car-lr-data-seg-table-list
(lr-proper-free-listp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*)
∧ definedp (LR-HEAP-NAME, *data-seg*)
∧ lr-boundary-nodep (lr-max-node (*data-seg*)))
→ same-signature (*data-seg*,
car (lr-data-seg-table-list (*progs*, *data-seg*, *table*)))

EVENT: Disable same-signature-car-lr-data-seg-table-list-helper.

THEOREM: length-car-lr-compile-quote
(lr-proper-free-listp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*)
∧ definedp (LR-HEAP-NAME, *data-seg*)
∧ lr-boundary-nodep (lr-max-node (*data-seg*)))
→ (length (cdr (assoc (*name*,
car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))))
= length (cdr (assoc (*name*, *data-seg*))))

THEOREM: lr-max-node-car-lr-compile-quote
(lr-proper-free-listp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*)
∧ definedp (LR-HEAP-NAME, *data-seg*)
∧ lr-boundary-nodep (lr-max-node (*data-seg*)))
→ (lr-max-node (car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))
= lr-max-node (*data-seg*))

THEOREM: lr-proper-free-listp-car-lr-init-data-seg-table
(lr-proper-free-listp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*)
∧ definedp (LR-HEAP-NAME, *data-seg*)
∧ lr-boundary-nodep (lr-max-node (*data-seg*)))
→ lr-proper-free-listp (car (lr-init-data-seg-table (*params*, *data-seg*, *table*)))

THEOREM: adpp-untag-lr-fp-addr-lr-init-data-seg
adpp (identity (untag (LR-FP-ADDR)), lr-init-data-seg (*heap-size*))

THEOREM: lr-max-node-lr-init-data-seg
$(\textit{heap-size} \not< 2)$
$\rightarrow$ (lr-max-node (lr-init-data-seg (*heap-size*))
 $=$ tag ('addr,
  cons (identity (LR-HEAP-NAME),
   identity (LR-NODE-SIZE) $*$ *heap-size*)))

THEOREM: fetch-lr-fp-addr-lr-init-data-seg
fetch (identity (LR-FP-ADDR), lr-init-data-seg (*heap-size*))
$=$ identity (add-addr (LR-F-ADDR, LR-NODE-SIZE))

THEOREM: lr-boundary-nodep-not-lessp-fact-helper
$((x < (y * z)) \land ((x \textbf{ mod } y) = 0) \land (x \in \mathbf{N}))$
$\rightarrow ((x < (y * (z - 1))) = (x \neq (y * (z - 1))))$

THEOREM: lessp-times-difference-fact
$((z \not\simeq 0) \land (x \not\simeq 0) \land ((x \textbf{ mod } y) = 0))$
$\rightarrow (((x - y) < (y * (z - 1))) = (x < (y * z)))$

THEOREM: lessp-times-difference-node-on-boundaryp-fact
$((\textit{heap-size} \not\simeq 0) \land (\text{offset}(\textit{addr}) \not\simeq 0) \land \text{lr-boundary-nodep}(\textit{addr}))$
$\rightarrow ((((((\text{offset}(\textit{addr}) - 1) - 1) - 1) - 1)$
 $< (\text{identity}(\text{LR-NODE-SIZE}) * (\textit{heap-size} - 1)))$
 $= (\text{offset}(\textit{addr}) < (\text{identity}(\text{LR-NODE-SIZE}) * \textit{heap-size})))$

THEOREM: lr-boundary-nodep-lessp-lr-node-size-0
lr-boundary-nodep (*addr*)
$\rightarrow ((((\text{offset}(\textit{addr}) - 1) = 1) = \mathbf{f})$
 $\land ((((\text{offset}(\textit{addr}) - 1) - 1) = 1) = \mathbf{f}))$

THEOREM: lr-boundary-nodep-lessp-lr-node-size-1
$((\text{offset}(\textit{addr}) \in \mathbf{N}) \land \text{lr-boundary-nodep}(\textit{addr}) \land (n < \text{LR-NODE-SIZE}))$
$\rightarrow ((n < \text{offset}(\textit{addr})) = (\text{offset}(\textit{addr}) \neq 0))$

THEOREM: lr-boundary-nodep-lessp-lr-node-size-2
lr-boundary-nodep (*addr*) $\rightarrow ((\text{offset}(\textit{addr}) = 1) = \mathbf{f})$

DEFINITION:
induct-hint-17 (*addr1*, *size*, *addr2*)
$=$ **if** *size* $\simeq 0$ **then t**
 elseif offset (*addr2*) $\simeq 0$ **then t**
 else induct-hint-17 (add-addr (*addr1*, LR-NODE-SIZE),
  size $- 1$,
  sub-addr (*addr2*, LR-NODE-SIZE)) **endif**

THEOREM: get-cdr-lr-init-heap-contents
((offset (*addr2*) < (LR-NODE-SIZE $*$ *heap-size*))
 $\land$ lr-boundary-nodep (*addr2*)
 $\land$ (offset (*addr2*) $\in$ **N**)
 $\land$ (offset (*addr1*) $\in$ **N**))
$\rightarrow$ (get (offset (*addr2*), cdr (lr-init-heap-contents (*addr1*, *heap-size*)))
 = add-addr (add-addr (*addr1*, offset (*addr2*)), LR-NODE-SIZE))

EVENT: Disable lr-boundary-nodep-lessp-lr-node-size-0.

EVENT: Disable lr-boundary-nodep-lessp-lr-node-size-1.

EVENT: Disable lr-boundary-nodep-lessp-lr-node-size-2.

THEOREM: length-cdr-assoc-lr-heap-name-lr-init-data-seg
(*heap-size* $\not<$ 2)
$\rightarrow$ (length (cdr (assoc (identity (LR-HEAP-NAME), lr-init-data-seg (*heap-size*))))
 = (1 + (*heap-size* $*$ identity (LR-NODE-SIZE))))

THEOREM: fetch-add-addr-ref-count-offset-lr-init-data-seg-help-1
((offset (*addr*) = 0)
 $\land$ (type (*addr*) = `'addr`)
 $\land$ (area-name (*addr*) = `'heap`))
$\rightarrow$ (add-addr (*addr*, 4) = `'(addr (heap . 4))`)

THEOREM: equal-add-addr-fact
(type (*addr1*) = type (*addr2*))
$\rightarrow$ ((add-addr (*addr1*, *n1*) = add-addr (*addr2*, *n2*))
 = ((area-name (*addr1*) = area-name (*addr2*))
 $\land$ ((offset (*addr1*) + *n1*) = (offset (*addr2*) + *n2*))))

DEFINITION:
lr-all-nodes (*min-offset*, *max-addr*)
= **if** offset (*max-addr*) $\simeq$ 0 **then nil**
 elseif *min-offset* $\not<$ offset (*max-addr*) **then nil**
 else cons (sub-addr (*max-addr*, LR-NODE-SIZE),
 lr-all-nodes (*min-offset*,
 sub-addr (*max-addr*, LR-NODE-SIZE))) **endif**

DEFINITION:
induct-hint-19 (*addr*, *max-addr*)
= **if** offset (*addr*) < offset (*max-addr*)
 then induct-hint-19 (add-addr (*addr*, LR-NODE-SIZE), *max-addr*)
 else t endif

THEOREM: lessp-times-plus-fact
$(n \not\simeq 0) \rightarrow (((n * v) < (n + (n * w))) = (v < (1 + w)))$

THEOREM: lessp-sub1-lessp-fact
$((x \in \mathbf{N}) \wedge (y \in \mathbf{N}) \wedge (x \neq 0) \wedge (x \neq y))$
$\rightarrow \quad (((x - 1) < y) = (x < y))$

THEOREM: remainder-difference-not-equal-lessp-fact
$(((x \;\mathbf{mod}\; n) = 0)$
$\wedge \quad ((y \;\mathbf{mod}\; n) = 0)$
$\wedge \quad (x \neq (y - n))$
$\wedge \quad (y \not< n)$
$\wedge \quad (x \in \mathbf{N})$
$\wedge \quad (y \in \mathbf{N}))$
$\rightarrow \quad ((x < (y - n)) = (x < y))$

EVENT: Disable lessp-sub1-lessp-fact.

THEOREM: lr-boundaryp-nodep-difference-node-size
lr-boundary-offsetp (offset)
$\rightarrow$ lr-boundary-offsetp $(\mathit{offset} - \text{LR-NODE-SIZE})$

THEOREM: lr-boundary-offsetp-difference-not-equal-lessp-fact-1
(lr-boundary-offsetp (x)
$\wedge$ lr-boundary-offsetp (y)
$\wedge \quad (x \neq (y - \text{LR-NODE-SIZE}))$
$\wedge \quad (y \not< \text{LR-NODE-SIZE})$
$\wedge \quad (x \in \mathbf{N})$
$\wedge \quad (y \in \mathbf{N}))$
$\rightarrow \quad ((x < (y - \text{LR-NODE-SIZE})) = (x < y))$

THEOREM: member-lr-all-nodes-helper
((offset $(\mathit{max\text{-}addr}) \not\simeq 0$)
$\wedge$ (offset $(\mathit{addr}) \in \mathbf{N}$)
$\wedge$ (cddr $(\mathit{addr}) = \mathbf{nil}$)
$\wedge$ listp (addr)
$\wedge$ listp (untag (addr))
$\wedge$ lr-boundary-nodep (addr)
$\wedge$ lr-boundary-nodep $(\mathit{max\text{-}addr})$
$\wedge$ (area-name (addr) = area-name $(\mathit{max\text{-}addr})$)
$\wedge$ (type (addr) = type $(\mathit{max\text{-}addr})$)
$\wedge$ $(\mathit{addr} \neq$ sub-addr $(\mathit{max\text{-}addr}, \text{LR-NODE-SIZE})))$
$\rightarrow$ ((offset (addr) < ((((offset $(\mathit{max\text{-}addr})$ − 1) − 1) − 1) − 1))
$=$ (offset (addr) < offset $(\mathit{max\text{-}addr})$)))

THEOREM: member-lr-all-nodes

$((\text{type}(addr) = \texttt{'addr})$
$\wedge \quad (\text{cddr}(addr) = \textbf{nil})$
$\wedge \quad \text{listp}(addr)$
$\wedge \quad \text{lr-boundary-nodep}(addr)$
$\wedge \quad (\text{area-name}(addr) = \text{LR-HEAP-NAME})$
$\wedge \quad \text{listp}(\text{untag}(addr))$
$\wedge \quad (\text{offset}(addr) \in \mathbf{N})$
$\wedge \quad (\text{type}(max\text{-}addr) = \texttt{'addr})$
$\wedge \quad (\text{cddr}(max\text{-}addr) = \textbf{nil})$
$\wedge \quad \text{listp}(max\text{-}addr)$
$\wedge \quad \text{lr-boundary-nodep}(max\text{-}addr)$
$\wedge \quad (\text{area-name}(max\text{-}addr) = \text{LR-HEAP-NAME})$
$\wedge \quad \text{listp}(\text{untag}(max\text{-}addr))$
$\wedge \quad \text{lr-boundary-offsetp}(min\text{-}offset)$
$\wedge \quad (\text{offset}(addr) \not< min\text{-}offset))$
$\rightarrow \quad ((addr \in \text{lr-all-nodes}(min\text{-}offset,\ max\text{-}addr))$
$\quad = \quad (\text{offset}(addr) < \text{offset}(max\text{-}addr)))$

EVENT: Disable member-lr-all-nodes-helper.

THEOREM: lr-all-nodes-nil

$(\text{lr-all-nodes}(min\text{-}offset,\ max\text{-}addr) = \textbf{nil})$
$= \quad ((\text{offset}(max\text{-}addr) \simeq 0) \vee (min\text{-}offset \not< \text{offset}(max\text{-}addr)))$

THEOREM: delete-append

$\text{delete}(e,\ \text{append}(x,\ y))$
$= \quad \textbf{if } e \in x \textbf{ then } \text{append}(\text{delete}(e,\ x),\ y)$
$\quad \textbf{else } \text{append}(x,\ \text{delete}(e,\ y)) \textbf{ endif}$

THEOREM: lessp-difference-node-size-sub-addr-2

$((offset < \text{offset}(addr))$
$\wedge \quad \text{lr-boundary-nodep}(addr)$
$\wedge \quad (\text{offset}(addr) \in \mathbf{N})$
$\wedge \quad \text{lr-boundary-offsetp}(offset))$
$\rightarrow \quad (((\text{offset}(addr) - \text{identity}(\text{LR-NODE-SIZE})) < offset) = \mathbf{f})$

THEOREM: not-member-lr-all-nodes-too-small-addr

$(\text{lr-boundary-nodep}(addr)$
$\wedge \quad \text{lr-boundary-nodep}(max\text{-}addr)$
$\wedge \quad \text{lr-boundary-offsetp}(min\text{-}offset)$
$\wedge \quad (\text{offset}(addr) < min\text{-}offset)$
$\wedge \quad (min\text{-}offset \in \mathbf{N}))$
$\rightarrow \quad (addr \notin \text{lr-all-nodes}(min\text{-}offset,\ max\text{-}addr))$

THEOREM: plist-delete
$\text{plist}\,(\text{delete}\,(e,\ x)) = \text{delete}\,(e,\ \text{plist}\,(x))$

THEOREM: lr-check-free-nodes-plist-node-list
$\text{lr-check-free-nodes}\,(addr,\ \text{plist}\,(node\text{-}list),\ data\text{-}seg,\ max\text{-}addr)$
$= \quad \text{lr-check-free-nodes}\,(addr,\ node\text{-}list,\ data\text{-}seg,\ max\text{-}addr)$

THEOREM: lr-all-nodes-offset-same-max
$\text{lr-all-nodes}\,(\text{offset}\,(addr),\ addr) = \mathbf{nil}$

THEOREM: lr-all-nodes-offset-max-addr-opener-helper
$((\text{offset}\,(addr) \not\simeq \texttt{0})$
$\wedge \quad \text{lr-boundary-nodep}\,(addr)$
$\wedge \quad (offset \in \mathbf{N})$
$\wedge \quad \text{lr-boundary-offsetp}\,(offset)$
$\wedge \quad (offset < \text{offset}\,(addr)))$
$\rightarrow \quad ((offset < ((((\text{offset}\,(addr) - 1) - 1) - 1) - 1))$
$\qquad = \quad (offset \neq ((((\text{offset}\,(addr) - 1) - 1) - 1) - 1)))$

THEOREM: lr-all-nodes-lessp-max-addr-opener
$((\text{type}\,(max\text{-}addr) = \texttt{'addr})$
$\wedge \quad \text{listp}\,(max\text{-}addr)$
$\wedge \quad (\text{cddr}\,(max\text{-}addr) = \mathbf{nil})$
$\wedge \quad \text{listp}\,(\text{untag}\,(max\text{-}addr))$
$\wedge \quad (\text{offset}\,(max\text{-}addr) \in \mathbf{N})$
$\wedge \quad \text{lr-boundary-nodep}\,(max\text{-}addr)$
$\wedge \quad (\text{area-name}\,(max\text{-}addr) = \text{LR-HEAP-NAME})$
$\wedge \quad (min\text{-}offset < \text{offset}\,(max\text{-}addr))$
$\wedge \quad (min\text{-}offset \in \mathbf{N})$
$\wedge \quad \text{lr-boundary-offsetp}\,(min\text{-}offset))$
$\rightarrow \quad (\text{lr-all-nodes}\,(min\text{-}offset,\ max\text{-}addr)$
$\qquad = \quad \text{append}\,(\text{lr-all-nodes}\,(min\text{-}offset + \text{identity}\,(\text{LR-NODE-SIZE}),$
$\qquad\qquad\qquad max\text{-}addr),$
$\qquad\qquad \text{list}\,(\text{tag}\,(\texttt{'addr},$
$\qquad\qquad\qquad \text{cons}\,(\text{identity}\,(\text{LR-HEAP-NAME}),\ min\text{-}offset)))))$

THEOREM: fetch-init-init-data-seg-generalized
$((\text{offset}\,(addr) \in \mathbf{N})$
$\wedge \quad (\text{type}\,(addr) = \texttt{'addr})$
$\wedge \quad (\text{cddr}\,(addr) = \mathbf{nil})$
$\wedge \quad \text{listp}\,(addr)$
$\wedge \quad \text{listp}\,(\text{untag}\,(addr))$
$\wedge \quad \text{lr-boundary-nodep}\,(addr)$
$\wedge \quad (\text{area-name}\,(addr) = \text{LR-HEAP-NAME})$
$\wedge \quad (\text{offset}\,(addr) < (\text{identity}\,(\text{LR-NODE-SIZE}) * heap\text{-}size))$

$\wedge$ (cdr (assoc (LR-HEAP-NAME, *data-seg*))
= lr-init-heap-contents (identity (tag (`'addr`,
cons (LR-HEAP-NAME, `0`))),
heap-size)))
$\rightarrow$ (fetch (add-addr (*addr*, identity (LR-REF-COUNT-OFFSET)), *data-seg*)
= add-addr (*addr*, `4`))

THEOREM: lessp-difference-node-size-sub-addr-3
((offset (*addr*) < (LR-NODE-SIZE $*$ *heap-size*))
$\wedge$ lr-boundary-nodep (*addr*)
$\wedge$ (offset (*addr*) $\in$ **N**))
$\rightarrow$ (((((((identity (LR-NODE-SIZE) $*$ *heap-size*) $-$ 1) $-$ 1) $-$ 1) $-$ 1)
< offset (*addr*))
= **f**)

THEOREM: lr-boundary-nodep-tag-cons-times-lr-node-size
lr-boundary-nodep (tag (x, cons (*name*, identity (LR-NODE-SIZE) $*$ *heap-size*)))

THEOREM: tag-type-name-offset-equal-same
((type (*addr*) = x)
$\wedge$ (cddr (*addr*) = **nil**)
$\wedge$ listp (untag (*addr*))
$\wedge$ (offset (*addr*) $\in$ **N**)
$\wedge$ (area-name (*addr*) = *name*))
$\rightarrow$ (tag (x, cons (*name*, offset (*addr*))) = *addr*)

THEOREM: lr-check-free-nodes-lr-free-list-nodes-init-data-seg
let *init-data-seg* **be** list (cons (area-name (LR-FP-ADDR), *any1*),
cons (area-name (LR-ANSWER-ADDR), *any2*),
cons (LR-HEAP-NAME,
lr-init-heap-contents (tag (`'addr`,
cons (LR-HEAP-NAME,
`0`)),
heap-size)))
in
((offset (*max-addr*) $\not<$ offset (*addr*))
$\wedge$ (type (*addr*) = `'addr`)
$\wedge$ (cddr (*addr*) = **nil**)
$\wedge$ listp (*addr*)
$\wedge$ listp (untag (*addr*))
$\wedge$ (offset (*addr*) $\in$ **N**)
$\wedge$ lr-boundary-nodep (*addr*)
$\wedge$ (area-name (*addr*) = LR-HEAP-NAME)
$\wedge$ (*max-addr* = lr-max-node (*init-data-seg*)))
$\rightarrow$ lr-check-free-nodes (*addr*,

```
                    lr-all-nodes (offset (addr), max-addr),
                    list (cons (identity (area-name (LR-FP-ADDR)),
                                any1),
                          cons (identity (area-name (LR-ANSWER-ADDR)),
                                any2),
                          cons (identity (LR-HEAP-NAME),
                                lr-init-heap-contents (identity (tag ('addr,
                                                                    cons (LR-HEAP-NAME,
                                                                          0))),
                                                       heap-size))),
                    max-addr) endlet
```

EVENT: Disable fetch-init-init-data-seg-generalized.

THEOREM: lr-free-list-nodes-deposit-a-list-lr-nodep

```
((type (addr) = 'addr)
 ∧  (cddr (addr) = nil)
 ∧  listp (addr)
 ∧  adpp (untag (addr), data-seg)
 ∧  lr-boundary-nodep (addr)
 ∧  (area-name (addr) = LR-HEAP-NAME)
 ∧  (type (max-addr) = 'addr)
 ∧  (cddr (max-addr) = nil)
 ∧  listp (max-addr)
 ∧  adpp (untag (max-addr), data-seg)
 ∧  lr-boundary-nodep (max-addr)
 ∧  (area-name (max-addr) = LR-HEAP-NAME))
 →  (lr-free-list-nodes (max-addr,
                         deposit-a-list (list (a, b, c, d), addr, data-seg))
       =  lr-free-list-nodes (max-addr,
                              deposit (b,
                                       add-addr (addr,
                                                 identity (LR-REF-COUNT-OFFSET)),
                                       data-seg)))
```

THEOREM: lr-check-free-nodes-deposit-a-list-lr-nodep

```
((type (addr) = 'addr)
 ∧  (cddr (addr) = nil)
 ∧  listp (addr)
 ∧  adpp (untag (addr), data-seg)
 ∧  lr-boundary-nodep (addr)
 ∧  (area-name (addr) = LR-HEAP-NAME)
 ∧  (type (max-addr) = 'addr)
 ∧  (cddr (max-addr) = nil)
```

$\wedge$ listp (*max-addr*)
$\wedge$ adpp (untag (*max-addr*), *data-seg*)
$\wedge$ lr-boundary-nodep (*max-addr*)
$\wedge$ (area-name (*max-addr*) = LR-HEAP-NAME)
$\wedge$ lr-node-listp (*node-list*, *data-seg*))
$\rightarrow$ (lr-check-free-nodes (*addr1*,
node-list,
deposit-a-list (list (*a*, *b*, *c*, *d*), *addr*, *data-seg*),
max-addr)
= lr-check-free-nodes (*addr1*,
node-list,
deposit (*b*,
add-addr (*addr*,
identity (LR-REF-COUNT-OFFSET)),
data-seg),
max-addr))

THEOREM: lr-all-nodes-not-lessp-min-offset-max-addr
(*min-offset* $\not<$ offset (*max-addr*))
$\rightarrow$ (lr-all-nodes (*min-offset*, *max-addr*) = **nil**)

THEOREM: fetch-init-init-data-seg-sub-addr
(((identity (LR-NODE-SIZE) $*$ *heap-size*) $\not<$ offset (*addr*))
$\wedge$ (offset (*addr*) $\in$ **N**)
$\wedge$ (cddr (*addr*) = **nil**)
$\wedge$ listp (*addr*)
$\wedge$ listp (untag (*addr*))
$\wedge$ (type (*addr*) = `'addr`)
$\wedge$ lr-boundary-nodep (*addr*)
$\wedge$ (area-name (*addr*) = LR-HEAP-NAME)
$\wedge$ ((offset (*addr*) $-$ identity (LR-NODE-SIZE))
$<$ (LR-NODE-SIZE $*$ *heap-size*))
$\wedge$ (offset (*addr*) $\neq$ `0`)
$\wedge$ (cdr (assoc (LR-HEAP-NAME, *data-seg*))
= lr-init-heap-contents (tag (`'addr`, cons (LR-HEAP-NAME, `0`)),
heap-size)))
$\rightarrow$ (fetch (add-addr (sub-addr (*addr*, identity (LR-NODE-SIZE)),
identity (LR-REF-COUNT-OFFSET)),
data-seg)
= *addr*)

THEOREM: lr-free-list-nodes-lr-init-heap-contents-generalized
(lr-boundary-nodep (*max-addr*)
$\wedge$ (area-name (*max-addr*) = LR-HEAP-NAME)
$\wedge$ (type (*max-addr*) = `'addr`)

$\wedge$ (cddr (*max-addr*) = **nil**)
$\wedge$ listp (*max-addr*)
$\wedge$ listp (untag (*max-addr*))
$\wedge$ ((LR-NODE-SIZE $*$ *heap-size*) $\not<$ offset (*max-addr*))
$\wedge$ (cdr (assoc (LR-HEAP-NAME, *data-seg*))
 = lr-init-heap-contents (identity (tag (`'addr`,
 cons (LR-HEAP-NAME, `0`))),
 heap-size)))
$\rightarrow$ (lr-free-list-nodes (*max-addr*, *data-seg*) = lr-all-nodes (`0`, *max-addr*))

THEOREM: lr-free-list-nodes-lr-init-heap-contents
(lr-boundary-nodep (*max-addr*)
$\wedge$ (area-name (*max-addr*) = LR-HEAP-NAME)
$\wedge$ (type (*max-addr*) = `'addr`)
$\wedge$ (cddr (*max-addr*) = **nil**)
$\wedge$ listp (*max-addr*)
$\wedge$ listp (untag (*max-addr*))
$\wedge$ ((LR-NODE-SIZE $*$ *heap-size*) $\not<$ offset (*max-addr*)))
$\rightarrow$ (lr-free-list-nodes (*max-addr*,
 list (cons (identity (area-name (LR-FP-ADDR)), *any1*),
 cons (identity (area-name (LR-ANSWER-ADDR)),
 any2),
 cons (identity (LR-HEAP-NAME),
 lr-init-heap-contents (identity (tag (`'addr`,
 cons (LR-HEAP-NAME,
 `0`))),
 heap-size))))
 = lr-all-nodes (`0`, *max-addr*))

THEOREM: lr-node-listp-lr-all-nodes
(lr-boundary-nodep (*addr*)
$\wedge$ (area-name (*addr*) = LR-HEAP-NAME)
$\wedge$ adpp (untag (*addr*), *data-seg*)
$\wedge$ (type (*addr*) = `'addr`))
$\rightarrow$ lr-node-listp (lr-all-nodes (*min-offset*, *addr*), *data-seg*)

THEOREM: plistp-lr-all-nodes
plistp (lr-all-nodes (*min-offset*, *max-addr*))

THEOREM: lr-free-list-nodes-lr-init-data-seg
(*heap-size* $\not<$ `2`)
$\rightarrow$ (lr-free-list-nodes (tag (`'addr`,
 cons (identity (LR-HEAP-NAME),
 identity (LR-NODE-SIZE) $*$ *heap-size*)),
 lr-init-data-seg (*heap-size*))

$=$ lr-all-nodes (identity (offset (add-addr (LR-F-ADDR, LR-NODE-SIZE))),
tag ('**addr**,
cons (identity (LR-HEAP-NAME),
identity (LR-NODE-SIZE) $*$ *heap-size*))))

THEOREM: lr-proper-free-listp-lr-init-data-seg-helper
(*heap-size* $\not<$ 2)
$\rightarrow$ lr-check-free-nodes (identity (add-addr (LR-F-ADDR, LR-NODE-SIZE)),
lr-all-nodes (identity (offset (add-addr (LR-F-ADDR,
LR-NODE-SIZE))),
tag ('**addr**,
cons (identity (LR-HEAP-NAME),
identity (LR-NODE-SIZE)
$*$ *heap-size*))),
lr-init-data-seg (*heap-size*),
tag ('**addr**,
cons (identity (LR-HEAP-NAME),
identity (LR-NODE-SIZE) $*$ *heap-size*)))

THEOREM: lr-proper-free-listp-lr-init-data-seg
(*heap-size* $\not<$ 2) $\rightarrow$ lr-proper-free-listp (lr-init-data-seg (*heap-size*))

THEOREM: definedp-lr-heap-name-lr-init-data-seg
definedp (identity (LR-HEAP-NAME), lr-init-data-seg (*heap-size*))

THEOREM: lr-proper-p-areasp-lr-heap-name-lr-init-data-seg
lr-proper-p-areasp (lr-init-data-seg (*heap-size*))

THEOREM: lr-proper-p-areasp-car-lr-init-data-seg-table
lr-proper-p-areasp (*data-seg*)
$\rightarrow$ lr-proper-p-areasp (car (lr-init-data-seg-table (*params*, *data-seg*, *table*)))

THEOREM: lr-proper-p-areasp-car-lr-data-seg-table-list
lr-proper-p-areasp (*data-seg*)
$\rightarrow$ lr-proper-p-areasp (car (lr-data-seg-table-list (*progs*, *data-seg*, *table*)))

THEOREM: definedp-table-definedp-car-lr-init-data-seg-table
definedp (*name*, car (lr-init-data-seg-table (*params*, *data-seg*, *table*)))
$=$ definedp (*name*, *data-seg*)

THEOREM: all-p-objects-lookup-cons-table
(all-p-objects-lookup (*list*, *table*, p) $\land$ p-objectp (y, p))
$\rightarrow$ all-p-objects-lookup (*list*, cons (cons (x, y), *table*), p)

THEOREM: p-objectp-opener-alt-lr-proper-free-listp

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-proper-free-listp}\,(\text{p-data-segment}\,(p))} \\
\wedge & \text{adpp}\,(\text{untag}\,(\text{lr-max-node}\,(\textit{data-seg})),\ \textit{data-seg}) \\
\wedge & \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\textit{data-seg})) \\
\wedge & (\textit{data-seg} = \text{p-data-segment}\,(p))) \\
\rightarrow & \text{p-objectp}\,(\text{fetch}\,(\text{identity}\,(\text{LR-FP-ADDR}),\ \textit{data-seg}),\ p)
\end{array}
$$

THEOREM: p-objectp-lookup-deposit-a-list

p-objectp (*object*,
  p-state (*pc*,
    ctrl-stk,
    temp-stk,
    prog-seg,
    data-seg,
    max-ctrl-stk-size,
    max-temp-stk-size,
    word-size,
    psw))
$\rightarrow$ p-objectp (*object*,
  p-state (*pc*,
    ctrl-stk,
    temp-stk,
    prog-seg,
    deposit-a-list (*stuff*, *addr*, *data-seg*),
    max-ctrl-stk-size,
    max-temp-stk-size,
    word-size,
    psw))

THEOREM: all-p-objects-lookup-deposit-a-list

all-p-objects-lookup (*list*,
  table,
  p-state (*pc*,
    ctrl-stk,
    temp-stk,
    prog-seg,
    data-seg,
    max-ctrl-stk-size,
    max-temp-stk-size,
    word-size,
    psw))
$\rightarrow$ all-p-objects-lookup (*list*,
  table,
  p-state (*pc*,

ctrl-stk,
temp-stk,
prog-seg,
deposit-a-list (*stuff*, *addr*, *data-seg*),
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))

THEOREM: p-objectp-lookup-deposit
p-objectp (*object*,
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
data-seg,
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))
$\rightarrow$ p-objectp (*object*,
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
deposit (*anything*, *addr*, *data-seg*),
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))

THEOREM: all-p-objects-lookup-deposit
all-p-objects-lookup (*list*,
table,
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
data-seg,
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))
$\rightarrow$ all-p-objects-lookup (*list*,

table,
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
deposit (*anything*, *addr*, *data-seg*),
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))

THEOREM: definedp-name-p-objectp-tag-0-lr-proper-p-areasp
lr-proper-p-areasp (*data-seg*)
$\rightarrow$ (p-objectp (list (`'addr`, cons (*name*, `0`)),
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
data-seg,
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))
= definedp (*name*, *data-seg*))

THEOREM: all-p-objects-lookup-lr-compile-quote
(all-p-objects-lookup (*list*,
table,
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
data-seg,
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))
$\land$ lr-proper-free-listp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ definedp (LR-HEAP-NAME, *data-seg*)
$\land$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ all-p-objects-lookup (*list*,
cdr (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)),
p-state (*pc*,

ctrl-stk,
temp-stk,
prog-seg,
car (lr-compile-quote (*flag*,
object,
data-seg,
table)),
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))

THEOREM: all-p-objects-lookup-lr-data-seg-table-body
(all-p-objects-lookup (*list*,
table,
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
data-seg,
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))
$\land$ lr-proper-free-listp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ definedp (LR-HEAP-NAME, *data-seg*)
$\land$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ all-p-objects-lookup (*list*,
cdr (lr-data-seg-table-body (*flag*,
body,
data-seg,
table)),
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
car (lr-data-seg-table-body (*flag*,
body,
data-seg,
table)),
max-ctrl-stk-size,
max-temp-stk-size,
word-size,

```
                              psw))
```

THEOREM: all-p-objects-lookup-lr-data-seg-table-list

```
 (all-p-objects-lookup (list,
                        table,
                        p-state (pc,
                                 ctrl-stk,
                                 temp-stk,
                                 prog-seg,
                                 data-seg,
                                 max-ctrl-stk-size,
                                 max-temp-stk-size,
                                 word-size,
                                 psw))
  ∧  lr-proper-free-listp (data-seg)
  ∧  lr-proper-p-areasp (data-seg)
  ∧  definedp (LR-HEAP-NAME, data-seg)
  ∧  lr-boundary-nodep (lr-max-node (data-seg)))
 →   all-p-objects-lookup (list,
                           cdr (lr-data-seg-table-list (progs, data-seg, table)),
                           p-state (pc,
                                    ctrl-stk,
                                    temp-stk,
                                    prog-seg,
                                    car (lr-data-seg-table-list (progs,
                                                                 data-seg,
                                                                 table)),
                                    max-ctrl-stk-size,
                                    max-temp-stk-size,
                                    word-size,
                                    psw))
```

THEOREM: p-objectp-lookup-lr-init-data-seg-table

```
 (p-objectp (object,
             p-state (pc,
                      ctrl-stk,
                      temp-stk,
                      prog-seg,
                      data-seg,
                      max-ctrl-stk-size,
                      max-temp-stk-size,
                      word-size,
                      psw))
  ∧  lr-proper-free-listp (data-seg)
```

$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ p-objectp (*object*,
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
car (lr-init-data-seg-table (*params*, *data-seg*, *table*)),
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))

THEOREM: assoc-definedp-table-lr-compile-quote
definedp (*object1*, *table*)
$\rightarrow$ (assoc (*object1*, cdr (lr-compile-quote (*flag*, *object2*, *data-seg*, *table*)))
= assoc (*object1*, *table*))

THEOREM: assoc-definedp-table-lr-init-data-seg-table
definedp (*object*, *table*)
$\rightarrow$ (assoc (*object*, cdr (lr-init-data-seg-table (*params*, *data-seg*, *table*)))
= assoc (*object*, *table*))

THEOREM: definedp-table-lr-compile-quote-self
(*flag* $\neq$ `'list`)
$\rightarrow$ definedp (*object*, cdr (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))

THEOREM: lr-s-similar-const-table-lr-good-pointerp-opener
(lr-s-similar-const-table (*table*, *data-seg*) $\wedge$ definedp (*object*, *table*))
$\rightarrow$ ((type (cdr (assoc (*object*, *table*))) = **'addr**)
$\wedge$ (cddr (cdr (assoc (*object*, *table*))) = **nil**)
$\wedge$ listp (cdr (assoc (*object*, *table*)))
$\wedge$ adpp (untag (cdr (assoc (*object*, *table*))), *data-seg*)
$\wedge$ lr-boundary-nodep (cdr (assoc (*object*, *table*)))
$\wedge$ (area-name (cdr (assoc (*object*, *table*)))
= identity (LR-HEAP-NAME))
$\wedge$ (type (fetch (add-addr (cdr (assoc (*object*, *table*)),
identity (LR-REF-COUNT-OFFSET)),
data-seg))
= **'nat**))

THEOREM: lr-s-similar-const-table-deposit-lr-fp-addr
(adpp (untag (LR-FP-ADDR), *data-seg*)
$\wedge$ lr-s-similar-const-table (*table*, *data-seg*))

$\rightarrow$ lr-s-similar-const-table (*table*,
deposit (*anything*,
identity (LR-FP-ADDR),
data-seg))

THEOREM: adpp-fetch-lr-fp-addr-car-lr-compile-quote
(lr-proper-free-listp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ adpp (untag (fetch (identity (LR-FP-ADDR),
car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))),
data-seg)

DEFINITION:
lr-good-pointerp-tablep (*table*, *data-seg*)
= **if** listp (*table*)
then lr-good-pointerp (cdar (*table*), *data-seg*)
$\wedge$ lr-good-pointerp-tablep (cdr (*table*), *data-seg*)
else t endif

THEOREM: lr-good-pointerp-tablep-definedp-table
(lr-good-pointerp-tablep (*table*, *data-seg*) $\wedge$ definedp (*object*, *table*))
$\rightarrow$ lr-good-pointerp (cdr (assoc (*object*, *table*)), *data-seg*)

THEOREM: lr-proper-free-listp-opener-2-area-name-alt
(lr-proper-free-listp (*data-seg*)
$\wedge$ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*)))
$\rightarrow$ (car (untag (fetch (identity (LR-FP-ADDR), *data-seg*))) = LR-HEAP-NAME)

THEOREM: lr-good-pointerp-tablep-deposit-free-ptr
lr-good-pointerp-tablep (*table*,
deposit (*anything*, identity (LR-FP-ADDR), *data-seg*))
= lr-good-pointerp-tablep (*table*, *data-seg*)

THEOREM: add1-lr-boundary-nodep
(lr-boundary-nodep (*addr1*) $\wedge$ lr-boundary-nodep (*addr2*))
$\rightarrow$ ((offset (*addr1*) = (1 + offset (*addr2*))) = **f**)

THEOREM: lr-boundary-offsetp-plus
lr-boundary-offsetp (*n*)
$\rightarrow$ (lr-boundary-offsetp ($m + n$) = lr-boundary-offsetp (*m*))

THEOREM: add1-add1-lr-boundary-nodep
(lr-boundary-nodep (*addr1*) $\wedge$ lr-boundary-nodep (*addr2*))
$\rightarrow$ ((offset (*addr1*) = (1 + (1 + offset (*addr2*)))) = **f**)

THEOREM: lr-good-pointerp-tablep-deposit-a-list
(lr-good-pointerp-tablep ($table$, $data\text{-}seg$)
 ∧ (offset ($addr$) ∈ **N**)
 ∧ lr-boundary-nodep ($addr$)
 ∧ (type ($ref\text{-}count$) = `'nat`)
 ∧ (untag ($ref\text{-}count$) ∈ **N**)
 ∧ (area-name ($addr$) = LR-HEAP-NAME)
 ∧ (type (tag) = `'nat`))
→ lr-good-pointerp-tablep ($table$,
                deposit-a-list (list (tag, $ref\text{-}count$, x, y),
                        $addr$,
                        $data\text{-}seg$))

THEOREM: lr-good-pointerp-table-cons
((type ($addr$) = `'addr`)
 ∧ (cddr ($addr$) = **nil**)
 ∧ listp ($addr$)
 ∧ adpp (untag ($addr$), $data\text{-}seg$)
 ∧ lr-boundary-nodep ($addr$)
 ∧ (area-name ($addr$) = LR-HEAP-NAME))
→ (lr-good-pointerp-tablep (cons (cons ($object$, $addr$), $table$), $data\text{-}seg$)
   = ((type (fetch (add-addr ($addr$, identity (LR-REF-COUNT-OFFSET)),
            $data\text{-}seg$))
     = `'nat`)
    ∧ lr-good-pointerp-tablep ($table$, $data\text{-}seg$)))

THEOREM: lr-proper-free-listp-length-sub1-not-lessp
(lr-proper-free-listp ($data\text{-}seg$)
 ∧ lr-proper-p-areasp ($data\text{-}seg$)
 ∧ definedp (LR-HEAP-NAME, $data\text{-}seg$)
 ∧ lr-boundary-nodep (lr-max-node ($data\text{-}seg$)))
→ ((length (cdr (assoc (identity (LR-HEAP-NAME), $data\text{-}seg$))) − 1)
   ≮ offset (fetch (identity (LR-FP-ADDR),
           car (lr-compile-quote ($flag$, $object$, $data\text{-}seg$, $table$)))))

THEOREM: lr-minimum-heapp-lr-compile-quote
(lr-proper-free-listp ($data\text{-}seg$)
 ∧ lr-proper-p-areasp ($data\text{-}seg$)
 ∧ definedp (LR-HEAP-NAME, $data\text{-}seg$)
 ∧ lr-boundary-nodep (lr-max-node ($data\text{-}seg$)))
→ (lr-minimum-heapp (car (lr-compile-quote ($flag$, $object$, $data\text{-}seg$, $table$)))
   = lr-minimum-heapp ($data\text{-}seg$))

DEFINITION:
s-heap-reqs ($flag$, $object$, $data\text{-}seg$, $table$)

```
=   if flag = 'list
    then if listp (object)
         then let pair  be  lr-compile-quote (t,
                                               car (object),
                                               data-seg,
                                               table)
              in
              s-heap-reqs (t, car (object), data-seg, table)
              +   s-heap-reqs ('list,
                               cdr (object),
                               car (pair),
                               cdr (pair)) endlet
         else 0 endif
    elseif definedp (object, table)  then 0
    elseif listp (object)
    then 1 + s-heap-reqs ('list,
                          list (car (object), cdr (object)),
                          data-seg,
                          table)
    elseif object ∈ N  then 1
    elseif truep (object)  then 1
    else 0 endif
```

EVENT: Disable s-heap-reqs.

DEFINITION:
s-heap-reqs-body (*flag*, *expr*, *data-seg*, *table*)

```
=   if flag = 'list
    then if listp (expr)
         then let dst1  be  lr-data-seg-table-body (t,
                                                    car (expr),
                                                    data-seg,
                                                    table)
              in
              s-heap-reqs-body (t, car (expr), data-seg, table)
              +   s-heap-reqs-body ('list,
                                    cdr (expr),
                                    car (dst1),
                                    cdr (dst1)) endlet
         else 0 endif
    elseif listp (expr)
    then if (car (expr) = S-TEMP-FETCH)
            ∨  (car (expr) = S-TEMP-EVAL)
            ∨  (car (expr) = S-TEMP-TEST)
```

```
        then s-heap-reqs-body (t, cadr (expr), data-seg, table)
        elseif car (expr) = 'quote
        then s-heap-reqs (t, cadr (expr), data-seg, table)
        else s-heap-reqs-body ('list, cdr (expr), data-seg, table) endif
    else 0 endif
```

DEFINITION:

```
s-heap-reqs-list (progs, data-seg, table)
=   if listp (progs)
    then s-heap-reqs-body (t, s-body (car (progs)), data-seg, table)
         +   s-heap-reqs-list (cdr (progs),
                               car (lr-data-seg-table-body (t,
                                                            s-body (car (progs)),
                                                            data-seg,
                                                            table)),
                               cdr (lr-data-seg-table-body (t,
                                                            s-body (car (progs)),
                                                            data-seg,
                                                            table)))
    else 0 endif
```

DEFINITION:

```
s-init-heap-reqs (params, data-seg, table)
=   if listp (params)
    then s-heap-reqs (t, cdar (params), data-seg, table)
         +   s-init-heap-reqs (cdr (params),
                               car (lr-compile-quote (t,
                                                      cdar (params),
                                                      data-seg,
                                                      table)),
                               cdr (lr-compile-quote (t,
                                                      cdar (params),
                                                      data-seg,
                                                      table)))
    else 0 endif
```

DEFINITION:

```
s-total-heap-reqs (progs, params, heap-size)
=   let init-ds-table1 be lr-compile-quote ('list,
                                            list (t, 0),
                                            lr-init-data-seg (heap-size),
                                            list (cons (f, LR-F-ADDR)))
    in
    let init-ds-table2 be lr-init-data-seg-table (params,
                                                  car (init-ds-table1),
```

```
                                        cdr (init-ds-table1))
   in
   2
   +  s-heap-reqs ('list,
                  list (t, 0),
                  lr-init-data-seg (heap-size),
                  list (cons (f, LR-F-ADDR)))
   +  s-init-heap-reqs (params,
                       car (init-ds-table1),
                       cdr (init-ds-table1))
   +  s-heap-reqs-list (progs,
                       car (init-ds-table2),
                       cdr (init-ds-table2)) endlet endlet
```

DEFINITION:

```
s-ws-reqs (flag, object, data-seg, table)
=  if flag = 'list
   then if listp (object)
        then let pair  be lr-compile-quote (t,
                                            car (object),
                                            data-seg,
                                            table)
             in
             max (s-ws-reqs (t, car (object), data-seg, table),
                  s-ws-reqs ('list,
                             cdr (object),
                             car (pair),
                             cdr (pair))) endlet
        else 0 endif
   elseif definedp (object, table)  then 0
   elseif listp (object)
   then max (log (2, LR-CONS-TAG),
             s-ws-reqs ('list,
                        list (car (object), cdr (object)),
                        data-seg,
                        table))
   elseif object ∈ N  then max (log (2, LR-ADD1-TAG), log (2, object))
   elseif truep (object)  then log (2, LR-TRUE-TAG)
   elseif falsep (object)  then log (2, LR-FALSE-TAG)
   else 0 endif
```

EVENT: Disable s-ws-reqs.

DEFINITION:

```
s-ws-reqs-body (flag, expr, data-seg, table)
=   if flag = 'list
    then if listp (expr)
         then let dst1  be  lr-data-seg-table-body (t,
                                                    car (expr),
                                                    data-seg,
                                                    table)
              in
              max (s-ws-reqs-body (t, car (expr), data-seg, table),
                   s-ws-reqs-body ('list,
                                   cdr (expr),
                                   car (dst1),
                                   cdr (dst1))) endlet
         else 0 endif
    elseif listp (expr)
    then if (car (expr) = S-TEMP-FETCH)
            ∨  (car (expr) = S-TEMP-EVAL)
            ∨  (car (expr) = S-TEMP-TEST)
         then s-ws-reqs-body (t, cadr (expr), data-seg, table)
         elseif car (expr) = 'quote
         then s-ws-reqs (t, cadr (expr), data-seg, table)
         else s-ws-reqs-body ('list, cdr (expr), data-seg, table) endif
    else 0 endif
```

DEFINITION:

```
s-ws-reqs-list (progs, data-seg, table)
=   if listp (progs)
    then max (s-ws-reqs-body (t, s-body (car (progs)), data-seg, table),
              s-ws-reqs-list (cdr (progs),
                              car (lr-data-seg-table-body (t,
                                                           s-body (car (progs)),
                                                           data-seg,
                                                           table)),
                              cdr (lr-data-seg-table-body (t,
                                                           s-body (car (progs)),
                                                           data-seg,
                                                           table))))
    else 0 endif
```

DEFINITION:

```
s-init-ws-reqs (params, data-seg, table)
=   if listp (params)
    then max (s-ws-reqs (t, cdar (params), data-seg, table),
              s-init-ws-reqs (cdr (params),
```

```
                                  car (lr-compile-quote (t,
                                                        cdar (params),
                                                        data-seg,
                                                        table)),
                                  cdr (lr-compile-quote (t,
                                                        cdar (params),
                                                        data-seg,
                                                        table))))
    else 0 endif
```

DEFINITION:
s-total-ws-reqs (*progs*, *params*, *heap-size*)

```
=   let init-ds-table1 be lr-compile-quote ('list,
                                            list (t, 0),
                                            lr-init-data-seg (heap-size),
                                            list (cons (f, LR-F-ADDR)))
    in
    let init-ds-table2 be lr-init-data-seg-table (params,
                                                  car (init-ds-table1),
                                                  cdr (init-ds-table1))
    in
    max (s-ws-reqs ('list,
                    list (f, t, 0),
                    lr-init-data-seg (heap-size),
                    nil),
         max (s-init-ws-reqs (params,
                              car (init-ds-table1),
                              cdr (init-ds-table1)),
              max (s-ws-reqs-list (progs,
                                   car (init-ds-table2),
                                   cdr (init-ds-table2)),
                   S-MAX-SUBR-REQS))) endlet endlet
```

DEFINITION:
s-restricted-objectp (*flag*, *object*)

```
=   if flag = 'list
    then if listp (object)
         then s-restricted-objectp (t, car (object))
              ∧  s-restricted-objectp ('list, cdr (object))
         else t endif
    elseif object = t then t
    elseif object = f then t
    elseif listp (object)
    then s-restricted-objectp ('list, list (car (object), cdr (object)))
```

elseif *object* $\in$ **N** **then t**
else f endif

DEFINITION:
s-data-seg-body-restrictedp (*flag*, *expr*)
= **if** *flag* = `'list`
then if listp (*expr*)
then s-data-seg-body-restrictedp (**t**, car (*expr*))
$\wedge$ s-data-seg-body-restrictedp (`'list`, cdr (*expr*))
else t endif
elseif listp (*expr*)
then if (car (*expr*) = S-TEMP-FETCH)
$\vee$ (car (*expr*) = S-TEMP-EVAL)
$\vee$ (car (*expr*) = S-TEMP-TEST)
then s-data-seg-body-restrictedp (**t**, cadr (*expr*))
elseif car (*expr*) = `'quote`
then s-restricted-objectp (**t**, cadr (*expr*))
else s-data-seg-body-restrictedp (`'list`, cdr (*expr*)) **endif**
else t endif

DEFINITION:
s-data-seg-list-restrictedp (*progs*)
= **if** listp (*progs*)
then s-data-seg-body-restrictedp (**t**, s-body (car (*progs*)))
$\wedge$ s-data-seg-list-restrictedp (cdr (*progs*))
else t endif

DEFINITION:
s-init-data-seg-restrictedp (*params*)
= **if** listp (*params*)
then s-restricted-objectp (**t**, cdar (*params*))
$\wedge$ s-init-data-seg-restrictedp (cdr (*params*))
else t endif

DEFINITION:
s-restrictedp (*progs*, *params*)
= (s-init-data-seg-restrictedp (*params*)
$\wedge$ s-data-seg-list-restrictedp (*progs*))

THEOREM: lr-minimum-heapp-not-equal-length-1
lr-minimum-heapp (*data-seg*)
$\rightarrow$ (length (cdr (assoc (identity (LR-HEAP-NAME), *data-seg*))) $\neq$ `1`)

THEOREM: lr-count-free-nodes-at-most
length (*node-list*) $\not<$ lr-count-free-nodes (*addr*, *node-list*, *data-seg*)

THEOREM: lr-proper-free-listp-lr-count-free-nodes-max-addr

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-proper-free-listp}\,(\textit{data-seg})} \\
\wedge & \text{definedp}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}) \\
\wedge & \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\textit{data-seg})) \\
\wedge & \text{lr-proper-p-areasp}\,(\textit{data-seg}) \\
\wedge & (\text{length}\,(\text{cdr}\,(\text{assoc}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}))) \\
 & \quad = \quad (1 + \text{offset}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \textit{data-seg})))) \\
\wedge & (\textit{max-addr} = \text{lr-max-node}\,(\textit{data-seg}))) \\
\rightarrow & (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{identity}\,(\text{LR-FP-ADDR}),\ \textit{data-seg}), \\
 & \qquad\qquad\qquad\quad \text{lr-free-list-nodes}\,(\textit{max-addr},\ \textit{data-seg}), \\
 & \qquad\qquad\qquad\quad \textit{data-seg}) \\
 & \quad = \quad 0)
\end{array}
$$

EVENT: Disable lr-count-free-nodes-at-most.

THEOREM: lr-count-free-nodes-deposit-a-list-lr-nodep

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{type}\,(\textit{addr1}) = \texttt{'addr})} \\
\wedge & (\text{cddr}\,(\textit{addr1}) = \textbf{nil}) \\
\wedge & \text{listp}\,(\textit{addr1}) \\
\wedge & \text{adpp}\,(\text{untag}\,(\textit{addr1}),\ \textit{data-seg}) \\
\wedge & \text{lr-boundary-nodep}\,(\textit{addr1}) \\
\wedge & (\text{area-name}\,(\textit{addr1}) = \text{LR-HEAP-NAME}) \\
\wedge & (\text{type}\,(\textit{ref-count}) = \texttt{'nat}) \\
\wedge & (\text{untag}\,(\textit{ref-count}) \in \mathbf{N}) \\
\wedge & \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\textit{data-seg})) \\
\wedge & \text{adpp}\,(\text{untag}\,(\text{lr-max-node}\,(\textit{data-seg})),\ \textit{data-seg}) \\
\wedge & \text{lr-node-listp}\,(\textit{node-list},\ \textit{data-seg}) \\
\wedge & (\textit{addr1} \notin \textit{node-list})) \\
\rightarrow & (\text{lr-count-free-nodes}\,(\textit{addr2}, \\
 & \qquad\qquad\qquad\quad \textit{node-list}, \\
 & \qquad\qquad\qquad\quad \text{deposit-a-list}\,(\text{list}\,(x,\ \textit{ref-count},\ y,\ z), \\
 & \qquad\qquad\qquad\qquad\qquad\qquad \textit{addr1}, \\
 & \qquad\qquad\qquad\qquad\qquad\qquad \textit{data-seg})) \\
 & \quad = \quad \text{lr-count-free-nodes}\,(\textit{addr2},\ \text{delete}\,(\textit{addr1},\ \textit{node-list}),\ \textit{data-seg}))
\end{array}
$$

THEOREM: lr-proper-free-listp-member-free-addr-lr-free-list-nodes

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{adpp}\,(\text{untag}\,(\text{lr-max-node}\,(\textit{data-seg})),\ \textit{data-seg})} \\
\wedge & \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\textit{data-seg})) \\
\wedge & (\textit{max-addr} = \text{lr-max-node}\,(\textit{data-seg})) \\
\wedge & (\text{offset}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \textit{data-seg})) \\
 & \quad < \quad (\text{length}\,(\text{cdr}\,(\text{assoc}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}))) - 1)) \\
\wedge & \text{lr-proper-free-listp}\,(\textit{data-seg})) \\
\rightarrow & (\text{fetch}\,(\text{identity}\,(\text{LR-FP-ADDR}),\ \textit{data-seg}) \\
 & \quad \in \quad \text{lr-free-list-nodes}\,(\textit{max-addr},\ \textit{data-seg}))
\end{array}
$$

THEOREM: lr-count-free-nodes-lr-compile-quote-s-heap-reqs-help1

$$
\begin{array}{l}
(\text{adpp}\,(\text{untag}\,(\text{lr-max-node}\,(\textit{data-seg})),\ \textit{data-seg}) \\
\ \wedge\ \ \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\textit{data-seg})) \\
\ \wedge\ \ (\text{offset}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \textit{data-seg})) \\
\qquad <\ \ (\text{length}\,(\text{cdr}\,(\text{assoc}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}))) - 1)) \\
\ \wedge\ \ \text{lr-proper-free-listp}\,(\textit{data-seg}) \\
\ \wedge\ \ \text{lr-proper-p-areasp}\,(\textit{data-seg}) \\
\ \wedge\ \ (\textit{max-addr} = \text{lr-max-node}\,(\textit{data-seg}))) \\
\to\ \ (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{identity}\,(\text{LR-FP-ADDR}),\ \textit{data-seg}), \\
\qquad\qquad\qquad \text{lr-free-list-nodes}\,(\textit{max-addr},\ \textit{data-seg}), \\
\qquad\qquad\qquad \textit{data-seg}) \\
\qquad =\ \ (1 + \text{lr-count-free-nodes}\,(\text{fetch}\,(\text{add-addr}\,(\text{fetch}\,(\text{identity}\,(\text{LR-FP-ADDR}), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{data-seg}), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{identity}\,(\text{LR-REF-COUNT-OFFSET})), \\
\qquad\qquad\qquad\qquad\qquad\qquad \textit{data-seg}), \\
\qquad\qquad\qquad\qquad\qquad \text{delete}\,(\text{fetch}\,(\text{identity}\,(\text{LR-FP-ADDR}), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{data-seg}), \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{lr-free-list-nodes}\,(\textit{max-addr}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{data-seg})), \\
\qquad\qquad\qquad\qquad\qquad \textit{data-seg})))
\end{array}
$$

THEOREM: lr-proper-free-listp-lr-count-free-nodes-max-addr-alt

$$
\begin{array}{l}
(\text{lr-proper-free-listp}\,(\textit{data-seg}) \\
\ \wedge\ \ \text{definedp}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}) \\
\ \wedge\ \ \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\textit{data-seg})) \\
\ \wedge\ \ \text{lr-proper-p-areasp}\,(\textit{data-seg}) \\
\ \wedge\ \ (\text{offset}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \textit{data-seg})) \\
\qquad \not<\ \ (\text{length}\,(\text{cdr}\,(\text{assoc}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}))) - 1)) \\
\ \wedge\ \ (\textit{max-addr} = \text{lr-max-node}\,(\textit{data-seg}))) \\
\to\ \ (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{identity}\,(\text{LR-FP-ADDR}),\ \textit{data-seg}), \\
\qquad\qquad\qquad \text{lr-free-list-nodes}\,(\textit{max-addr},\ \textit{data-seg}), \\
\qquad\qquad\qquad \textit{data-seg}) \\
\qquad =\ \ \texttt{0})
\end{array}
$$

THEOREM: s-heap-reqs-object-t

$$
\begin{array}{l}
((\textit{flag} \neq \texttt{'list}) \wedge (\neg\ \text{definedp}\,(\mathbf{t},\ \textit{table}))) \\
\to\ \ (\text{s-heap-reqs}\,(\textit{flag},\ \mathbf{t},\ \textit{data-seg},\ \textit{table}) = \texttt{1})
\end{array}
$$

THEOREM: lessp-lr-boundary-offsetp-nodep-plus-node-size-fact-1

$$
\begin{array}{l}
(\text{lr-boundary-offsetp}\,(\textit{offset}) \wedge \text{lr-boundary-nodep}\,(\textit{addr})) \\
\to\ \ ((\textit{offset} < (\text{identity}\,(\text{LR-NODE-SIZE}) + \text{offset}\,(\textit{addr}))) \\
\qquad =\ \ (\text{offset}\,(\textit{addr}) \not< \textit{offset}))
\end{array}
$$

THEOREM: lr-count-free-nodes-lr-compile-quote-s-heap-reqs

$$
(\text{lr-proper-free-listp}\,(\textit{data-seg})
$$

$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-minimum-heapp (*data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs (*flag*, *object*, *data-seg*, *table*)))
$\rightarrow$ ((lr-count-free-nodes (fetch (LR-FP-ADDR,
car (lr-compile-quote (*flag*,
object,
data-seg,
table))),
lr-free-list-nodes (lr-max-node (*data-seg*),
car (lr-compile-quote (*flag*,
object,
data-seg,
table))),
car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))
$+$ s-heap-reqs (*flag*, *object*, *data-seg*, *table*))
$=$ lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg))

THEOREM: lr-compile-quote-lr-good-pointerp-tablep-help-1
let *ccar* **be** lr-compile-quote (**t**, *object*, *data-seg*, *table*)
in
(lr-proper-free-listp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\wedge$ lr-minimum-heapp (*data-seg*)
$\wedge$ (lr-count-free-nodes (fetch (identity (LR-FP-ADDR), *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ (s-heap-reqs (**t**, *object*, *data-seg*, *table*) $+$ x)))
$\rightarrow$ (lr-count-free-nodes (fetch (identity (LR-FP-ADDR), car (*ccar*)),
lr-free-list-nodes (lr-max-node (*data-seg*),
car (*ccar*)),
car (*ccar*))
$\not<$ x) **endlet**

THEOREM: s-heap-reqs-flag-list-nil-opener
s-heap-reqs ('list, **nil**, *data-seg*, *table*) = 0

THEOREM: lr-compile-quote-flag-list-nil-opener
lr-compile-quote ('list, **nil**, *data-seg*, *table*) = cons (*data-seg*, *table*)

THEOREM: s-heap-reqs-flag-list-cons-opener
s-heap-reqs ('list, cons (x, y), *data-seg*, *table*)
= (s-heap-reqs (**t**, x, *data-seg*, *table*)
    + s-heap-reqs ('list,
        y,
        car (lr-compile-quote (**t**, x, *data-seg*, *table*)),
        cdr (lr-compile-quote (**t**, x, *data-seg*, *table*))))

THEOREM: lr-compile-quote-flag-list-cons-opener
lr-compile-quote ('list, cons (x, y), *data-seg*, *table*)
= lr-compile-quote ('list,
        y,
        car (lr-compile-quote (**t**, x, *data-seg*, *table*)),
        cdr (lr-compile-quote (**t**, x, *data-seg*, *table*)))

THEOREM: lr-compile-quote-lr-good-pointerp-tablep-help-2
(lr-proper-free-listp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*)
∧ definedp (LR-HEAP-NAME, *data-seg*)
∧ lr-boundary-nodep (lr-max-node (*data-seg*))
∧ lr-minimum-heapp (*data-seg*)
∧ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
        lr-free-list-nodes (lr-max-node (*data-seg*),
            data-seg),
        data-seg)
    ≠ 0)
∧ (lr-count-free-nodes (fetch (add-addr (fetch (LR-FP-ADDR, *data-seg*),
                LR-REF-COUNT-OFFSET),
            data-seg),
        delete (fetch (LR-FP-ADDR, *data-seg*),
            lr-free-list-nodes (lr-max-node (*data-seg*),
                data-seg)),
        data-seg)
    ≮ s-heap-reqs ('list,
        list (car (*object*), cdr (*object*)),
        data-seg,
        table))
∧ listp (*object*))
→ ((length (cdr (assoc (identity (LR-HEAP-NAME), *data-seg*))) − 1)

$\not<$ (offset (fetch (identity (LR-FP-ADDR),
car (lr-compile-quote (`'list`,
list (car (*object*),
cdr (*object*)),
data-seg,
table))))
$+$ identity (LR-NODE-SIZE)))

EVENT: Disable s-heap-reqs-flag-list-cons-opener.

EVENT: Disable lr-compile-quote-flag-list-cons-opener.

THEOREM: lr-compile-quote-lr-good-pointerp-tablep-help-3
(lr-proper-free-listp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\wedge$ lr-minimum-heapp (*data-seg*)
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ `1`))
$\rightarrow$ ((length (cdr (assoc (identity (LR-HEAP-NAME), *data-seg*))) $-$ 1)
$\not<$ (offset (fetch (identity (LR-FP-ADDR), *data-seg*))
$+$ identity (LR-NODE-SIZE)))

THEOREM: lr-compile-quote-lr-good-pointerp-tablep
(lr-proper-free-listp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\wedge$ lr-minimum-heapp (*data-seg*)
$\wedge$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\wedge$ s-restricted-objectp (*flag*, *object*)
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs (*flag*, *object*, *data-seg*, *table*))
$\wedge$ definedp (**f**, *table*))
$\rightarrow$ lr-good-pointerp-tablep (cdr (lr-compile-quote (*flag*,
object,

data-seg,
table)),
car (lr-compile-quote (*flag*,
object,
data-seg,
table)))

THEOREM: lr-nodep-car-lr-compile-quote
lr-nodep (*addr*, *data-seg*)
→ lr-nodep (*addr*, car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))

THEOREM: lr-proper-free-listp-opener-2-lr-nodep
(lr-proper-free-listp (*data-seg*)
∧ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
∧ lr-boundary-nodep (lr-max-node (*data-seg*))
∧ (*addr* = fetch (identity (LR-FP-ADDR), *data-seg*)))
→ lr-nodep (*addr*, *data-seg*)

THEOREM: lr-nodep-deposit-a-list
lr-nodep (*addr1*, *data-seg*)
→ lr-nodep (*addr1*, deposit-a-list (*list*, *addr2*, *data-seg*))

DEFINITION:
induct-hint-16 (*object*, *list*, *data-seg*, *table*)
= **if** *list* ≃ **nil** **then** **t**
elseif *object* = car (*list*) **then** **t**
else induct-hint-16 (*object*,
cdr (*list*),
car (lr-compile-quote (**t**,
car (*list*),
data-seg,
table)),
cdr (lr-compile-quote (**t**,
car (*list*),
data-seg,
table))) **endif**

THEOREM: definedp-object-cdr-lr-compile-quote-list
(*object* ∈ *list*)
→ definedp (*object*, cdr (lr-compile-quote (`'list`, *list*, *data-seg*, *table*)))

THEOREM: lr-good-pointerp-cdr-assoc-car-lr-compile-quote-list
(lr-proper-free-listp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*)
∧ definedp (LR-HEAP-NAME, *data-seg*)

$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\wedge$ lr-minimum-heapp (*data-seg*)
$\wedge$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\wedge$ s-restricted-objectp (`'list`, *object-list*)
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\neq$ `0`)
$\wedge$ ((lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg) $-$ 1)
$\not<$ s-heap-reqs (`'list`, *object-list*, *data-seg*, *table*))
$\wedge$ definedp (**f**, *table*)
$\wedge$ (*object* $\in$ *object-list*))
$\rightarrow$ lr-good-pointerp (cdr (assoc (*object*,
cdr (lr-compile-quote (`'list`,
object-list,
data-seg,
table)))),
car (lr-compile-quote (`'list`,
object-list,
data-seg,
table)))

THEOREM: lr-good-pointerp-deposit-non-ref-not-good-pointerp
(lr-boundary-nodep (*addr*)
$\wedge$ adpp (untag (*addr*), *data-seg*)
$\wedge$ (area-name (*addr*) = `'heap`)
$\wedge$ (type (*addr*) = `'addr`)
$\wedge$ (cddr (*addr*) = **nil**)
$\wedge$ listp (*addr*)
$\wedge$ (n < LR-NODE-SIZE)
$\wedge$ ($n \neq$ LR-REF-COUNT-OFFSET)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-good-pointerp (*good-pointer*, *data-seg*)
$\wedge$ (type (fetch (add-addr (*addr*, LR-REF-COUNT-OFFSET), *data-seg*)) $\neq$ `'nat`))
$\rightarrow$ lr-good-pointerp (*good-pointer*, deposit (x, add-addr (*addr*, n), *data-seg*))

THEOREM: lr-good-pointerp-deposit-ref-count-not-good-pointerp
(lr-boundary-nodep (*addr*)
$\wedge$ adpp (untag (*addr*), *data-seg*)
$\wedge$ (area-name (*addr*) = `'heap`)

$\wedge$ (type ($addr$) = 'addr)
$\wedge$ (cddr ($addr$) = **nil**)
$\wedge$ listp ($addr$)
$\wedge$ (type (x) = 'nat)
$\wedge$ definedp (LR-HEAP-NAME, $data\text{-}seg$)
$\wedge$ lr-good-pointerp ($good\text{-}pointer$, $data\text{-}seg$)
$\wedge$ (type (fetch (add-addr ($addr$, LR-REF-COUNT-OFFSET), $data\text{-}seg$)) $\neq$ 'nat))
$\rightarrow$ lr-good-pointerp ($good\text{-}pointer$,
deposit (x,
add-addr ($addr$, identity (LR-REF-COUNT-OFFSET)),
$data\text{-}seg$))

THEOREM: lr-good-pointerp-deposit-non-add-addr-not-good-pointerp
(lr-boundary-nodep ($addr$)
$\wedge$ adpp (untag ($addr$), $data\text{-}seg$)
$\wedge$ (area-name ($addr$) = 'heap)
$\wedge$ (type ($addr$) = 'addr)
$\wedge$ (cddr ($addr$) = **nil**)
$\wedge$ listp ($addr$)
$\wedge$ definedp (LR-HEAP-NAME, $data\text{-}seg$)
$\wedge$ lr-good-pointerp ($good\text{-}pointer$, $data\text{-}seg$)
$\wedge$ (type (fetch (add-addr ($addr$, LR-REF-COUNT-OFFSET), $data\text{-}seg$)) $\neq$ 'nat))
$\rightarrow$ lr-good-pointerp ($good\text{-}pointer$, deposit (x, $addr$, $data\text{-}seg$))

THEOREM: lr-check-numberp-addrp-deposit-a-list-cons-same-addr
((type ($addr$) = 'addr)
$\wedge$ (cddr ($addr$) = **nil**)
$\wedge$ listp ($addr$)
$\wedge$ adpp (untag ($addr$), $data\text{-}seg$)
$\wedge$ lr-boundary-nodep ($addr$)
$\wedge$ (area-name ($addr$) = 'heap)
$\wedge$ (offset ($addr$) $\not<$ (LR-NODE-SIZE + offset (LR-F-ADDR)))
$\wedge$ (type (fetch (add-addr ($addr$, LR-REF-COUNT-OFFSET), $data\text{-}seg$))
= 'addr)
$\wedge$ lr-good-pointerp ($good\text{-}pointer$, $data\text{-}seg$)
$\wedge$ definedp ('heap, $data\text{-}seg$)
$\wedge$ lr-proper-p-areasp ($data\text{-}seg$)
$\wedge$ (type ($ref\text{-}count$) = 'nat)
$\wedge$ (untag ($ref\text{-}count$) $\in \mathbf{N}$)
$\wedge$ (type ($tagged\text{-}number$) = 'nat)
$\wedge$ (untag ($tagged\text{-}number$) $\in \mathbf{N}$))
$\rightarrow$ lr-check-numberp-addrp ($addr$,
deposit-a-list (list (identity (tag ('nat,
LR-ADD1-TAG)),

ref-count,
tagged-number,
good-pointer),
addr,
data-seg))

THEOREM: lr-proper-heapp-nodep-deposit-a-list-numberp
(lr-nodep (*max-addr*, *data-seg*)
$\land$ lr-nodep (*addr*, *data-seg*)
$\land$ lr-proper-heapp-nodep (*max-addr*, *data-seg*)
$\land$ (type (fetch (add-addr (*addr*, LR-REF-COUNT-OFFSET), *data-seg*)) $\neq$ `'nat`)
$\land$ lr-good-pointerp (*good-pointer*, *data-seg*)
$\land$ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ (type (*ref-count*) = `'nat`)
$\land$ (untag (*ref-count*) $\in \mathbf{N}$)
$\land$ (type (*tagged-number*) = `'nat`)
$\land$ (untag (*tagged-number*) $\in \mathbf{N}$))
$\rightarrow$ lr-proper-heapp-nodep (*max-addr*,
deposit-a-list (list (identity (tag (`'nat`,
LR-ADD1-TAG)),
ref-count,
tagged-number,
good-pointer),
addr,
data-seg))

THEOREM: lr-proper-heapp2-deposit-a-list-numberp
(lr-nodep (*max-addr*, *data-seg*)
$\land$ lr-nodep (*addr*, *data-seg*)
$\land$ lr-proper-heapp2 (*max-addr*, *data-seg*)
$\land$ (type (fetch (add-addr (*addr*, LR-REF-COUNT-OFFSET), *data-seg*)) $\neq$ `'nat`)
$\land$ lr-good-pointerp (*good-pointer*, *data-seg*)
$\land$ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ (type (*tag*) = `'nat`)
$\land$ (untag (*tag*) $\in \mathbf{N}$)
$\land$ (type (*tagged-number*) = `'nat`)
$\land$ (untag (*tagged-number*) $\in \mathbf{N}$))
$\rightarrow$ lr-proper-heapp2 (*max-addr*,
deposit-a-list (list (identity (tag (`'nat`, LR-ADD1-TAG)),
tag,
tagged-number,
good-pointer),

$addr$,
$data\text{-}seg$))

THEOREM: lr-good-pointerp-lr-undef-addr
(lr-minimum-heapp ($data\text{-}seg$)
 ∧ lr-proper-p-areasp ($data\text{-}seg$)
 ∧ lr-proper-heapp2 (lr-max-node ($data\text{-}seg$), $data\text{-}seg$)
 ∧ lr-nodep (lr-max-node ($data\text{-}seg$), $data\text{-}seg$))
 → lr-good-pointerp (identity (LR-UNDEF-ADDR), $data\text{-}seg$)

THEOREM: lr-proper-heapp-nodep-deposit-a-list-truep
(lr-nodep ($max\text{-}addr$, $data\text{-}seg$)
 ∧ lr-nodep ($addr$, $data\text{-}seg$)
 ∧ lr-proper-heapp-nodep ($max\text{-}addr$, $data\text{-}seg$)
 ∧ (type (fetch (add-addr ($addr$, LR-REF-COUNT-OFFSET), $data\text{-}seg$)) ≠ **'nat**)
 ∧ lr-good-pointerp ($good\text{-}pointer1$, $data\text{-}seg$)
 ∧ lr-good-pointerp ($good\text{-}pointer2$, $data\text{-}seg$)
 ∧ adpp (untag (lr-max-node ($data\text{-}seg$)), $data\text{-}seg$)
 ∧ lr-proper-p-areasp ($data\text{-}seg$)
 ∧ (type ($ref\text{-}count$) = **'nat**)
 ∧ (untag ($ref\text{-}count$) ∈ **N**))
 → lr-proper-heapp-nodep ($max\text{-}addr$,
deposit-a-list (list (identity (tag (**'nat**,
LR-TRUE-TAG)),
$ref\text{-}count$,
$good\text{-}pointer1$,
$good\text{-}pointer2$),
$addr$,
$data\text{-}seg$))

THEOREM: lr-proper-heapp2-deposit-a-list-truep
(lr-nodep ($max\text{-}addr$, $data\text{-}seg$)
 ∧ lr-nodep ($addr$, $data\text{-}seg$)
 ∧ lr-proper-heapp2 ($max\text{-}addr$, $data\text{-}seg$)
 ∧ (type (fetch (add-addr ($addr$, LR-REF-COUNT-OFFSET), $data\text{-}seg$)) ≠ **'nat**)
 ∧ lr-good-pointerp ($good\text{-}pointer1$, $data\text{-}seg$)
 ∧ lr-good-pointerp ($good\text{-}pointer2$, $data\text{-}seg$)
 ∧ adpp (untag (lr-max-node ($data\text{-}seg$)), $data\text{-}seg$)
 ∧ lr-proper-p-areasp ($data\text{-}seg$)
 ∧ (type (tag) = **'nat**)
 ∧ (untag (tag) ∈ **N**))
 → lr-proper-heapp2 ($max\text{-}addr$,
deposit-a-list (list (identity (tag (**'nat**, LR-TRUE-TAG)),
tag,
$good\text{-}pointer1$,

good-pointer2),
addr,
data-seg))

THEOREM: lr-compile-quote-preserves-lr-proper-heapp2
(lr-proper-free-listp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*)
∧ lr-nodep (lr-max-node (*data-seg*), *data-seg*)
∧ definedp (LR-HEAP-NAME, *data-seg*)
∧ lr-boundary-nodep (lr-max-node (*data-seg*))
∧ lr-good-pointerp-tablep (*table*, *data-seg*)
∧ s-restricted-objectp (*flag*, *object*)
∧ lr-minimum-heapp (*data-seg*)
∧ lr-proper-heapp2 (lr-max-node (*data-seg*), *data-seg*)
∧ definedp (**f**, *table*)
∧ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs (*flag*, *object*, *data-seg*, *table*)))
→ lr-proper-heapp2 (lr-max-node (*data-seg*),
car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))

THEOREM: plistp-pair-formals-with-addresses
plistp (pair-formals-with-addresses (*formals*, *table*))

THEOREM: strip-cars-pair-formals-with-addresses
strip-cars (pair-formals-with-addresses (*formals*, *table*))
= strip-cars (*formals*)

THEOREM: strip-cars-lr-make-initial-temps
strip-cars (lr-make-initial-temps (*temp-vars*)) = plist (*temp-vars*)

THEOREM: lr-s-similar-const-table-implies-lr-good-pointerp-tablep
lr-s-similar-const-table (*table*, *data-seg*)
→ lr-good-pointerp-tablep (*table*, *data-seg*)

THEOREM: lr-s-similar-const-table-deposit-cons
(lr-proper-heapp (*data-seg*)
∧ lr-s-similar-const-table (*table*, *data-seg*)
∧ lr-proper-p-areasp (*data-seg*))
→ lr-s-similar-const-table (*table*,
deposit-a-list (list (*x0*, *x1*, *x2*, *x3*),
fetch (identity (LR-FP-ADDR),
data-seg),
data-seg))

THEOREM: lr-valp-deposit-a-list-cons-cons

$$
\begin{array}{ll}
& (\text{listp}\,(\textit{object}) \\
\land & \text{lr-proper-p-areasp}\,(\textit{data-seg}) \\
\land & \text{lr-proper-heapp}\,(\textit{data-seg}) \\
\land & \text{lr-s-similar-const-table}\,(\textit{table},\ \textit{data-seg}) \\
\land & (\text{type}\,(\textit{ref-count}) = \texttt{'nat}) \\
\land & (\text{untag}\,(\textit{ref-count}) \in \mathbf{N}) \\
\land & \text{definedp}\,(\text{car}\,(\textit{object}),\ \textit{table}) \\
\land & \text{definedp}\,(\text{cdr}\,(\textit{object}),\ \textit{table}) \\
\land & \text{definedp}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}) \\
\land & (\textit{addr} = \text{fetch}\,(\text{identity}\,(\text{LR-FP-ADDR}),\ \textit{data-seg}))) \\
\rightarrow & \text{lr-valp}\,(\textit{object}, \\
& \quad \textit{addr}, \\
& \quad \text{deposit-a-list}\,(\text{list}\,(\text{identity}\,(\text{tag}\,(\texttt{'nat},\ \text{LR-CONS-TAG})), \\
& \qquad \textit{ref-count}, \\
& \qquad \text{cdr}\,(\text{assoc}\,(\text{car}\,(\textit{object}),\ \textit{table})), \\
& \qquad \text{cdr}\,(\text{assoc}\,(\text{cdr}\,(\textit{object}),\ \textit{table}))), \\
& \qquad \textit{addr}, \\
& \qquad \textit{data-seg}))
\end{array}
$$

THEOREM: lr-valp-deposit-a-list-cons-numberp

$$
\begin{array}{ll}
& ((\textit{object} \in \mathbf{N}) \\
\land & \text{lr-proper-p-areasp}\,(\textit{data-seg}) \\
\land & \text{lr-proper-heapp}\,(\textit{data-seg}) \\
\land & (\text{type}\,(\textit{ref-count}) = \texttt{'nat}) \\
\land & (\text{untag}\,(\textit{ref-count}) \in \mathbf{N}) \\
\land & \text{definedp}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}) \\
\land & (\textit{addr} = \text{fetch}\,(\text{identity}\,(\text{LR-FP-ADDR}),\ \textit{data-seg}))) \\
\rightarrow & \text{lr-valp}\,(\textit{object}, \\
& \quad \textit{addr}, \\
& \quad \text{deposit-a-list}\,(\text{list}\,(\text{identity}\,(\text{tag}\,(\texttt{'nat},\ \text{LR-ADD1-TAG})), \\
& \qquad \textit{ref-count}, \\
& \qquad \text{tag}\,(\texttt{'nat},\ \textit{object}), \\
& \qquad \text{identity}\,(\text{LR-UNDEF-ADDR})), \\
& \qquad \textit{addr}, \\
& \qquad \textit{data-seg}))
\end{array}
$$

THEOREM: lr-compile-quote-preserves-lr-valp

$$
\begin{array}{ll}
& (\text{lr-proper-p-areasp}\,(\textit{data-seg}) \\
\land & \text{lr-good-pointerp-tablep}\,(\textit{table},\ \textit{data-seg}) \\
\land & \text{s-restricted-objectp}\,(\textit{flag},\ \textit{object}) \\
\land & \text{lr-minimum-heapp}\,(\textit{data-seg}) \\
\land & \text{lr-nodep}\,(\text{lr-max-node}\,(\textit{data-seg}),\ \textit{data-seg}) \\
\land & \text{lr-proper-free-listp}\,(\textit{data-seg})
\end{array}
$$

$\land$ lr-proper-heapp2 (lr-max-node (*data-seg*), *data-seg*)
$\land$ definedp (**f**, *table*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs (*flag*, *object*, *data-seg*, *table*))
$\land$ lr-valp (*value*, *addr*, *data-seg*))
$\rightarrow$ lr-valp (*value*, *addr*, car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))

THEOREM: lr-compile-quote-preserves-lr-proper-heapp
(lr-proper-p-areasp (*data-seg*)
$\land$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\land$ lr-proper-heapp (*data-seg*)
$\land$ s-restricted-objectp (*flag*, *object*)
$\land$ definedp (**f**, *table*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs (*flag*, *object*, *data-seg*, *table*)))
$\rightarrow$ lr-proper-heapp (car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))

THEOREM: lr-valp-deposit-a-list-cons-truep
(lr-proper-p-areasp (*data-seg*)
$\land$ lr-proper-heapp (*data-seg*)
$\land$ (type (*ref-count*) = `'nat`)
$\land$ (untag (*ref-count*) $\in \mathbf{N}$)
$\land$ definedp (LR-HEAP-NAME, *data-seg*)
$\land$ (*addr* = fetch (identity (LR-FP-ADDR), *data-seg*)))
$\rightarrow$ lr-valp (**t**,
addr,
deposit-a-list (list (identity (tag (`'nat`, LR-TRUE-TAG)),
ref-count,
identity (LR-UNDEF-ADDR),
identity (LR-UNDEF-ADDR)),
addr,
data-seg))

THEOREM: lr-s-similar-const-table-lr-compile-quote
(lr-proper-p-areasp (*data-seg*)
$\land$ lr-proper-heapp (*data-seg*)
$\land$ lr-s-similar-const-table (*table*, *data-seg*)
$\land$ s-restricted-objectp (*flag*, *object*)
$\land$ definedp (**f**, *table*)

$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs (*flag*, *object*, *data-seg*, *table*)))
$\rightarrow$ lr-s-similar-const-table (cdr (lr-compile-quote (*flag*,
object,
data-seg,
table)),
car (lr-compile-quote (*flag*,
object,
data-seg,
table)))

THEOREM: p-objectp-cdr-assoc-car-lr-compile-quote
(lr-proper-p-areasp (*data-seg*)
$\land$ lr-proper-heapp (*data-seg*)
$\land$ lr-s-similar-const-table (*table*, *data-seg*)
$\land$ s-restricted-objectp (*flag*, *object*)
$\land$ (*word-size* $\not<$ s-ws-reqs (*flag*, *object*, *data-seg*, *table*))
$\land$ definedp (**f**, *table*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs (*flag*, *object*, *data-seg*, *table*))
$\land$ (*flag* $\neq$ `'list`))
$\rightarrow$ p-objectp (cdr (assoc (*object*,
cdr (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))),
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)),
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))

THEOREM: lr-count-free-nodes-s-init-heap-reqs
let *ccar* **be** lr-compile-quote (**t**, *object*, *data-seg*, *table*)
in
(lr-proper-p-areasp (*data-seg*)
$\land$ lr-proper-heapp (*data-seg*)

$\land$ lr-s-similar-const-table (*table*, *data-seg*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ (s-heap-reqs (**t**, *object*, *data-seg*, *table*)
$+$ s-init-heap-reqs (*params*, car (*ccar*), cdr (*ccar*)))))
$\rightarrow$ (lr-count-free-nodes (fetch (identity (LR-FP-ADDR), car (*ccar*)),
lr-free-list-nodes (lr-max-node (*data-seg*),
car (*ccar*)),
car (*ccar*))
$\not<$ s-init-heap-reqs (*params*, car (*ccar*), cdr (*ccar*))) **endlet**

THEOREM: all-p-objects-lookup-strip-cdrs-lr-init-data-seg-table
(lr-proper-p-areasp (*data-seg*)
$\land$ lr-proper-heapp (*data-seg*)
$\land$ lr-s-similar-const-table (*table*, *data-seg*)
$\land$ definedp (**f**, *table*)
$\land$ definedp (**t**, *table*)
$\land$ s-init-data-seg-restrictedp (*params*)
$\land$ (*word-size* $\not<$ s-init-ws-reqs (*params*, *data-seg*, *table*))
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-init-heap-reqs (*params*, *data-seg*, *table*)))
$\rightarrow$ all-p-objects-lookup (strip-cdrs (*params*),
cdr (lr-init-data-seg-table (*params*, *data-seg*, *table*)),
p-state (*pc*,
ctrl-stk,
temp-stk,
prog-seg,
car (lr-init-data-seg-table (*params*,
data-seg,
table)),
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))

THEOREM: lr-minimum-heapp-lr-init-data-seg
(*heap-size* $\not<$ 4) $\rightarrow$ lr-minimum-heapp (lr-init-data-seg (*heap-size*))

THEOREM: adpp-cons-heap-name-node-size-lr-init-data-seg
(*heap-size* $\not<$ 2)

$\rightarrow$ adpp (cons (identity (LR-HEAP-NAME), identity (LR-NODE-SIZE) $*$ *heap-size*), lr-init-data-seg (*heap-size*))

THEOREM: lr-check-f-addrp-lr-undef-addr-lr-init-data-seg
((offset (*addr*) = identity (offset (LR-UNDEF-ADDR)))
$\wedge$ (type (*addr*) = `'addr`)
$\wedge$ (cddr (*addr*) = **nil**)
$\wedge$ listp (*addr*)
$\wedge$ (area-name (*addr*) = LR-HEAP-NAME)
$\wedge$ adpp (untag (*addr*), *data-seg*))
$\rightarrow$ lr-check-undef-addrp (*addr*, *data-seg*)

THEOREM: fetch-offset-lr-t-addr-ref-count-offset-compile-quote-t
((type (*addr*) = `'addr`)
$\wedge$ (cddr (*addr*) = **nil**)
$\wedge$ listp (*addr*)
$\wedge$ (area-name (*addr*) = LR-HEAP-NAME)
$\wedge$ adpp (untag (*addr*), *data-seg*)
$\wedge$ lr-boundary-nodep (*addr*)
$\wedge$ ($\neg$ definedp (**t**, *table*))
$\wedge$ lr-proper-free-listp (*data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\wedge$ (offset (fetch (LR-FP-ADDR, *data-seg*)) < LR-MINIMUM-HEAP-SIZE)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ ((length (cdr (assoc (LR-HEAP-NAME, *data-seg*))) $-$ 1)
$\not<$ (offset (*addr*) + LR-NODE-SIZE)))
$\rightarrow$ (fetch (add-addr (*addr*, identity (LR-REF-COUNT-OFFSET)), car (lr-compile-quote (**t**, **t**, *data-seg*, *table*)))
= **if** offset (*addr*) = offset (fetch (LR-FP-ADDR, *data-seg*))
then identity (tag (`'nat`, `1`))
else fetch (add-addr (*addr*, identity (LR-REF-COUNT-OFFSET)), *data-seg*) **endif**)

THEOREM: definedp-cdr-lr-compile-quote-t
definedp (x, cdr (lr-compile-quote (**t**, **t**, *data-seg*, *table*)))
= ((x = **t**) $\vee$ definedp (x, *table*))

THEOREM: fetch-lr-fp-addr-compile-quote-t
(($\neg$ definedp (**t**, *table*))
$\wedge$ lr-proper-free-listp (*data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\wedge$ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
$\wedge$ ((length (value (LR-HEAP-NAME, *data-seg*)) $-$ 1)
$\not<$ (offset (fetch (LR-FP-ADDR, *data-seg*)) + LR-NODE-SIZE))

$\wedge$ lr-proper-p-areasp (*data-seg*))
$\rightarrow$ (fetch (identity (LR-FP-ADDR), car (lr-compile-quote (**t**, **t**, *data-seg*, *table*)))
= fetch (add-addr (fetch (identity (LR-FP-ADDR), *data-seg*),
identity (LR-REF-COUNT-OFFSET)),
data-seg))

THEOREM: numberp-lessp-4-not-3-not-2-not-1-must-be-0
$((c \in \mathbf{N}) \wedge (c \neq 3) \wedge (c \neq 2) \wedge (c \neq 1) \wedge (c < 4))$
$\rightarrow (c = 0)$

EVENT: Disable fetch-offset-lr-t-addr-ref-count-offset-compile-quote-t.

THEOREM: lessp-minimum-heap-size-not-0-f-t-must-be-undef-alt-1-help
(lr-boundary-offsetp (*offset*)
$\wedge$ ($offset \in \mathbf{N}$)
$\wedge$ ($offset \neq$ offset (LR-UNDEF-ADDR))
$\wedge$ ($offset \neq$ offset (LR-F-ADDR))
$\wedge$ ($offset \neq$ offset (LR-T-ADDR))
$\wedge$ ($offset \neq$ offset (LR-0-ADDR)))
$\rightarrow$ ($offset \not<$ LR-MINIMUM-HEAP-SIZE)

THEOREM: lessp-minimum-heap-size-not-0-f-t-must-be-undef-alt-1
((offset (*addr*) < LR-MINIMUM-HEAP-SIZE)
$\wedge$ (type (*addr*) = `'addr`)
$\wedge$ (cddr (*addr*) = **nil**)
$\wedge$ listp (*addr*)
$\wedge$ (area-name (*addr*) = `'heap`)
$\wedge$ adpp (untag (*addr*), lr-init-data-seg (*heap-size*))
$\wedge$ lr-boundary-nodep (*addr*)
$\wedge$ (offset (*addr*) $\neq$ offset (LR-0-ADDR))
$\wedge$ (offset (*addr*) $\neq$ offset (LR-T-ADDR))
$\wedge$ (offset (*addr*) $\neq$ offset (LR-F-ADDR)))
$\rightarrow$ (fetch (*addr*, lr-init-data-seg (*heap-size*))
= identity (tag (`'nat`, LR-UNDEFINED-TAG)))

THEOREM: lessp-lr-boundary-offsetp-3
(lr-boundary-offsetp (*offset*) $\wedge$ ($offset \in \mathbf{N}$))
$\rightarrow$ $((offset < 3) = (offset = 0))$

THEOREM: numberp-lessp-2-not-1-must-be-0
$((c \in \mathbf{N}) \wedge (c \neq 1) \wedge (c < 2)) \rightarrow (c = 0)$

THEOREM: not-lessp-difference-lr-boundary-offsetp-fact
(($offset \in \mathbf{N}$) $\wedge$ ($offset <$ (LR-NODE-SIZE + offset (LR-F-ADDR))))
$\rightarrow$ (lr-boundary-offsetp (*offset*)

$= \quad ((\mathit{offset} = \text{identity}(\text{offset}(\textsc{lr-f-addr})))$
$\quad \lor \quad (\mathit{offset} = \text{identity}(\text{offset}(\textsc{lr-undef-addr})))))$

THEOREM: fetch-ref-count-lr-init-data-seg-free-list

$(((\textsc{lr-node-size} * \mathit{heap\text{-}size}) \not< (\text{offset}(\mathit{addr}) + \textsc{lr-node-size}))$
$\land \quad (\text{type}(\mathit{addr}) = \texttt{'addr})$
$\land \quad (\text{cddr}(\mathit{addr}) = \mathbf{nil})$
$\land \quad \text{listp}(\mathit{addr})$
$\land \quad (\text{area-name}(\mathit{addr}) = \textsc{lr-heap-name})$
$\land \quad \text{listp}(\text{untag}(\mathit{addr}))$
$\land \quad \text{lr-boundary-nodep}(\mathit{addr})$
$\land \quad (\mathit{heap\text{-}size} \not< 2))$
$\to \quad (\text{fetch}(\text{add-addr}(\mathit{addr}, \text{identity}(\textsc{lr-ref-count-offset})),$
$\qquad \text{lr-init-data-seg}(\mathit{heap\text{-}size}))$
$\quad = \quad \mathbf{if}\ \text{offset}(\textsc{lr-f-addr}) < \text{offset}(\mathit{addr})$
$\qquad \mathbf{then}\ \text{add-addr}(\mathit{addr}, \text{identity}(\textsc{lr-node-size}))$
$\qquad \mathbf{else}\ \text{identity}(\text{tag}(\texttt{'nat}, \texttt{1}))\ \mathbf{endif})$

THEOREM: lessp-offset-lr-init-data-seg-adpp-untag-lessp-offset

$(\text{adpp}(\text{untag}(\mathit{addr}), \text{lr-init-data-seg}(\mathit{heap\text{-}size}))$
$\land \quad \text{lr-boundary-offsetp}(\mathit{offset})$
$\land \quad \text{lr-boundary-offsetp}(\text{offset}(\mathit{addr}))$
$\land \quad (\mathit{offset} < \text{offset}(\mathit{addr}))$
$\land \quad (\text{area-name}(\mathit{addr}) = \textsc{lr-heap-name})$
$\land \quad (\mathit{heap\text{-}size} \not< 2))$
$\to \quad ((\mathit{offset} < (\text{identity}(\textsc{lr-node-size}) * \mathit{heap\text{-}size})) = \mathbf{t})$

THEOREM: lr-count-free-nodes-lr-all-nodes

$(\text{lr-boundary-nodep}(\mathit{addr})$
$\land \quad (\text{area-name}(\mathit{addr}) = \textsc{lr-heap-name})$
$\land \quad \text{listp}(\text{untag}(\mathit{addr}))$
$\land \quad (\text{offset}(\mathit{addr}) \in \mathbf{N})$
$\land \quad (\text{type}(\mathit{addr}) = \texttt{'addr})$
$\land \quad (\text{cddr}(\mathit{addr}) = \mathbf{nil})$
$\land \quad \text{listp}(\mathit{addr})$
$\land \quad \text{lr-boundary-nodep}(\mathit{max\text{-}addr})$
$\land \quad (\text{area-name}(\mathit{max\text{-}addr}) = \textsc{lr-heap-name})$
$\land \quad \text{adpp}(\text{untag}(\mathit{max\text{-}addr}), \text{lr-init-data-seg}(\mathit{heap\text{-}size}))$
$\land \quad (\text{type}(\mathit{max\text{-}addr}) = \texttt{'addr})$
$\land \quad (\text{cddr}(\mathit{max\text{-}addr}) = \mathbf{nil})$
$\land \quad \text{listp}(\mathit{max\text{-}addr})$
$\land \quad (\text{offset}(\textsc{lr-f-addr}) < \text{offset}(\mathit{addr}))$
$\land \quad (\mathit{heap\text{-}size} \not< 2))$
$\to \quad (\text{lr-count-free-nodes}(\mathit{addr},$
$\qquad\qquad \text{lr-all-nodes}(\text{offset}(\mathit{addr}), \ \mathit{max\text{-}addr}),$

$$\begin{array}{ll} & \text{lr-init-data-seg}\,(\textit{heap-size})) \\ = & \text{length}\,(\text{lr-all-nodes}\,(\text{offset}\,(\textit{addr}),\ \textit{max-addr}))) \end{array}$$

THEOREM: same-signature-car-lr-init-data-seg-table-help-1

$$\begin{array}{l} \textbf{let}\ \textit{comp-obj}\ \ \textbf{be}\ \text{lr-compile-quote}\,(\mathbf{t},\ \textit{object},\ \textit{data-seg},\ \textit{table}) \\ \textbf{in} \\ (\text{same-signature}\,(\text{car}\,(\textit{comp-obj}), \\ \qquad\qquad\qquad \text{car}\,(\text{lr-init-data-seg-table}\,(\textit{params}, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{car}\,(\textit{comp-obj}), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{cdr}\,(\textit{comp-obj})))) \\ \ \wedge\ \ \text{lr-proper-free-listp}\,(\textit{data-seg}) \\ \ \wedge\ \ \text{lr-proper-p-areasp}\,(\textit{data-seg}) \\ \ \wedge\ \ \text{definedp}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}) \\ \ \wedge\ \ \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\textit{data-seg}))) \\ \rightarrow\ \ \text{same-signature}\,(\textit{data-seg}, \\ \qquad\qquad\qquad\quad \text{car}\,(\text{lr-init-data-seg-table}\,(\textit{params}, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{car}\,(\textit{comp-obj}), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{cdr}\,(\textit{comp-obj})))) \ \textbf{endlet} \end{array}$$

THEOREM: same-signature-car-lr-init-data-seg-table

$$\begin{array}{l} (\text{lr-proper-free-listp}\,(\textit{data-seg}) \\ \ \wedge\ \ \text{lr-proper-p-areasp}\,(\textit{data-seg}) \\ \ \wedge\ \ \text{definedp}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}) \\ \ \wedge\ \ \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\textit{data-seg}))) \\ \rightarrow\ \ \text{same-signature}\,(\textit{data-seg}, \\ \qquad\qquad\qquad\quad \text{car}\,(\text{lr-init-data-seg-table}\,(\textit{params},\ \textit{data-seg},\ \textit{table}))) \end{array}$$

THEOREM: lr-max-node-car-lr-init-data-seg-table

$$\begin{array}{l} (\text{lr-proper-free-listp}\,(\textit{data-seg}) \\ \ \wedge\ \ \text{lr-proper-p-areasp}\,(\textit{data-seg}) \\ \ \wedge\ \ \text{definedp}\,(\text{LR-HEAP-NAME},\ \textit{data-seg}) \\ \ \wedge\ \ \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\textit{data-seg}))) \\ \rightarrow\ \ (\text{lr-max-node}\,(\text{car}\,(\text{lr-init-data-seg-table}\,(\textit{params},\ \textit{data-seg},\ \textit{table}))) \\ \qquad = \ \ \text{lr-max-node}\,(\textit{data-seg})) \end{array}$$

THEOREM: s-heap-reqs-object-0

$$\begin{array}{l} ((\textit{flag} \neq \texttt{'list}) \wedge (\neg\ \text{definedp}\,(\texttt{0},\ \textit{table}))) \\ \rightarrow\ \ (\text{s-heap-reqs}\,(\textit{flag},\ \texttt{0},\ \textit{data-seg},\ \textit{table}) = \texttt{1}) \end{array}$$

THEOREM: lr-free-list-nodes-deposit-0

$$\begin{array}{l} ((\text{type}\,(\textit{max-addr}) = \texttt{'addr}) \\ \ \wedge\ \ (\text{cddr}\,(\textit{max-addr}) = \textbf{nil}) \\ \ \wedge\ \ \text{listp}\,(\textit{max-addr}) \\ \ \wedge\ \ \text{adpp}\,(\text{untag}\,(\textit{max-addr}),\ \textit{data-seg}) \\ \ \wedge\ \ \text{lr-boundary-nodep}\,(\textit{max-addr}) \end{array}$$

```
 ∧   (area-name (max-addr) = LR-HEAP-NAME)
 ∧   (¬ definedp (0, table))
 ∧   lr-proper-free-listp (data-seg)
 ∧   adpp (untag (lr-max-node (data-seg)), data-seg)
 ∧   lr-boundary-nodep (lr-max-node (data-seg))
 ∧   lr-proper-p-areasp (data-seg)
 ∧   ((length (cdr (assoc (LR-HEAP-NAME, data-seg))) − 1)
        ≮   (offset (fetch (LR-FP-ADDR, data-seg)) + LR-NODE-SIZE)))
→    (lr-free-list-nodes (max-addr,
                          car (lr-compile-quote (t, 0, data-seg, table)))
       =   lr-free-list-nodes (max-addr,
                                deposit (identity (tag ('nat, 1)),
                                         add-addr (fetch (identity (LR-FP-ADDR),
                                                          data-seg),
                                                   identity (LR-REF-COUNT-OFFSET)),
                                         data-seg)))
```

THEOREM: lr-free-list-nodes-deposit-t

```
((type (max-addr) = 'addr)
 ∧   (cddr (max-addr) = nil)
 ∧   listp (max-addr)
 ∧   adpp (untag (max-addr), data-seg)
 ∧   lr-boundary-nodep (max-addr)
 ∧   (area-name (max-addr) = LR-HEAP-NAME)
 ∧   (¬ definedp (t, table))
 ∧   lr-proper-free-listp (data-seg)
 ∧   adpp (untag (lr-max-node (data-seg)), data-seg)
 ∧   lr-boundary-nodep (lr-max-node (data-seg))
 ∧   lr-proper-p-areasp (data-seg)
 ∧   ((length (cdr (assoc (LR-HEAP-NAME, data-seg))) − 1)
        ≮   (offset (fetch (LR-FP-ADDR, data-seg)) + LR-NODE-SIZE)))
→    (lr-free-list-nodes (max-addr,
                          car (lr-compile-quote (t, t, data-seg, table)))
       =   lr-free-list-nodes (max-addr,
                                deposit (identity (tag ('nat, 1)),
                                         add-addr (fetch (identity (LR-FP-ADDR),
                                                          data-seg),
                                                   identity (LR-REF-COUNT-OFFSET)),
                                         data-seg)))
```

THEOREM: fetch-lr-fp-addr-compile-quote-0

```
((¬ definedp (0, table))
 ∧   ((length (cdr (assoc (LR-HEAP-NAME, data-seg))) − 1)
        ≮   (offset (fetch (LR-FP-ADDR, data-seg)) + LR-NODE-SIZE))
```

$\land$ definedp (area-name (LR-FP-ADDR), *data-seg*))
$\rightarrow$ (fetch (identity (LR-FP-ADDR),
car (lr-compile-quote (`t`, `0`, *data-seg*, *table*)))
= fetch (add-addr (fetch (identity (LR-FP-ADDR), *data-seg*),
identity (LR-REF-COUNT-OFFSET)),
data-seg))

THEOREM: fetch-add-addr-ref-count-f-addr-lr-init-data-seg
(*heap-size* $\not<$ `2`)
$\rightarrow$ (fetch (identity (add-addr (LR-F-ADDR, LR-REF-COUNT-OFFSET)),
lr-init-data-seg (*heap-size*))
= identity (tag (`'nat`, `1`)))

THEOREM: lr-good-pointerp-tablep-f-lr-f-addr-lr-init-data-seg
(*heap-size* $\not<$ `4`)
$\rightarrow$ lr-good-pointerp-tablep (list (cons (**f**, identity (LR-F-ADDR))),
lr-init-data-seg (*heap-size*))

THEOREM: lr-proper-heapp-nodep-lr-undef-addr-lr-init-data-seg
(*heap-size* $\not<$ `2`)
$\rightarrow$ lr-proper-heapp-nodep (identity (LR-UNDEF-ADDR),
lr-init-data-seg (*heap-size*))

DEFINITION:
induct-hint-2 (*offset*)
= **if** *offset* < offset (add-addr (LR-F-ADDR, LR-NODE-SIZE)) **then t**
else induct-hint-2 (*offset* − LR-NODE-SIZE) **endif**

THEOREM: lr-boundary-offsetp-difference-lr-node-size
lr-boundary-offsetp (*offset*)
$\rightarrow$ lr-boundary-offsetp (*offset* − identity (LR-NODE-SIZE))

THEOREM: fetch-lr-f-addr-lr-init-data-seg
(*heap-size* $\not<$ `2`)
$\rightarrow$ (fetch (identity (LR-F-ADDR), lr-init-data-seg (*heap-size*))
= identity (tag (`'nat`, LR-FALSE-TAG)))

THEOREM: lr-proper-heapp-nodep-lr-init-data-seg-helper
((*offset* $\not<$ offset (LR-F-ADDR))
$\land$ (*heap-size* $\not<$ `2`)
$\land$ (*offset* $\in \mathbf{N}$)
$\land$ lr-boundary-nodep (tag (`'addr`, cons (LR-HEAP-NAME, *offset*)))
$\land$ ((LR-NODE-SIZE $*$ *heap-size*) $\not<$ (*offset* + LR-NODE-SIZE)))
$\rightarrow$ lr-proper-heapp-nodep (tag (`'addr`, cons (identity (LR-HEAP-NAME), *offset*)),
lr-init-data-seg (*heap-size*))

THEOREM: lr-proper-heapp2-lr-init-data-seg-helper

$$
\begin{array}{ll}
& ((\mathit{heap\text{-}size} \not< \mathtt{2}) \\
\wedge & (\mathit{offset} < \text{length}\,(\text{cdr}\,(\text{assoc}\,(\text{LR-HEAP-NAME}, \\
& \qquad\qquad \text{lr-init-data-seg}\,(\mathit{heap\text{-}size}))))) \\
\wedge & (\mathit{offset} \in \mathbf{N}) \\
\wedge & \text{lr-boundary-nodep}\,(\text{tag}\,(\texttt{'addr}, \text{cons}\,(\text{LR-HEAP-NAME}, \mathit{offset})))) \\
\rightarrow & \text{lr-proper-heapp2}\,(\text{tag}\,(\texttt{'addr}, \text{cons}\,(\text{LR-HEAP-NAME}, \mathit{offset})), \\
& \qquad \text{lr-init-data-seg}\,(\mathit{heap\text{-}size}))
\end{array}
$$

THEOREM: lr-proper-heapp2-lr-init-data-seg

$$
\begin{array}{ll}
& (\mathit{heap\text{-}size} \not< \mathtt{2}) \\
\rightarrow & \text{lr-proper-heapp2}\,(\text{tag}\,(\texttt{'addr}, \\
& \qquad \text{cons}\,(\text{identity}\,(\text{LR-HEAP-NAME}), \\
& \qquad\quad \text{identity}\,(\text{LR-NODE-SIZE}) * \mathit{heap\text{-}size})), \\
& \qquad \text{lr-init-data-seg}\,(\mathit{heap\text{-}size}))
\end{array}
$$

THEOREM: fetch-add-addr-ref-count-lr-t-addr-lr-init-data-seg

$$
\begin{array}{ll}
& (\mathit{heap\text{-}size} \not< \mathtt{4}) \\
\rightarrow & (\text{fetch}\,(\text{identity}\,(\text{add-addr}\,(\text{LR-T-ADDR}, \text{LR-REF-COUNT-OFFSET})), \\
& \qquad \text{lr-init-data-seg}\,(\mathit{heap\text{-}size})) \\
& \quad = \text{identity}\,(\text{add-addr}\,(\text{LR-T-ADDR}, \text{LR-NODE-SIZE})))
\end{array}
$$

THEOREM: fetch-t-addr-compile-quote-list-init-data-seg

$$
\begin{array}{ll}
& ((\mathit{heap\text{-}size} \not< \mathtt{4}) \wedge (\neg\ \text{definedp}\,(\mathbf{t}, \mathit{table}))) \\
\rightarrow & (\text{fetch}\,(\text{identity}\,(\text{LR-T-ADDR}), \\
& \qquad \text{car}\,(\text{lr-compile-quote}\,(\texttt{'list}, \\
& \qquad\qquad \text{list}\,(\mathbf{t}, \mathtt{0}), \\
& \qquad\qquad \text{lr-init-data-seg}\,(\mathit{heap\text{-}size}), \\
& \qquad\qquad \mathit{table}))) \\
& \quad = \text{identity}\,(\text{tag}\,(\texttt{'nat}, \text{LR-TRUE-TAG})))
\end{array}
$$

THEOREM: fetch-ref-count-t-addr-compile-quote-list-init-data-seg

$$
\begin{array}{ll}
& ((\mathit{heap\text{-}size} \not< \mathtt{4}) \wedge (\neg\ \text{definedp}\,(\mathbf{t}, \mathit{table}))) \\
\rightarrow & (\text{fetch}\,(\text{identity}\,(\text{add-addr}\,(\text{LR-T-ADDR}, \text{LR-REF-COUNT-OFFSET})), \\
& \qquad \text{car}\,(\text{lr-compile-quote}\,(\texttt{'list}, \\
& \qquad\qquad \text{list}\,(\mathbf{t}, \mathtt{0}), \\
& \qquad\qquad \text{lr-init-data-seg}\,(\mathit{heap\text{-}size}), \\
& \qquad\qquad \mathit{table}))) \\
& \quad = \text{identity}\,(\text{tag}\,(\texttt{'nat}, \mathtt{1})))
\end{array}
$$

THEOREM: fetch-add-addr-ref-count-lr-0-addr-lr-init-data-seg

$$
\begin{array}{ll}
& (\mathit{heap\text{-}size} \not< \mathtt{4}) \\
\rightarrow & (\text{fetch}\,(\text{identity}\,(\text{add-addr}\,(\text{LR-0-ADDR}, \text{LR-REF-COUNT-OFFSET})), \\
& \qquad \text{lr-init-data-seg}\,(\mathit{heap\text{-}size})) \\
& \quad = \text{identity}\,(\text{add-addr}\,(\text{LR-0-ADDR}, \text{LR-NODE-SIZE})))
\end{array}
$$

THEOREM: fetch-0-addr-compile-quote-list-init-data-seg
((*heap-size* $\not<$ 4) $\land$ ($\neg$ definedp (**t**, *table*)) $\land$ ($\neg$ definedp (0, *table*)))
$\rightarrow$ (fetch (identity (LR-0-ADDR),
car (lr-compile-quote ('list,
list (**t**, 0),
lr-init-data-seg (*heap-size*),
table)))
= identity (tag ('nat, LR-ADD1-TAG)))

THEOREM: fetch-ref-count-0-addr-compile-quote-list-init-data-seg
((*heap-size* $\not<$ 4) $\land$ ($\neg$ definedp (**t**, *table*)) $\land$ ($\neg$ definedp (0, *table*)))
$\rightarrow$ (fetch (identity (add-addr (LR-0-ADDR, LR-REF-COUNT-OFFSET)),
car (lr-compile-quote ('list,
list (**t**, 0),
lr-init-data-seg (*heap-size*),
table)))
= identity (tag ('nat, 1)))

THEOREM: fetch-unbox-nat-0-addr-compile-quote-list-init-data-seg
((*heap-size* $\not<$ 4) $\land$ ($\neg$ definedp (**t**, *table*)) $\land$ ($\neg$ definedp (0, *table*)))
$\rightarrow$ (fetch (identity (add-addr (LR-0-ADDR, LR-UNBOX-NAT-OFFSET)),
car (lr-compile-quote ('list,
list (**t**, 0),
lr-init-data-seg (*heap-size*),
table)))
= identity (tag ('nat, 0)))

THEOREM: lr-proper-heapp-lr-compile-quote-ft-lr-init-data-seg
(*heap-size* $\not<$ 4)
$\rightarrow$ lr-proper-heapp (car (lr-compile-quote ('list,
list (**t**, 0),
lr-init-data-seg (*heap-size*),
list (cons (**f**, identity (LR-F-ADDR))))))

THEOREM: cdr-compile-quote-list-t0-lr-init-data-seg-cons-table
(($\neg$ definedp (0, *table*)) $\land$ ($\neg$ definedp (**t**, *table*)) $\land$ (*heap-size* $\not<$ 4))
$\rightarrow$ (cdr (lr-compile-quote ('list,
list (**t**, 0),
lr-init-data-seg (*heap-size*),
table))
= cons (cons (0, identity (LR-0-ADDR)),
cons (cons (**t**, identity (LR-T-ADDR)), *table*)))

THEOREM: fetch-f-addr-compile-quote-list-init-data-seg
((*heap-size* $\not<$ 4) $\land$ ($\neg$ definedp (**t**, *table*)) $\land$ ($\neg$ definedp (0, *table*)))

$\rightarrow$ (fetch (identity (LR-F-ADDR),
car (lr-compile-quote ('list,
list (t, 0),
lr-init-data-seg (*heap-size*),
table)))
$=$ identity (tag ('nat, LR-FALSE-TAG)))

THEOREM: fetch-ref-count-f-addr-compile-quote-list-init-data-seg
((*heap-size* $\not<$ 4) $\wedge$ ($\neg$ definedp (t, *table*)) $\wedge$ ($\neg$ definedp (0, *table*)))
$\rightarrow$ (fetch (identity (add-addr (LR-F-ADDR, LR-REF-COUNT-OFFSET)),
car (lr-compile-quote ('list,
list (t, 0),
lr-init-data-seg (*heap-size*),
table)))
$=$ identity (tag ('nat, 1)))

THEOREM: lr-s-similar-const-table-compile-quote-t0-init-data-seg
(*heap-size* $\not<$ 4)
$\rightarrow$ lr-s-similar-const-table (cdr (lr-compile-quote ('list,
list (t, 0),
lr-init-data-seg (*heap-size*),
list (cons (**f**,
identity (LR-F-ADDR))))),
car (lr-compile-quote ('list,
list (t, 0),
lr-init-data-seg (*heap-size*),
list (cons (**f**,
identity (LR-F-ADDR))))))

EVENT: Disable cdr-compile-quote-list-t0-lr-init-data-seg-cons-table.

THEOREM: fetch-fp-addr-compile-quote-list-t0-lr-init-data-seg-cons-table
(($\neg$ definedp (0, *table*)) $\wedge$ ($\neg$ definedp (t, *table*)) $\wedge$ (*heap-size* $\not<$ 4))
$\rightarrow$ (fetch (identity (LR-FP-ADDR),
car (lr-compile-quote ('list,
list (t, 0),
lr-init-data-seg (*heap-size*),
table)))
$=$ identity (add-addr (LR-0-ADDR, LR-NODE-SIZE)))

THEOREM: lr-free-list-nodes-lr-compile-quote-t0
(($\neg$ definedp (t, *table*)) $\wedge$ ($\neg$ definedp (0, *table*)) $\wedge$ (*heap-size* $\not<$ 4))
$\rightarrow$ (lr-free-list-nodes (tag ('addr,
cons (identity (LR-HEAP-NAME),

identity (LR-NODE-SIZE) $*$ *heap-size*)),
car (lr-compile-quote (`'list`,
list (`t`, `0`),
lr-init-data-seg (*heap-size*),
table)))
= lr-all-nodes (identity (LR-NODE-SIZE + offset (LR-0-ADDR)),
tag (`'addr`,
cons (identity (LR-HEAP-NAME),
identity (LR-NODE-SIZE) $*$ *heap-size*))))

THEOREM: listp-lr-all-nodes
listp (lr-all-nodes (*min-offset*, *max-addr*))
= ((offset (*max-addr*) $\not\simeq$ `0`) $\wedge$ (*min-offset* $<$ offset (*max-addr*)))

THEOREM: length-lr-all-nodes
(lr-boundary-offsetp (*offset*) $\wedge$ (*offset* $\in \mathbf{N}$) $\wedge$ lr-boundary-nodep (*addr*))
$\rightarrow$ (length (lr-all-nodes (*offset*, *addr*))
= ((offset (*addr*) $-$ *offset*) $\div$ identity (LR-NODE-SIZE)))

THEOREM: lr-count-free-nodes-append-lr-all-nodes-fact
(((LR-NODE-SIZE $*$ *heap-size*) $\not<$ LR-MINIMUM-HEAP-SIZE)
$\wedge$ ($\neg$ definedp (`t`, *table*))
$\wedge$ ($\neg$ definedp (`0`, *table*)))
$\rightarrow$ (lr-count-free-nodes (identity (add-addr (LR-0-ADDR, LR-NODE-SIZE)),
lr-all-nodes (identity (LR-MINIMUM-HEAP-SIZE),
tag (`'addr`,
cons (identity (LR-HEAP-NAME),
identity (LR-NODE-SIZE)
$*$ *heap-size*))),
car (lr-compile-quote (`'list`,
list (`t`, `0`),
lr-init-data-seg (*heap-size*),
table)))
= (*heap-size* $-$ LR-NODE-SIZE))

THEOREM: proper-p-alistp-pair-formal-with-addresses
(all-litatoms (strip-cars (*params*))
$\wedge$ s-init-data-seg-restrictedp (*params*)
$\wedge$ (*heap-size* $\not<$ s-total-heap-reqs (s-progs (s), *params*, *heap-size*))
$\wedge$ (*word-size* $\not<$ s-total-ws-reqs (s-progs (s), *params*, *heap-size*)))
$\rightarrow$ proper-p-alistp (pair-formals-with-addresses (*params*,
cdr (lr-data-seg-table (s-progs (s),
params,
heap-size))),
lr->p (s->lr1 (s,

```
                                        p-state (pc,
                                                 ctrl-stk,
                                                 temp-stk,
                                                 prog-seg,
                                                 car (lr-data-seg-table (s-progs (s),
                                                                         params,
                                                                         heap-size)),
                                                 max-ctrl-stk-size,
                                                 max-temp-stk-size,
                                                 word-size,
                                                 psw),
                                        cdr (lr-data-seg-table (s-progs (s),
                                                                params,
                                                                heap-size)))))
```

THEOREM: proper-p-alistp-lr-make-initial-temps

```
(lr-proper-heapp (data-seg)
 ∧  all-litatoms (temp-vars)
 ∧  lr-proper-p-areasp (data-seg))
→   proper-p-alistp (lr-make-initial-temps (temp-vars),
                     lr->p (s->lr1 (s,
                                    p-state (pc,
                                             ctrl-stk,
                                             temp-stk,
                                             prog-seg,
                                             data-seg,
                                             max-ctrl,
                                             max-temp,
                                             word-size,
                                             psw),
                                    table)))
```

THEOREM: lr-minimum-heapp-lr-data-seg-table-body

```
(lr-proper-free-listp (data-seg)
 ∧  lr-proper-p-areasp (data-seg)
 ∧  definedp (LR-HEAP-NAME, data-seg)
 ∧  lr-boundary-nodep (lr-max-node (data-seg)))
→   (lr-minimum-heapp (car (lr-data-seg-table-body (flag, body, data-seg, table)))
      =  lr-minimum-heapp (data-seg))
```

THEOREM: lr-count-free-nodes-lr-compile-quote-s-heap-reqs-flag-t

```
(lr-proper-free-listp (data-seg)
 ∧  lr-proper-p-areasp (data-seg)
 ∧  definedp (LR-HEAP-NAME, data-seg)
 ∧  lr-minimum-heapp (data-seg)
```

```
 ∧   lr-boundary-nodep (lr-max-node (data-seg))
 ∧   (lr-count-free-nodes (fetch (LR-FP-ADDR, data-seg),
                          lr-free-list-nodes (lr-max-node (data-seg),
                                              data-seg),
                          data-seg)
      ≮   s-heap-reqs (t, object, data-seg, table)))
 →   ((s-heap-reqs (t, object, data-seg, table)
       +   lr-count-free-nodes (fetch (identity (LR-FP-ADDR),
                                       car (lr-compile-quote (t,
                                                              object,
                                                              data-seg,
                                                              table))),
                                lr-free-list-nodes (lr-max-node (data-seg),
                                                    car (lr-compile-quote (t,
                                                                           object,
                                                                           data-seg,
                                                                           table))),
                                car (lr-compile-quote (t,
                                                       object,
                                                       data-seg,
                                                       table))))
      =   lr-count-free-nodes (fetch (identity (LR-FP-ADDR), data-seg),
                               lr-free-list-nodes (lr-max-node (data-seg),
                                                   data-seg),
                               data-seg))
```

THEOREM: lr-count-free-nodes-lr-data-seg-table-body-s-heap-reqs

```
(lr-proper-free-listp (data-seg)
 ∧   lr-proper-p-areasp (data-seg)
 ∧   definedp (LR-HEAP-NAME, data-seg)
 ∧   lr-minimum-heapp (data-seg)
 ∧   lr-boundary-nodep (lr-max-node (data-seg))
 ∧   (lr-count-free-nodes (fetch (LR-FP-ADDR, data-seg),
                          lr-free-list-nodes (lr-max-node (data-seg),
                                              data-seg),
                          data-seg)
      ≮   s-heap-reqs-body (flag, body, data-seg, table)))
 →   ((lr-count-free-nodes (fetch (LR-FP-ADDR,
                                   car (lr-data-seg-table-body (flag,
                                                                body,
                                                                data-seg,
                                                                table))),
                            lr-free-list-nodes (lr-max-node (data-seg),
                                                car (lr-data-seg-table-body (flag,
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{body},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{data\text{-}seg},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{table}))),\\
\qquad\qquad\qquad\qquad \text{car}\,(\text{lr-data-seg-table-body}\,(\mathit{flag},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{body},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{data\text{-}seg},\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{table})))\\
\quad + \quad \text{s-heap-reqs-body}\,(\mathit{flag},\ \mathit{body},\ \mathit{data\text{-}seg},\ \mathit{table}))\\
= \quad \text{lr-count-free-nodes}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \mathit{data\text{-}seg}),\\
\qquad\qquad\qquad\qquad \text{lr-free-list-nodes}\,(\text{lr-max-node}\,(\mathit{data\text{-}seg}),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{data\text{-}seg}),\\
\qquad\qquad\qquad\qquad \mathit{data\text{-}seg}))
\end{array}
$$

THEOREM: lr-data-seg-table-body-lr-good-pointerp-tablep-help1

$$
\begin{array}{l}
\textbf{let}\ \mathit{data\text{-}seg\text{-}tab}\ \ \textbf{be}\ \text{lr-data-seg-table-body}\,(\mathbf{t},\ \text{car}\,(\mathit{body}),\ \mathit{data\text{-}seg},\ \mathit{table})\\
\textbf{in}\\
(\text{listp}\,(\mathit{body})\\
\ \wedge\ \ \text{lr-proper-free-listp}\,(\mathit{data\text{-}seg})\\
\ \wedge\ \ \text{lr-proper-p-areasp}\,(\mathit{data\text{-}seg})\\
\ \wedge\ \ \text{definedp}\,(\text{LR-HEAP-NAME},\ \mathit{data\text{-}seg})\\
\ \wedge\ \ \text{lr-minimum-heapp}\,(\mathit{data\text{-}seg})\\
\ \wedge\ \ \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\mathit{data\text{-}seg}))\\
\ \wedge\ \ (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \mathit{data\text{-}seg}),\\
\qquad\qquad\qquad\qquad \text{lr-free-list-nodes}\,(\text{lr-max-node}\,(\mathit{data\text{-}seg}),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{data\text{-}seg}),\\
\qquad\qquad\qquad\qquad \mathit{data\text{-}seg})\\
\qquad \not< \ \ (\text{s-heap-reqs-body}\,(\mathbf{t},\ \text{car}\,(\mathit{body}),\ \mathit{data\text{-}seg},\ \mathit{table})\\
\qquad\qquad + \ \ \text{s-heap-reqs-body}\,(\texttt{'list},\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{cdr}\,(\mathit{body}),\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{car}\,(\mathit{data\text{-}seg\text{-}tab}),\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{cdr}\,(\mathit{data\text{-}seg\text{-}tab})))))\\
\to\ \ (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{identity}\,(\text{LR-FP-ADDR}),\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{car}\,(\mathit{data\text{-}seg\text{-}tab})),\\
\qquad\qquad\qquad\qquad \text{lr-free-list-nodes}\,(\text{lr-max-node}\,(\mathit{data\text{-}seg}),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{car}\,(\mathit{data\text{-}seg\text{-}tab})),\\
\qquad\qquad\qquad\qquad \text{car}\,(\mathit{data\text{-}seg\text{-}tab}))\\
\qquad \not< \ \ \text{s-heap-reqs-body}\,(\texttt{'list},\\
\qquad\qquad\qquad\qquad\qquad \text{cdr}\,(\mathit{body}),\\
\qquad\qquad\qquad\qquad\qquad \text{car}\,(\mathit{data\text{-}seg\text{-}tab}),\\
\qquad\qquad\qquad\qquad\qquad \text{cdr}\,(\mathit{data\text{-}seg\text{-}tab})))\ \textbf{endlet}
\end{array}
$$

THEOREM: lr-data-seg-table-body-lr-good-pointerp-tablep

$$
\begin{array}{l}
(\text{lr-proper-free-listp}\,(\mathit{data\text{-}seg})\\
\ \wedge\ \ \text{lr-proper-p-areasp}\,(\mathit{data\text{-}seg})
\end{array}
$$

$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\wedge$ lr-minimum-heapp (*data-seg*)
$\wedge$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\wedge$ s-data-seg-body-restrictedp (*flag*, *body*)
$\wedge$ definedp (**f**, *table*)
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
                lr-free-list-nodes (lr-max-node (*data-seg*),
                                data-seg),
                data-seg)
    $\not<$ s-heap-reqs-body (*flag*, *body*, *data-seg*, *table*)))
$\rightarrow$ lr-good-pointerp-tablep (cdr (lr-data-seg-table-body (*flag*,
                                body,
                                data-seg,
                                table)),
                    car (lr-data-seg-table-body (*flag*,
                                body,
                                data-seg,
                                table)))

THEOREM: lr-data-seg-table-list-lr-good-pointerp-tablep-helper-1
let *dst-body* **be** lr-data-seg-table-body (**t**,
                        s-body (car (*progs*)),
                        data-seg,
                        table)
in
(listp (*progs*)
$\wedge$ lr-proper-free-listp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\wedge$ lr-minimum-heapp (*data-seg*)
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
                lr-free-list-nodes (lr-max-node (*data-seg*),
                                data-seg),
                data-seg)
    $\not<$ (s-heap-reqs-body (**t**, s-body (car (*progs*)), *data-seg*, *table*)
        $+$ s-heap-reqs-list (cdr (*progs*),
                    car (*dst-body*),
                    cdr (*dst-body*)))))
$\rightarrow$ (lr-count-free-nodes (fetch (identity (LR-FP-ADDR), car (*dst-body*)),
                lr-free-list-nodes (lr-max-node (*data-seg*),
                                car (*dst-body*)),
                car (*dst-body*))

$\not<$ s-heap-reqs-list (cdr (*progs*),
car (*dst-body*),
cdr (*dst-body*))) **endlet**

THEOREM: lr-init-data-seg-table-lr-good-pointerp-tablep
(lr-proper-free-listp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ definedp (LR-HEAP-NAME, *data-seg*)
$\land$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\land$ lr-minimum-heapp (*data-seg*)
$\land$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\land$ s-init-data-seg-restrictedp (*params*)
$\land$ definedp (**f**, *table*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-init-heap-reqs (*params*, *data-seg*, *table*)))
$\rightarrow$ lr-good-pointerp-tablep (cdr (lr-init-data-seg-table (*params*,
data-seg,
table)),
car (lr-init-data-seg-table (*params*,
data-seg,
table)))

THEOREM: lr-proper-heapp-car-lr-data-seg-table-body
(lr-proper-heapp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\land$ s-data-seg-body-restrictedp (*flag*, *body*)
$\land$ definedp (**f**, *table*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs-body (*flag*, *body*, *data-seg*, *table*)))
$\rightarrow$ lr-proper-heapp (car (lr-data-seg-table-body (*flag*, *body*, *data-seg*, *table*)))

THEOREM: lr-proper-heapp-car-lr-data-seg-table-list
(lr-proper-heapp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\land$ s-data-seg-list-restrictedp (*progs*)
$\land$ definedp (**f**, *table*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),

lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs-list (*progs*, *data-seg*, *table*)))
$\rightarrow$ lr-proper-heapp (car (lr-data-seg-table-list (*progs*, *data-seg*, *table*)))

THEOREM: lr-proper-heapp-car-lr-init-data-seg-table
(lr-proper-heapp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\land$ s-init-data-seg-restrictedp (*params*)
$\land$ definedp (**f**, *table*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-init-heap-reqs (*params*, *data-seg*, *table*)))
$\rightarrow$ lr-proper-heapp (car (lr-init-data-seg-table (*params*, *data-seg*, *table*)))

THEOREM: lr-count-free-nodes-lr-init-data-seg-table-s-init-heap-reqs
(lr-proper-free-listp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ definedp (LR-HEAP-NAME, *data-seg*)
$\land$ lr-minimum-heapp (*data-seg*)
$\land$ lr-boundary-nodep (lr-max-node (*data-seg*))
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-init-heap-reqs (*params*, *data-seg*, *table*)))
$\rightarrow$ ((lr-count-free-nodes (fetch (LR-FP-ADDR,
car (lr-init-data-seg-table (*params*,
data-seg,
table))),
lr-free-list-nodes (lr-max-node (*data-seg*),
car (lr-init-data-seg-table (*params*,
data-seg,
table))),
car (lr-init-data-seg-table (*params*,
data-seg,
table)))
$+$ s-init-heap-reqs (*params*, *data-seg*, *table*))
$=$ lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),

data-seg),
data-seg))

THEOREM: lr-proper-heapp-car-lr-data-seg-table-helper-1
let *data-seg-table* **be** lr-compile-quote ('list,
list (**t**, 0),
lr-init-data-seg (*heap-size*),
list (cons (**f**, identity (LR-F-ADDR))))
in
((*heap-size* $\not<$ (2
+ 2
+ s-init-heap-reqs (*params*,
car (*data-seg-table*),
cdr (*data-seg-table*))
+ *x*))
$\wedge$ (*max-addr* = lr-max-node (car (*data-seg-table*))))
$\rightarrow$ (lr-count-free-nodes (fetch (identity (LR-FP-ADDR),
car (lr-init-data-seg-table (*params*,
car (*data-seg-table*),
cdr (*data-seg-table*)))),
lr-free-list-nodes (*max-addr*,
car (lr-init-data-seg-table (*params*,
car (*data-seg-table*),
cdr (*data-seg-table*)))),
car (lr-init-data-seg-table (*params*,
car (*data-seg-table*),
cdr (*data-seg-table*))))
$\not<$ *x*) **endlet**

THEOREM: lr-proper-heapp-car-lr-data-seg-table
((*heap-size* $\not<$ s-total-heap-reqs (*progs*, *params*, *heap-size*))
$\wedge$ s-restrictedp (*progs*, *params*))
$\rightarrow$ lr-proper-heapp (car (lr-data-seg-table (*progs*, *params*, *heap-size*)))

THEOREM: litatom-car-gensym
litatom (car (gensym (*initial*, *num-list*, *atom-list*)))

THEOREM: all-litatoms-strip-cdrs-lr-make-temp-name-alist-1
all-litatoms (strip-cdrs (lr-make-temp-name-alist-1 (*initial*,
num-list,
temp-list,
formals)))

THEOREM: all-litatoms-strip-cdrs-lr-make-temp-name-alist
all-litatoms (strip-cdrs (lr-make-temp-name-alist (*temp-list*, *formals*)))

THEOREM: lr-proper-p-areasp-car-lr-data-seg-table
lr-proper-p-areasp (car (lr-data-seg-table (*progs*, *params*, *heap-size*)))

THEOREM: plist-strip-cdrs
plist (strip-cdrs (x)) = strip-cdrs (x)

THEOREM: lr-make-temp-name-alist-1-plist-arg-1
lr-make-temp-name-alist-1 (*initial*, *num-list*, plist (*temp-list*), *formals*)
= lr-make-temp-name-alist-1 (*initial*, *num-list*, *temp-list*, *formals*)

THEOREM: lr-make-temp-name-alist-plist-arg-1
lr-make-temp-name-alist (plist (*temp-list*), *formals*)
= lr-make-temp-name-alist (*temp-list*, *formals*)

THEOREM: length-lr-make-initial-temps
length (lr-make-initial-temps (*temp-vars*)) = length (*temp-vars*)

THEOREM: length-strip-cdrs
length (strip-cdrs (*alist*)) = length (*alist*)

THEOREM: length-pair-formals-with-addresses
length (pair-formals-with-addresses (*formals*, *alist*)) = length (*formals*)

THEOREM: s-good-statep-length-s-temp-list
s-good-statep (s, c)
→ (length (s-temp-list (s-prog (s))) = length (s-temps (s)))

THEOREM: plistp-comp-programs-1
plistp (comp-programs-1 (*progs*))

THEOREM: proper-p-instructionp-ret
proper-p-instructionp (`'(ret)`, *name*, p)

THEOREM: proper-p-instructionp-eq
proper-p-instructionp (`'(eq)`, *name*, p)

THEOREM: proper-p-instructionp-fetch
proper-p-instructionp (`'(fetch)`, *name*, p)

THEOREM: proper-p-instructionp-deposit
proper-p-instructionp (`'(deposit)`, *name*, p)

THEOREM: proper-p-instructionp-add-addr
proper-p-instructionp (`'(add-addr)`, *name*, p)

THEOREM: proper-p-instructionp-pop-global-free-ptr
lr-proper-heapp (p-data-segment (l))
→ proper-p-instructionp (`'(pop-global free-ptr)`, *name*, lr->p (l))

THEOREM: proper-p-instructionp-push-global-free-ptr
lr-proper-heapp (p-data-segment (l))
$\rightarrow$ proper-p-instructionp (`'(push-global free-ptr)`, *name*, lr->p (l))

THEOREM: proper-p-instructionp-push-local-temp-car
lr-programs-properp (l, *table*)
$\rightarrow$ proper-p-instructionp (`'(push-local x)`, `'car`, lr->p (l))

THEOREM: proper-p-instructionp-push-local-temp-cdr
lr-programs-properp (l, *table*)
$\rightarrow$ proper-p-instructionp (`'(push-local x)`, `'cdr`, lr->p (l))

THEOREM: proper-p-instructionp-push-local-temp-cons
lr-programs-properp (l, *table*)
$\rightarrow$ proper-p-instructionp (`'(push-local temp)`, `'cons`, lr->p (l))

THEOREM: proper-p-instructionp-set-local-temp-cons
lr-programs-properp (l, *table*)
$\rightarrow$ proper-p-instructionp (`'(set-local temp)`, `'cons`, lr->p (l))

THEOREM: proper-labeled-p-instructionsp-append
plistp (*instrs1*)
$\rightarrow$ (proper-labeled-p-instructionsp (append (*instrs1*, *instrs2*), *name*, *p*)
  = (proper-labeled-p-instructionsp (*instrs1*, *name*, *p*)
    $\wedge$ proper-labeled-p-instructionsp (*instrs2*, *name*, *p*)))

THEOREM: lessp-number-cons-cur-expr-dv-1-listp
listp (cur-expr (*pos*, *body*))
$\rightarrow$ (number-cons (cur-expr (dv (*pos*, `1`), *body*))
  < number-cons (cur-expr (*pos*, *body*)))

THEOREM: lessp-number-cons-cur-expr-dv-2-listp
listp (cur-expr (*pos*, *body*))
$\rightarrow$ (number-cons (cur-expr (dv (*pos*, `2`), *body*))
  < number-cons (cur-expr (*pos*, *body*)))

THEOREM: lessp-number-cons-cur-expr-dv-3-listp
listp (cur-expr (*pos*, *body*))
$\rightarrow$ (number-cons (cur-expr (dv (*pos*, `3`), *body*))
  < number-cons (cur-expr (*pos*, *body*)))

THEOREM: lessp-number-cons-restn-cdr
(listp (*pos*)
 $\wedge$ (car (last (*pos*)) < length (cur-expr (butlast (*pos*), *body*)))
 $\wedge$ listp (cur-expr (butlast (*pos*), *body*)))
$\rightarrow$ (number-cons (restn (car (last (*pos*)), cdr (cur-expr (butlast (*pos*), *body*))))
  < number-cons (restn (car (last (*pos*)), cur-expr (butlast (*pos*), *body*))))

THEOREM: proper-p-instructionp-test-bool-and-jump-label
proper-p-instructionp (list (`'test-bool-and-jump`, x, *lab*), *name*, p)
$=$ find-labelp (*lab*, program-body (assoc (*name*, p-prog-segment (p))))

THEOREM: proper-p-instructionp-jump-label
proper-p-instructionp (list (`'jump`, *lab*), *name*, p)
$=$ find-labelp (*lab*, program-body (assoc (*name*, p-prog-segment (p))))

THEOREM: plist-lr-convert-num-to-ascii
plistp (lr-convert-num-to-ascii (*number*, *list*)) $=$ plistp (*list*)

THEOREM: zerop-lr-convert-num-to-ascii
(*number* $\simeq$ `0`)
$\rightarrow$ (lr-convert-num-to-ascii (*number*, *list*) $=$ cons (ASCII-0, *list*))

THEOREM: lr-convert-digit-to-ascii-equal
(lr-convert-digit-to-ascii (m) $=$ lr-convert-digit-to-ascii (n))
$=$ (fix (m) $=$ fix (n))

THEOREM: zerop-lr-convert-digit-to-ascii
(*number* $\simeq$ `0`) $\rightarrow$ (lr-convert-digit-to-ascii (*number*) $=$ ASCII-0)

THEOREM: equal-ascii-0-lr-convert-digit-to-ascii
(lr-convert-digit-to-ascii (*number*) $=$ identity (ASCII-0)) $=$ (*number* $\simeq$ `0`)

THEOREM: length-lr-convert-num-to-ascii
length (lr-convert-num-to-ascii (*number*, *list*)) $\not<$ (1 + length (*list*))

DEFINITION:
induct-hint-22 (n, m, *list*)
$=$ **if** $n <$ `10` **then t**
  elseif $m <$ `10` **then t**
  else induct-hint-22 ($n \div$ `10`,
    $m \div$ `10`,
    cons (lr-convert-digit-to-ascii (n **mod** `10`),
      list)) **endif**

THEOREM: lr-convert-num-to-ascii-equal-arg1
(lr-convert-num-to-ascii (x, *list1*) $=$ lr-convert-num-to-ascii (x, *list2*))
$=$ (*list1* $=$ *list2*)

DEFINITION:
induct-hint-23 (n, m, *list1*, *list2*)
$=$ **if** $n <$ `10` **then t**
  elseif $m <$ `10` **then t**
  else induct-hint-23 ($n \div$ `10`,

$m \div 10$,
cons (lr-convert-digit-to-ascii (n **mod** 10),
  list1),
cons (lr-convert-digit-to-ascii (m **mod** 10),
  list2)) **endif**

THEOREM: lr-convert-num-to-ascii-equal-arg2-lengths-helper-1
(length (*list1*) = length (*list2*))
$\rightarrow$ (lr-convert-num-to-ascii (*number*, cons (x, *list1*)) $\neq$ cons (y, *list2*))

THEOREM: lr-convert-num-to-ascii-equal-arg2-lengths
((length (*list1*) = length (*list2*)) $\wedge$ (*list1* $\neq$ *list2*))
$\rightarrow$ (lr-convert-num-to-ascii (x, *list1*)
  $\neq$ lr-convert-num-to-ascii (y, *list2*))

THEOREM: lr-convert-num-to-ascii-equal-arg2-helper-1
($x \neq y$)
$\rightarrow$ (lr-convert-num-to-ascii (w, cons (x, *list*))
  $\neq$ lr-convert-num-to-ascii (z, cons (y, *list*)))

THEOREM: lr-convert-num-to-ascii-equal-arg2
(lr-convert-num-to-ascii (m, *list*) = lr-convert-num-to-ascii (n, *list*))
= (fix (m) = fix (n))

THEOREM: lr-make-label-equal
(lr-make-label (m) = lr-make-label (n)) = (fix (m) = fix (n))

THEOREM: find-labelp-lr-make-label-label-instrs
find-labelp (lr-make-label (m), label-instrs (*instrs*, n))
= (($m \not< n$) $\wedge$ ($m <$ (n + length (*instrs*))))

THEOREM: find-labelp-lr-make-label-comp-body
($n <$ length (comp-body (*body*)))
$\rightarrow$ find-labelp (lr-make-label (n), comp-body (*body*))

THEOREM: label-instrs-append
($n \in \mathbf{N}$)
$\rightarrow$ (label-instrs (append (*instrs1*, *instrs2*), n)
  = append (label-instrs (*instrs1*, n),
    label-instrs (*instrs2*, n + length (*instrs1*))))

THEOREM: proper-p-temp-var-dclsp-all-litatoms-all-undef-addrs
(all-litatoms (strip-cars (*temp-var-dcls*))
 $\wedge$ all-undef-addrs (strip-cadrs (*temp-var-dcls*))
 $\wedge$ lr-proper-heapp (p-data-segment (p))
 $\wedge$ lr-proper-p-areasp (p-data-segment (p)))
$\rightarrow$ proper-p-temp-var-dclsp (*temp-var-dcls*, p)

THEOREM: plistp-label-instrs
plistp (label-instrs (*instrs*, *n*))

DEFINITION:
not-labelledp-instrs (*instrs*)
= **if** listp (*instrs*)
  then (¬ labelledp (car (*instrs*)))
    ∧ not-labelledp-instrs (cdr (*instrs*))
  else t endif

THEOREM: label-instrs-proper-labeled-p-instructionsp
(proper-labeled-p-instructionsp (*instrs*, *name*, *p*)
∧ not-labelledp-instrs (*instrs*))
→ proper-labeled-p-instructionsp (label-instrs (*instrs*, *n*), *name*, *p*)

THEOREM: not-labelledp-instrs-append
not-labelledp-instrs (append (*instrs1*, *instrs2*))
= (not-labelledp-instrs (*instrs1*) ∧ not-labelledp-instrs (*instrs2*))

THEOREM: not-labelledp-instrs-comp-body-1
not-labelledp-instrs (comp-body-1 (*flag*, *body*, *n*))

THEOREM: comp-body-1-list-not-listp
(¬ listp (*body*)) → (comp-body-1 (`'list`, *body*, *n*) = **nil**)

THEOREM: proper-labeled-p-instructionsp-nil
proper-labeled-p-instructionsp (**nil**, *name*, *p*)

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-1
(proper-labeled-p-instructionsp (comp-body-1 (**t**, *test-body*, *n1*),
  name,
  p-state (*pc*,
    ctrl-stk,
    temp-stk,
    comp-programs (*prog-seg*),
    data-seg,
    max-ctrl,
    max-temp,
    word-size,
    psw))
∧ proper-labeled-p-instructionsp (comp-body-1 (**t**, *then-body*, *n2*),
  name,
  p-state (*pc*,
    ctrl-stk,
    temp-stk,

comp-programs (*prog-seg*),
data-seg,
max-ctrl,
max-temp,
word-size,
psw))

$\wedge$ proper-labeled-p-instructionsp (*else-instrs*,
name,
p-state (*pc*,
ctrl-stk,
temp-stk,
comp-programs (*prog-seg*),
data-seg,
max-ctrl,
max-temp,
word-size,
psw))

$\wedge$ lr-proper-heapp (*data-seg*)

$\wedge$ lr-proper-p-areasp (*data-seg*)

$\wedge$ definedp (*name*, *prog-seg*)

$\wedge$ (((*n*
$+$ lr-p-c-size (**t**, *test-body*)
$+$ lr-p-c-size (**t**, *then-body*)
$+$ length (*else-instrs*)
$+$ 4) $-$ 1)
$<$ lr-p-c-size (**t**, program-body (assoc (*name*, *prog-seg*)))))

$\rightarrow$ proper-labeled-p-instructionsp (comp-if (comp-body-1 (**t**, *test-body*, *n1*),
comp-body-1 (**t**, *then-body*, *n2*),
else-instrs,
n),
name,
p-state (*pc*,
ctrl-stk,
temp-stk,
comp-programs (*prog-seg*),
data-seg,
max-ctrl,
max-temp,
word-size,
psw))

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-2
(listp (*body*)
$\wedge$ ((car (*body*) $=$ S-TEMP-FETCH) $\vee$ (car (*body*) $=$ S-TEMP-TEST))

∧ lr-proper-exprp (**t**,
body,
program-names,
formal-vars (assoc (*name*, *prog-seg*)),
strip-cars (temp-var-dcls (assoc (*name*, *prog-seg*))),
table)
∧ definedp (*name*, *prog-seg*))
→ proper-labeled-p-instructionsp (list (list (`'push-local`, caddr (*body*))),
name,
p-state (*pc*,
ctrl-stk,
temp-stk,
comp-programs (*prog-seg*),
data-seg,
max-ctrl,
max-temp,
word-size,
psw))

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-3
(listp (*body*)
∧ ((car (*body*) = S-TEMP-EVAL) ∨ (car (*body*) = S-TEMP-TEST))
∧ lr-proper-exprp (**t**,
body,
program-names,
formal-vars (assoc (*name*, *prog-seg*)),
strip-cars (temp-var-dcls (assoc (*name*, *prog-seg*))),
table)
∧ definedp (*name*, *prog-seg*))
→ proper-labeled-p-instructionsp (list (list (`'set-local`, caddr (*body*))),
name,
p-state (*pc*,
ctrl-stk,
temp-stk,
comp-programs (*prog-seg*),
data-seg,
max-ctrl,
max-temp,
word-size,
psw))

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-4
(listp (*body*)
∧ (car (*body*) = S-TEMP-TEST)

```
∧   proper-labeled-p-instructionsp (comp-body-1 (t, cadr (body), n + 4),
                                    name,
                                    p-state (pc,
                                             ctrl-stk,
                                             temp-stk,
                                             comp-programs (prog-seg),
                                             data-seg,
                                             max-ctrl,
                                             max-temp,
                                             word-size,
                                             psw))
∧   lr-proper-exprp (t,
                     body,
                     program-names,
                     formal-vars (assoc (name, prog-seg)),
                     strip-cars (temp-var-dcls (assoc (name, prog-seg))),
                     table)
∧   lr-proper-heapp (data-seg)
∧   lr-proper-p-areasp (data-seg)
∧   (lr-p-c-size (t, program-body (assoc (name, prog-seg)))
      ≮   (n + lr-p-c-size (t, cadr (body)) + 7))
∧   definedp (name, prog-seg))
→   proper-labeled-p-instructionsp (comp-temp-test (body,
                                                    comp-body-1 (t,
                                                                 cadr (body),
                                                                 n + 4),
                                                    n),
                                    name,
                                    p-state (pc,
                                             ctrl-stk,
                                             temp-stk,
                                             comp-programs (prog-seg),
                                             data-seg,
                                             max-ctrl,
                                             max-temp,
                                             word-size,
                                             psw))
```

DEFINITION:

```
assoc-cdr (x, l)
=   if listp (l)
    then if x = cdar (l)  then car (l)
         else assoc-cdr (x, cdr (l)) endif
    else f endif
```

THEOREM: lr-s-similar-const-table-lr-valp-member-strip-cdrs

$((addr \in \text{strip-cdrs}(table)) \land \text{lr-s-similar-const-table}(table,\ data\text{-}seg))$
$\rightarrow\ \text{lr-valp}(\text{car}(\text{assoc-cdr}(addr,\ table)),\ addr,\ data\text{-}seg)$

THEOREM: lr-s-similar-const-table-type-addr-member-strip-cdrs

$((addr \in \text{strip-cdrs}(table)) \land \text{lr-s-similar-const-table}(table,\ data\text{-}seg))$
$\rightarrow\ ((\text{type}(addr) = \texttt{'addr})$
$\quad \land\ (\text{cddr}(addr) = \textbf{nil})$
$\quad \land\ \text{listp}(addr)$
$\quad \land\ \text{adpp}(\text{untag}(addr),\ data\text{-}seg)$
$\quad \land\ \text{lr-boundary-nodep}(addr)$
$\quad \land\ (\text{area-name}(addr) = \text{identity}(\text{LR-HEAP-NAME}))$
$\quad \land\ (\text{type}(\text{fetch}(\text{add-addr}(addr,\ \text{identity}(\text{LR-REF-COUNT-OFFSET})),\ data\text{-}seg)) = \texttt{'nat}))$

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-5

$(\text{listp}(body)$
$\land\ (\text{car}(body) = \texttt{'quote})$
$\land\ \text{lr-proper-heapp}(data\text{-}seg)$
$\land\ \text{lr-proper-p-areasp}(data\text{-}seg)$
$\land\ \text{lr-proper-exprp}(\mathbf{t},\ body,\ \text{strip-logic-fnames}(\text{cdr}(prog\text{-}seg)),\ \text{formal-vars}(\text{assoc}(name,\ prog\text{-}seg)),\ \text{strip-cars}(\text{temp-var-dcls}(\text{assoc}(name,\ prog\text{-}seg))),\ table)$
$\land\ \text{lr-s-similar-const-table}(table,\ data\text{-}seg)$
$\land\ \text{definedp}(name,\ prog\text{-}seg))$
$\rightarrow\ \text{proper-labeled-p-instructionsp}(\text{list}(\text{list}(\texttt{'push-constant},\ \text{cadr}(body))),\ name,\ \text{p-state}(pc,\ ctrl\text{-}stk,\ temp\text{-}stk,\ \text{comp-programs}(prog\text{-}seg),\ data\text{-}seg,\ max\text{-}ctrl,\ max\text{-}temp,\ word\text{-}size,\ psw))$

EVENT: Disable lr-s-similar-const-table-type-addr-member-strip-cdrs.

THEOREM: lr-proper-exprp-flag-list-cdr-funcall

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\mathit{body})} \\
\land & (\text{car}\,(\mathit{body}) \neq \texttt{'if}) \\
\land & (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-FETCH}) \\
\land & (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-EVAL}) \\
\land & (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-TEST}) \\
\land & (\text{car}\,(\mathit{body}) \neq \texttt{'quote}) \\
\land & \text{lr-proper-exprp}\,(\mathbf{t}, \mathit{body}, \mathit{program\text{-}names}, \mathit{formals}, \mathit{temps}, \mathit{table})) \\
\to & \text{lr-proper-exprp}\,(\texttt{'list}, \text{cdr}\,(\mathit{body}), \mathit{program\text{-}names}, \mathit{formals}, \mathit{temps}, \mathit{table})
\end{array}
$$

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-6-1

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\mathit{body})} \\
\land & (\text{car}\,(\mathit{body}) \neq \texttt{'if}) \\
\land & (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-FETCH}) \\
\land & (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-EVAL}) \\
\land & (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-TEST}) \\
\land & (\text{car}\,(\mathit{body}) \neq \texttt{'quote}) \\
\land & \text{lr-proper-exprp}\,(\mathbf{t}, \mathit{body}, \text{strip-logic-fnames}\,(\text{cdr}\,(\mathit{prog\text{-}seg})), \mathit{formals}, \mathit{temps}, \mathit{table}) \\
\land & \text{definedp}\,(\text{car}\,(\mathit{body}), \text{P-RUNTIME-SUPPORT-PROGRAMS})) \\
\to & \text{proper-labeled-p-instructionsp}\,(\text{list}\,(\text{list}\,(\texttt{'call}, \text{car}\,(\mathit{body}))), \\
& \quad \mathit{name}, \\
& \quad \text{p-state}\,(\mathit{pc}, \mathit{ctrl\text{-}stk}, \mathit{temp\text{-}stk}, \text{comp-programs}\,(\mathit{prog\text{-}seg}), \mathit{data\text{-}seg}, \mathit{max\text{-}ctrl}, \mathit{max\text{-}temp}, \mathit{word\text{-}size}, \mathit{psw}))
\end{array}
$$

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-6-2

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\mathit{body})} \\
\land & (\text{car}\,(\mathit{body}) \neq \texttt{'if}) \\
\land & (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-FETCH}) \\
\land & (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-EVAL}) \\
\land & (\text{car}\,(\mathit{body}) \neq \text{S-TEMP-TEST}) \\
\land & (\text{car}\,(\mathit{body}) \neq \texttt{'quote}) \\
\land & \text{lr-proper-exprp}\,(\mathbf{t}, \\
& \quad \mathit{body},
\end{array}
$$

```
                        strip-logic-fnames (cdr (prog-seg)),
                        formals,
                        temps,
                        table)
 ∧  (¬ definedp (car (body), P-RUNTIME-SUPPORT-PROGRAMS))
 ∧  all-user-fnamesp (cdr (strip-cars (prog-seg)))
 ∧  definedp (name, prog-seg))
 →  proper-labeled-p-instructionsp (list (list ('call,
                                                user-fname (car (body)))),
                                    name,
                                    p-state (pc,
                                             ctrl-stk,
                                             temp-stk,
                                             comp-programs (prog-seg),
                                             data-seg,
                                             max-ctrl,
                                             max-temp,
                                             word-size,
                                             psw))
```

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-7

```
((¬ listp (body))
 ∧  lr-proper-exprp (t,
                     body,
                     program-names,
                     formal-vars (assoc (name, prog-seg)),
                     temps,
                     table)
 ∧  definedp (name, prog-seg))
 →  proper-labeled-p-instructionsp (list (list ('push-local, body)),
                                    name,
                                    p-state (pc,
                                             ctrl-stk,
                                             temp-stk,
                                             comp-programs (prog-seg),
                                             data-seg,
                                             max-ctrl,
                                             max-temp,
                                             word-size,
                                             psw))
```

THEOREM: not-definedp-not-listp

```
(¬ listp (alist)) → (¬ definedp (name, alist))
```

THEOREM: proper-labeled-p-instructionsp-comp-body-1

(lr-proper-heapp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ all-user-fnamesp (cdr (strip-cars (*prog-seg*)))
$\land$ lr-proper-exprp (**if** *flag* = `'list` **then** `'list`
else t endif,
body,
strip-logic-fnames (cdr (*prog-seg*)),
formal-vars (assoc (*name*, *prog-seg*)),
strip-cars (temp-var-dcls (assoc (*name*, *prog-seg*))),
table)
$\land$ lr-s-similar-const-table (*table*, *data-seg*)
$\land$ ((n + lr-p-c-size (**if** *flag* = `'list` **then** `'list`
else t endif,
body))
< (1 + lr-p-c-size (**t**, program-body (assoc (*name*, *prog-seg*)))))
$\land$ definedp (*name*, *prog-seg*))
$\rightarrow$ proper-labeled-p-instructionsp (comp-body-1 (*flag*, *body*, n),
name,
p-state (*pc*,
ctrl-stk,
temp-stk,
comp-programs (*prog-seg*),
data-seg,
max-ctrl,
max-temp,
word-size,
psw))

THEOREM: all-undef-addrs-strip-cadrs-lr-make-temp-var-dcls
all-undef-addrs (strip-cadrs (lr-make-temp-var-dcls (*temp-alist*)))

THEOREM: proper-p-programp-p-car-code
(lr-programs-properp (l, *table*)
$\land$ (p-word-size (l) $\in \mathbf{N}$)
$\land$ (p-word-size (l) $\not<$ S-MAX-SUBR-REQS)
$\land$ lr-proper-heapp (p-data-segment (l))
$\land$ lr-proper-p-areasp (p-data-segment (l)))
$\rightarrow$ proper-p-programp (P-CAR-CODE, lr->p (l))

THEOREM: proper-p-programp-p-cdr-code
(lr-programs-properp (l, *table*)
$\land$ (p-word-size (l) $\in \mathbf{N}$)
$\land$ (p-word-size (l) $\not<$ S-MAX-SUBR-REQS)
$\land$ lr-proper-heapp (p-data-segment (l))
$\land$ lr-proper-p-areasp (p-data-segment (l)))

$\rightarrow$ proper-p-programp (P-CDR-CODE, lr->p (l))

THEOREM: proper-p-programp-p-cons-code
(lr-programs-properp $(l,\ table)$
$\wedge$ (p-word-size $(l) \in \mathbf{N}$)
$\wedge$ (p-word-size $(l) \not<$ S-MAX-SUBR-REQS)
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-proper-p-areasp (p-data-segment (l)))
$\rightarrow$ proper-p-programp (P-CONS-CODE, lr->p (l))

THEOREM: proper-p-programp-p-false-code
(lr-programs-properp $(l,\ table)$
$\wedge$ (p-word-size $(l) \in \mathbf{N}$)
$\wedge$ (p-word-size $(l) \not<$ log (2, LR-FALSE-TAG))
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-proper-p-areasp (p-data-segment (l)))
$\rightarrow$ proper-p-programp (P-FALSE-CODE, lr->p (l))

THEOREM: proper-p-programp-p-falsep-code
(lr-programs-properp $(l,\ table)$
$\wedge$ (p-word-size $(l) \in \mathbf{N}$)
$\wedge$ (p-word-size $(l) \not<$ log (2, LR-FALSE-TAG))
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-proper-p-areasp (p-data-segment (l)))
$\rightarrow$ proper-p-programp (P-FALSEP-CODE, lr->p (l))

THEOREM: proper-p-programp-p-listp-code
(lr-programs-properp $(l,\ table)$
$\wedge$ (p-word-size $(l) \in \mathbf{N}$)
$\wedge$ (p-word-size $(l) \not<$ S-MAX-SUBR-REQS)
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-proper-p-areasp (p-data-segment (l)))
$\rightarrow$ proper-p-programp (P-LISTP-CODE, lr->p (l))

THEOREM: proper-p-programp-p-nlistp-code
(lr-programs-properp $(l,\ table)$
$\wedge$ (p-word-size $(l) \in \mathbf{N}$)
$\wedge$ (p-word-size $(l) \not<$ S-MAX-SUBR-REQS)
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-proper-p-areasp (p-data-segment (l)))
$\rightarrow$ proper-p-programp (P-NLISTP-CODE, lr->p (l))

THEOREM: proper-p-programp-p-true-code
(lr-programs-properp $(l,\ table)$
$\wedge$ (p-word-size $(l) \in \mathbf{N}$)

$\wedge$ (p-word-size (l) $\not<$ log (2, LR-TRUE-TAG))
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-proper-p-areasp (p-data-segment (l)))
$\rightarrow$ proper-p-programp (P-TRUE-CODE, lr->p (l))

THEOREM: proper-p-programp-p-truep-code
(lr-programs-properp (l, *table*)
$\wedge$ (p-word-size (l) $\in \mathbf{N}$)
$\wedge$ (p-word-size (l) $\not<$ log (2, LR-TRUE-TAG))
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-proper-p-areasp (p-data-segment (l)))
$\rightarrow$ proper-p-programp (P-TRUEP-CODE, lr->p (l))

THEOREM: lr-s-similar-const-table-lr-data-seg-table-body
(lr-proper-p-areasp (*data-seg*)
$\wedge$ lr-proper-heapp (*data-seg*)
$\wedge$ lr-s-similar-const-table (*table*, *data-seg*)
$\wedge$ s-data-seg-body-restrictedp (*flag*, *body*)
$\wedge$ definedp (**f**, *table*)
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs-body (*flag*, *body*, *data-seg*, *table*)))
$\rightarrow$ lr-s-similar-const-table (cdr (lr-data-seg-table-body (*flag*,
body,
data-seg,
table)),
car (lr-data-seg-table-body (*flag*,
body,
data-seg,
table)))

THEOREM: lr-s-similar-const-table-lr-data-seg-table-list
(lr-proper-p-areasp (*data-seg*)
$\wedge$ lr-proper-heapp (*data-seg*)
$\wedge$ lr-s-similar-const-table (*table*, *data-seg*)
$\wedge$ s-data-seg-list-restrictedp (*progs*)
$\wedge$ definedp (**f**, *table*)
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs-list (*progs*, *data-seg*, *table*)))
$\rightarrow$ lr-s-similar-const-table (cdr (lr-data-seg-table-list (*progs*,

$data\text{-}seg$,
$table$)),
car (lr-data-seg-table-list ($progs$,
$data\text{-}seg$,
$table$)))

THEOREM: lr-s-similar-const-table-lr-init-data-seg-table
(lr-proper-p-areasp ($data\text{-}seg$)
∧ lr-proper-heapp ($data\text{-}seg$)
∧ lr-s-similar-const-table ($table$, $data\text{-}seg$)
∧ s-init-data-seg-restrictedp ($params$)
∧ definedp (**f**, $table$)
∧ (lr-count-free-nodes (fetch (LR-FP-ADDR, $data\text{-}seg$),
lr-free-list-nodes (lr-max-node ($data\text{-}seg$),
$data\text{-}seg$),
$data\text{-}seg$)
$\not<$ s-init-heap-reqs ($params$, $data\text{-}seg$, $table$)))
→ lr-s-similar-const-table (cdr (lr-init-data-seg-table ($params$,
$data\text{-}seg$,
$table$)),
car (lr-init-data-seg-table ($params$,
$data\text{-}seg$,
$table$)))

THEOREM: lr-s-similar-const-table-cdr-car-lr-data-seg-table
(($heap\text{-}size$ $\not<$ s-total-heap-reqs ($progs$, $params$, $heap\text{-}size$))
∧ s-restrictedp ($progs$, $params$))
→ lr-s-similar-const-table (cdr (lr-data-seg-table ($progs$, $params$, $heap\text{-}size$)),
car (lr-data-seg-table ($progs$, $params$, $heap\text{-}size$)))

DEFINITION:
induct-hint-3 ($flag$, pos, $prog$)
= **if** $flag$ = `'list`
then if $pos \simeq$ **nil then t**
elseif listp (restn (car (last (pos)),
cur-expr (butlast (pos), s-body ($prog$))))
then induct-hint-3 (**t**, pos, $prog$)
∧ induct-hint-3 (`'list`, nx (pos), $prog$)
else t endif
elseif listp (cur-expr (pos, s-body ($prog$)))
then if car (cur-expr (pos, s-body ($prog$))) = `'if`
then induct-hint-3 (**t**, dv (pos, `1`), $prog$)
∧ induct-hint-3 (**t**, dv (pos, `2`), $prog$)
∧ induct-hint-3 (**t**, dv (pos, `3`), $prog$)
elseif car (cur-expr (pos, s-body ($prog$))) = S-TEMP-FETCH **then t**

elseif (car (cur-expr (*pos*, s-body (*prog*))) = S-TEMP-EVAL)
∨ (car (cur-expr (*pos*, s-body (*prog*))) = S-TEMP-TEST)
then induct-hint-3 (`t`, dv (*pos*, `1`), *prog*)
elseif car (cur-expr (*pos*, s-body (*prog*))) = `'quote` **then** `t`
else induct-hint-3 (`'list`, dv (*pos*, `1`), *prog*) **endif**
else t endif

THEOREM: lr-proper-exprp-p-lr-compile-programs-helper-1
listp (*pos*)
→ (lr-compile-body (`t`,
get (car (last (*pos*)), cur-expr (butlast (*pos*), *body*)),
temp-alist,
table)
= lr-compile-body (`t`, cur-expr (*pos*, *body*), *temp-alist*, *table*))

THEOREM: good-posp1-cons-lessp-4-if-s-proper-exprp
((car (cur-expr (*pos*, *body*)) = `'if`)
∧ good-posp1 (*pos*, *body*)
∧ s-proper-exprp (`t`, *body*, *program-names*, *formals*, *temp-list*))
→ (good-posp1 (dv (*pos*, `1`), *body*)
∧ good-posp1 (dv (*pos*, `2`), *body*)
∧ good-posp1 (dv (*pos*, `3`), *body*))

THEOREM: lr-proper-exprp-flag-list-cons
lr-proper-exprp (`'list`, cons (*car*, *cdr*), *program-names*, *formals*, *temps*, *table*)
= (lr-proper-exprp (`'list`, *cdr*, *program-names*, *formals*, *temps*, *table*)
∧ lr-proper-exprp (`t`, *car*, *program-names*, *formals*, *temps*, *table*))

THEOREM: lr-proper-exprp-flag-list-nil
lr-proper-exprp (`'list`, **nil**, *program-names*, *formals*, *temps*, *table*)

THEOREM: length-3-cdr-cddr-not-nil
((length (x) = `3`) ∧ plistp (x))
→ ((cdddr (x) = **nil**) ∧ (cddr (x) ≠ **nil**) ∧ (cdr (x) ≠ **nil**))

THEOREM: lr-proper-exprp-flag-not-list-cons-if-helper
(*flag* ≠ `'list`)
→ (lr-proper-exprp (*flag*,
list (`'if`, *test*, *then*, *else*),
program-names,
formals,
temp-alist,
table)
= ((`'if` ∉ *program-names*)
∧ lr-proper-exprp (`t`,

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad \mathit{test}, \\
\qquad\qquad\qquad\qquad \mathit{program\text{-}names}, \\
\qquad\qquad\qquad\qquad \mathit{formals}, \\
\qquad\qquad\qquad\qquad \mathit{temp\text{-}alist}, \\
\qquad\qquad\qquad\qquad \mathit{table}) \\
\quad \land \quad \text{lr-proper-exprp}\,(\mathbf{t}, \\
\qquad\qquad\qquad\qquad \mathit{then}, \\
\qquad\qquad\qquad\qquad \mathit{program\text{-}names}, \\
\qquad\qquad\qquad\qquad \mathit{formals}, \\
\qquad\qquad\qquad\qquad \mathit{temp\text{-}alist}, \\
\qquad\qquad\qquad\qquad \mathit{table}) \\
\quad \land \quad \text{lr-proper-exprp}\,(\mathbf{t}, \\
\qquad\qquad\qquad\qquad \mathit{else}, \\
\qquad\qquad\qquad\qquad \mathit{program\text{-}names}, \\
\qquad\qquad\qquad\qquad \mathit{formals}, \\
\qquad\qquad\qquad\qquad \mathit{temp\text{-}alist}, \\
\qquad\qquad\qquad\qquad \mathit{table})))
\end{array}
$$

THEOREM: lr-proper-exprp-flag-not-list-cons-if

$$
\begin{array}{l}
((\mathit{flag} \neq \texttt{'list}) \\
\land \quad \text{listp}\,(\text{cur-expr}\,(\mathit{pos},\ \mathit{body})) \\
\land \quad (\text{car}\,(\text{cur-expr}\,(\mathit{pos},\ \mathit{body})) = \texttt{'if}) \\
\land \quad \text{good-posp1}\,(\mathit{pos},\ \mathit{body}) \\
\land \quad \text{lr-proper-exprp}\,(\mathbf{t}, \\
\qquad\qquad\qquad \text{lr-compile-body}\,(\mathbf{t}, \\
\qquad\qquad\qquad\qquad\qquad \text{cur-expr}\,(\text{dv}\,(\mathit{pos},\ \texttt{3}),\ \mathit{body}), \\
\qquad\qquad\qquad\qquad\qquad \mathit{temp\text{-}alist}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{table}), \\
\qquad\qquad\qquad \mathit{program\text{-}names}, \\
\qquad\qquad\qquad \mathit{formals}, \\
\qquad\qquad\qquad \text{strip-cdrs}\,(\mathit{temp\text{-}alist}), \\
\qquad\qquad\qquad \mathit{table}) \\
\land \quad \text{lr-proper-exprp}\,(\mathbf{t}, \\
\qquad\qquad\qquad \text{lr-compile-body}\,(\mathbf{t}, \\
\qquad\qquad\qquad\qquad\qquad \text{cur-expr}\,(\text{dv}\,(\mathit{pos},\ \texttt{2}),\ \mathit{body}), \\
\qquad\qquad\qquad\qquad\qquad \mathit{temp\text{-}alist}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{table}), \\
\qquad\qquad\qquad \mathit{program\text{-}names}, \\
\qquad\qquad\qquad \mathit{formals}, \\
\qquad\qquad\qquad \text{strip-cdrs}\,(\mathit{temp\text{-}alist}), \\
\qquad\qquad\qquad \mathit{table}) \\
\land \quad \text{lr-proper-exprp}\,(\mathbf{t}, \\
\qquad\qquad\qquad \text{lr-compile-body}\,(\mathbf{t}, \\
\qquad\qquad\qquad\qquad\qquad \text{cur-expr}\,(\text{dv}\,(\mathit{pos},\ \texttt{1}),\ \mathit{body}),
\end{array}
$$

```
                                      temp-alist,
                                      table),
                      program-names,
                      formals,
                      strip-cdrs (temp-alist),
                      table)
 ∧   s-proper-exprp (t, body, program-names, formals, strip-cars (temp-alist)))
→    lr-proper-exprp (flag,
                      cons ('if,
                            lr-compile-body ('list,
                                             cdr (cur-expr (pos, body)),
                                             temp-alist,
                                             table)),
                      program-names,
                      formals,
                      strip-cdrs (temp-alist),
                      table)
```

THEOREM: lr-proper-exprp-flag-not-list-cons-temp-fetch

```
((flag ≠ 'list)
 ∧   good-posp1 (pos, body)
 ∧   listp (cur-expr (pos, body))
 ∧   (car (cur-expr (pos, body)) = S-TEMP-FETCH)
 ∧   s-proper-exprp (t, body, program-names, formals, strip-cars (temp-alist)))
→    lr-proper-exprp (flag,
                      list (identity (S-TEMP-FETCH),
                            x,
                            cdr (assoc (cadr (cur-expr (pos, body)), temp-alist))),
                      program-names,
                      formals,
                      strip-cdrs (temp-alist),
                      table)
```

THEOREM: good-posp1-dv-1-temp-eval-test

```
(good-posp1 (pos, body)
 ∧   ((car (cur-expr (pos, body)) = S-TEMP-EVAL)
      ∨   (car (cur-expr (pos, body)) = S-TEMP-TEST))
 ∧   listp (cur-expr (pos, body)))
→    good-posp1 (dv (pos, 1), body)
```

THEOREM: length-last-fact

```
(length (x) = 1) → (last (x) = x)
```

THEOREM: lr-proper-exprp-flag-not-list-cons-temp-eval

```
((flag ≠ 'list)
```

$\wedge$ listp (cur-expr (*pos*, *body*))
$\wedge$ good-posp1 (*pos*, *body*)
$\wedge$ (car (cur-expr (*pos*, *body*)) = S-TEMP-EVAL)
$\wedge$ lr-proper-exprp (**t**,
lr-compile-body (**t**,
cur-expr (dv (*pos*, 1), *body*),
temp-alist,
table),
program-names,
formals,
strip-cdrs (*temp-alist*),
table)
$\wedge$ s-proper-exprp (**t**, *body*, *program-names*, *formals*, strip-cars (*temp-alist*)))
$\rightarrow$ lr-proper-exprp (*flag*,
list (identity (S-TEMP-EVAL),
lr-compile-body (**t**,
cadr (cur-expr (*pos*, *body*)),
temp-alist,
table),
cdr (assoc (cadr (cur-expr (*pos*, *body*)), *temp-alist*))),
program-names,
formals,
strip-cdrs (*temp-alist*),
table)

THEOREM: lr-proper-exprp-flag-not-list-cons-temp-test
((*flag* $\neq$ **'list**)
$\wedge$ listp (cur-expr (*pos*, *body*))
$\wedge$ good-posp1 (*pos*, *body*)
$\wedge$ (car (cur-expr (*pos*, *body*)) = S-TEMP-TEST)
$\wedge$ lr-proper-exprp (**t**,
lr-compile-body (**t**,
cur-expr (dv (*pos*, 1), *body*),
temp-alist,
table),
program-names,
formals,
strip-cdrs (*temp-alist*),
table)
$\wedge$ s-proper-exprp (**t**, *body*, *program-names*, *formals*, strip-cars (*temp-alist*)))
$\rightarrow$ lr-proper-exprp (*flag*,
list (identity (S-TEMP-TEST),
lr-compile-body (**t**,
cadr (cur-expr (*pos*, *body*)),

temp-alist,
table),
cdr (assoc (cadr (cur-expr (*pos*, *body*)), *temp-alist*))),
program-names,
formals,
strip-cdrs (*temp-alist*),
table)

DEFINITION:
lr-data-seg-table-body-n (*n*, *body*, *data-seg*, *table*)
= **if** $n \simeq$ `0` **then** cons (*data-seg*, *table*)
else lr-data-seg-table-body-n ($n - 1$,
cdr (*body*),
car (lr-data-seg-table-body (**t**,
car (*body*),
data-seg,
table)),
cdr (lr-data-seg-table-body (**t**,
car (*body*),
data-seg,
table))) **endif**

DEFINITION:
induct-hint-20 (*pos*, *body*, *data-seg*, *table*)
= **if** *pos* $\simeq$ **nil** **then** **t**
elseif *body* $\simeq$ **nil** **then** **f**
elseif car (*body*) = S-TEMP-FETCH **then** **f**
elseif (car (*body*) = S-TEMP-EVAL) $\vee$ (car (*body*) = S-TEMP-TEST)
then induct-hint-20 (cdr (*pos*), cadr (*body*), *data-seg*, *table*)
elseif car (*body*) = `'quote` **then** **f**
else induct-hint-20 (cdr (*pos*),
get (car (*pos*), *body*),
car (lr-data-seg-table-body-n (car (*pos*) $-$ 1,
cdr (*body*),
data-seg,
table)),
cdr (lr-data-seg-table-body-n (car (*pos*) $-$ 1,
cdr (*body*),
data-seg,
table))) **endif**

THEOREM: lr-data-seg-body-list-n
lr-data-seg-table-body (`'list`, *body*, *data-seg*, *table*)
= lr-data-seg-table-body-n (length (*body*), *body*, *data-seg*, *table*)

THEOREM: definedp-table-definedp-cdr-lr-data-seg-table-body-n
definedp (*object*, *table*)
$\rightarrow$ definedp (*object*, cdr (lr-data-seg-table-body-n (*n*, *body*, *data-seg*, *table*)))

THEOREM: definedp-lr-data-seg-body-list-n-not-lessp
(definedp (x, cdr (lr-data-seg-table-body-n (*n*, *body*, *data-seg*, *table*)))
$\land$ ($m \not< n$))
$\rightarrow$ definedp (x, cdr (lr-data-seg-table-body-n (*m*, *body*, *data-seg*, *table*)))

THEOREM: lr-data-seg-table-body-add1-opener
($n <$ length (*body*))
$\rightarrow$ (lr-data-seg-table-body-n ($1 + n$, *body*, *data-seg*, *table*)
= lr-data-seg-table-body (**t**,
get (*n*, *body*),
car (lr-data-seg-table-body-n (*n*,
body,
data-seg,
table)),
cdr (lr-data-seg-table-body-n (*n*,
body,
data-seg,
table))))

THEOREM: lr-data-seg-table-body-flag-t-flag-t
(definedp (*object*,
cdr (lr-data-seg-table-body (**t**,
get (*n*, *body*),
car (lr-data-seg-table-body-n (*n*,
body,
data-seg,
table)),
cdr (lr-data-seg-table-body-n (*n*,
body,
data-seg,
table)))))
$\land$ ($n \in \mathbf{N}$)
$\land$ ($n <$ length (*body*)))
$\rightarrow$ definedp (*object*,
cdr (lr-data-seg-table-body (`'list`, *body*, *data-seg*, *table*)))

THEOREM: definedp-cadr-cur-expr-quote-lr-data-seg-table-body
(listp (cur-expr (*pos*, *body*))
$\land$ (car (cur-expr (*pos*, *body*)) = `'quote`)
$\land$ good-posp1 (*pos*, *body*))
$\rightarrow$ definedp (cadr (cur-expr (*pos*, *body*)),
cdr (lr-data-seg-table-body (**t**, *body*, *data-seg*, *table*)))

THEOREM: definedp-cadr-cur-expr-quote-lr-data-seg-table-list

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog)))} \\
\land & (\text{car}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog))) = \texttt{'quote}) \\
\land & (prog \in progs) \\
\land & \text{good-posp1}\,(pos,\ \text{s-body}\,(prog))) \\
\rightarrow & \text{definedp}\,(\text{cadr}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog))), \\
& \qquad\qquad \text{cdr}\,(\text{lr-data-seg-table-list}\,(progs,\ data\text{-}seg,\ table)))
\end{array}
$$

THEOREM: definedp-cadr-cur-expr-quote-lr-data-seg-table

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog)))} \\
\land & (\text{car}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog))) = \texttt{'quote}) \\
\land & (prog \in progs) \\
\land & \text{good-posp1}\,(pos,\ \text{s-body}\,(prog))) \\
\rightarrow & \text{definedp}\,(\text{cadr}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog))), \\
& \qquad\qquad \text{cdr}\,(\text{lr-data-seg-table}\,(progs,\ params,\ heap\text{-}size)))
\end{array}
$$

THEOREM: lr-proper-exprp-p-lr-compile-programs-helper-2

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog)))} \\
\land & (\text{car}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog))) = \texttt{'quote}) \\
\land & (prog \in progs) \\
\land & \text{good-posp1}\,(pos,\ \text{s-body}\,(prog)) \\
\land & (heap\text{-}size \not< \text{s-total-heap-reqs}\,(progs,\ params,\ heap\text{-}size)) \\
\land & \text{s-restrictedp}\,(progs,\ params)) \\
\rightarrow & (\text{type}\,(\text{cdr}\,(\text{assoc}\,(\text{cadr}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog))), \\
& \qquad\qquad\qquad\qquad \text{cdr}\,(\text{lr-data-seg-table}\,(progs,\ params,\ heap\text{-}size))))) \\
& \quad = \ \texttt{'addr})
\end{array}
$$

THEOREM: good-posp-dv-1-funcall-opened

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog)))} \\
\land & (\text{car}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog))) \neq \texttt{'if}) \\
\land & (\text{car}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog))) \neq \text{S-TEMP-EVAL}) \\
\land & (\text{car}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog))) \neq \text{S-TEMP-TEST}) \\
\land & (\text{car}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog))) \neq \text{S-TEMP-FETCH}) \\
\land & (\text{car}\,(\text{cur-expr}\,(pos,\ \text{s-body}\,(prog))) \neq \texttt{'quote}) \\
\land & \text{good-posp1}\,(pos,\ \text{s-body}\,(prog))) \\
\rightarrow & \text{good-posp}\,(\texttt{'list},\ \text{dv}\,(pos,\ \texttt{1}),\ \text{s-body}\,(prog))
\end{array}
$$

THEOREM: s-restrict-subrps-list-lr-proper-get-t

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{s-restrict-subrps}\,(\texttt{'list},\ \text{cdr}\,(expr))} \\
\land & \text{listp}\,(expr) \\
\land & (n \not\simeq \texttt{0}) \\
\land & (n < \text{length}\,(expr))) \\
\rightarrow & \text{s-restrict-subrps}\,(\mathbf{t},\ \text{get}\,(n,\ expr))
\end{array}
$$

THEOREM: s-restrict-subrps-t-lr-proper-get-t

((car (*body*) $\neq$ S-TEMP-FETCH)
 $\wedge$ (car (*body*) $\neq$ S-TEMP-EVAL)
 $\wedge$ (car (*body*) $\neq$ S-TEMP-TEST)
 $\wedge$ (car (*body*) $\neq$ `'quote`)
 $\wedge$ listp (*body*)
 $\wedge$ s-proper-exprp (**t**, *body*, *program-names*, *formals*, *temp-list*)
 $\wedge$ ($n \not\simeq$ `0`)
 $\wedge$ ($n <$ length (*body*))
 $\wedge$ s-restrict-subrps (**t**, *body*))
$\rightarrow$ s-restrict-subrps (**t**, get (n, *body*))

EVENT: Disable s-restrict-subrps-list-lr-proper-get-t.

THEOREM: s-restrict-subrps-s-restrict-subrps-cur-expr
(s-restrict-subrps (**t**, *body*)
 $\wedge$ s-proper-exprp (**t**, *body*, *program-names*, *formals*, *temp-list*)
 $\wedge$ good-posp1 (*pos*, *body*))
$\rightarrow$ s-restrict-subrps (**t**, cur-expr (*pos*, *body*))

THEOREM: lr-proper-exprp-flag-not-list-cons-funcall
((*flag* $\neq$ `'list`)
 $\wedge$ listp (cur-expr (*pos*, s-body (*prog*)))
 $\wedge$ good-posp1 (*pos*, s-body (*prog*))
 $\wedge$ (car (cur-expr (*pos*, s-body (*prog*))) $\neq$ `'if`)
 $\wedge$ (car (cur-expr (*pos*, s-body (*prog*))) $\neq$ S-TEMP-FETCH)
 $\wedge$ (car (cur-expr (*pos*, s-body (*prog*))) $\neq$ S-TEMP-EVAL)
 $\wedge$ (car (cur-expr (*pos*, s-body (*prog*))) $\neq$ S-TEMP-TEST)
 $\wedge$ (car (cur-expr (*pos*, s-body (*prog*))) $\neq$ `'quote`)
 $\wedge$ lr-proper-exprp (`'list`,
 lr-compile-body (`'list`,
 cdr (cur-expr (*pos*, s-body (*prog*))),
 temp-alist,
 table),
 program-names,
 formals,
 temps,
 table)
 $\wedge$ s-restrict-subrps (**t**, s-body (*prog*))
 $\wedge$ s-proper-exprp (**t**, s-body (*prog*), *program-names*, *formals*, *temp-list*))
$\rightarrow$ lr-proper-exprp (*flag*,
 cons (car (cur-expr (*pos*, s-body (*prog*))),
 lr-compile-body (`'list`,
 cdr (cur-expr (*pos*, s-body (*prog*))),
 temp-alist,

$table$)),
$program\text{-}names$,
$formals$,
$temps$,
$table$)

THEOREM: lr-proper-exprp-flag-not-list-not-listp

$$
\begin{array}{ll}
((\mathit{flag} \neq \texttt{'list}) & \\
\wedge & (\neg\ \text{listp}(\text{cur-expr}(\mathit{pos},\ \mathit{body}))) \\
\wedge & \text{good-posp1}(\mathit{pos},\ \mathit{body}) \\
\wedge & \text{s-proper-exprp}(\mathbf{t},\ \mathit{body},\ \mathit{program\text{-}names},\ \mathit{formals},\ \mathit{temp\text{-}list})) \\
\rightarrow & \text{lr-proper-exprp}(\mathit{flag}, \\
& \quad \text{cur-expr}(\mathit{pos},\ \mathit{body}), \\
& \quad \mathit{program\text{-}names}, \\
& \quad \mathit{formals}, \\
& \quad \mathit{temps}, \\
& \quad \mathit{table})
\end{array}
$$

THEOREM: lr-proper-exprp-p-lr-compile-programs

$$
\begin{array}{ll}
(\text{s-restrict-subrps}(\mathbf{t},\ \text{s-body}(\mathit{prog})) & \\
\wedge & (\mathit{prog} \in \mathit{progs}) \\
\wedge & \text{subsetp}(\mathit{progs},\ \mathit{all\text{-}progs}) \\
\wedge & \text{s-proper-exprp}(\mathbf{t}, \\
& \quad \text{s-body}(\mathit{prog}), \\
& \quad \mathit{program\text{-}names}, \\
& \quad \mathit{formals}, \\
& \quad \text{strip-cars}(\mathit{temp\text{-}alist})) \\
\wedge & \text{good-posp}(\mathit{flag},\ \mathit{pos},\ \text{s-body}(\mathit{prog})) \\
\wedge & (\mathit{heap\text{-}size} \not< \text{s-total-heap-reqs}(\mathit{all\text{-}progs},\ \mathit{params},\ \mathit{heap\text{-}size})) \\
\wedge & \text{s-restrictedp}(\mathit{all\text{-}progs},\ \mathit{params})) \\
\rightarrow & \text{lr-proper-exprp}(\mathit{flag}, \\
& \quad \text{lr-compile-body}(\mathit{flag}, \\
& \qquad \textbf{if } \mathit{flag} = \texttt{'list} \\
& \qquad \textbf{then } \text{restn}(\text{car}(\text{last}(\mathit{pos})), \\
& \qquad\qquad \text{cur-expr}(\text{butlast}(\mathit{pos}), \\
& \qquad\qquad\quad \text{s-body}(\mathit{prog}))) \\
& \qquad \textbf{else } \text{cur-expr}(\mathit{pos},\ \text{s-body}(\mathit{prog}))\ \textbf{endif}, \\
& \qquad \mathit{temp\text{-}alist}, \\
& \qquad \text{cdr}(\text{lr-data-seg-table}(\mathit{all\text{-}progs}, \\
& \qquad\qquad \mathit{params}, \\
& \qquad\qquad \mathit{heap\text{-}size}))), \\
& \quad \mathit{program\text{-}names}, \\
& \quad \mathit{formals}, \\
& \quad \text{strip-cdrs}(\mathit{temp\text{-}alist}), \\
& \quad \text{cdr}(\text{lr-data-seg-table}(\mathit{all\text{-}progs},\ \mathit{params},\ \mathit{heap\text{-}size})))
\end{array}
$$

EVENT: Disable lr-proper-exprp-p-lr-compile-programs-helper-1.

THEOREM: all-litatoms-s-formals-member-s-programs-properp
$((prog \in progs) \land$ s-programs-properp $(progs,\ program\text{-}names))$
$\rightarrow$ all-litatoms (s-formals $(prog)$)

THEOREM: s-restrict-subrps-s-body-member-s-restrict-subrps-progs
(s-restrict-subrps-progs $(progs) \land (prog \in progs)$)
$\rightarrow$ s-restrict-subrps ($\mathbf{t}$, s-body $(prog)$)

THEOREM: lr-proper-exprp-p-lr-compile-programs-flag-t
(s-restrict-subrps-progs $(all\text{-}progs)$
$\land$ $(prog \in all\text{-}progs)$
$\land$ s-programs-properp $(all\text{-}progs,\ program\text{-}names)$
$\land$ $(heap\text{-}size \not< $ s-total-heap-reqs $(all\text{-}progs,\ params,\ heap\text{-}size))$
$\land$ s-restrictedp $(all\text{-}progs,\ params)$
$\land$ $(temp\text{-}alist =$ lr-make-temp-name-alist (s-temp-list $(prog)$,
s-formals $(prog)$)))
$\rightarrow$ lr-proper-exprp ($\mathbf{t}$,
lr-compile-body ($\mathbf{t}$,
s-body $(prog)$,
$temp\text{-}alist$,
cdr (lr-data-seg-table $(all\text{-}progs$,
$params$,
$heap\text{-}size$))),
$program\text{-}names$,
s-formals $(prog)$,
strip-cdrs $(temp\text{-}alist)$,
cdr (lr-data-seg-table $(all\text{-}progs,\ params,\ heap\text{-}size)$))

THEOREM: lr-programs-properp-1-lr-compile-programs
(s-programs-properp $(all\text{-}progs,\ program\text{-}names)$
$\land$ s-programs-okp $(progs)$
$\land$ s-restrict-subrps-progs $(all\text{-}progs)$
$\land$ subsetp $(progs,\ all\text{-}progs)$
$\land$ $(heap\text{-}size \not< $ s-total-heap-reqs $(all\text{-}progs,\ params,\ heap\text{-}size))$
$\land$ s-restrictedp $(all\text{-}progs,\ params)$)
$\rightarrow$ lr-programs-properp-1 (lr-compile-programs $(progs$,
cdr (lr-data-seg-table $(all\text{-}progs$,
$params$,
$heap\text{-}size$))),
$program\text{-}names$,
cdr (lr-data-seg-table $(all\text{-}progs$,
$params$,
$heap\text{-}size$)))

EVENT: Disable all-litatoms-s-formals-member-s-programs-properp.

THEOREM: subsetp-cdr
subsetp (cdr (x), x)

THEOREM: lr-programs-properp-s->lr-opened
(s-good-statep (s, c)
 ∧ s-restrict-subrps-progs (s-progs (s))
 ∧ ($heap\text{-}size \not< $ s-total-heap-reqs (s-progs (s), s-params (s), $heap\text{-}size$))
 ∧ s-restrictedp (s-progs (s), s-params (s))
 ∧ litatom (name (car (s-progs (s)))))
 → lr-programs-properp (s->lr1 (s,
        p-state (pc,
            $ctrl\text{-}stk$,
            $temp\text{-}stk$,
            $anything$,
            $data\text{-}seg$,
            $max\text{-}ctrl$,
            $max\text{-}temp$,
            $word\text{-}size$,
            nil),
        cdr (lr-data-seg-table (s-progs (s),
            s-params (s),
            $heap\text{-}size$))),
    cdr (lr-data-seg-table (s-progs (s),
        s-params (s),
        $heap\text{-}size$)))

THEOREM: fall-off-proofp-append-cons-ret
fall-off-proofp (append ($instrs$, list (list (`'dl`, $label$, $comment$, `'(ret)`))))

THEOREM: proper-labeled-p-instructionsp-label-ret
litatom ($label$)
 → proper-labeled-p-instructionsp (list (list (`'dl`, $label$, $comment$, `'(ret)`)),
    $name$,
    p)

THEOREM: member-definedp-car
($x \in y$) → definedp (car (x), y)

THEOREM: all-litatoms-s-formals-member-lr-programs-properp
(($prog \in prog\text{-}seg$) ∧ (¬ all-litatoms (formal-vars ($prog$))))
 → (¬ lr-programs-properp-1 ($prog\text{-}seg$, $program\text{-}names$, $table$))

THEOREM: properp-p-temp-var-dclps-member-lr-programs-properp
$((prog \in progs)$
$\land$ lr-proper-heapp $(data\text{-}seg)$
$\land$ lr-proper-p-areasp $(data\text{-}seg)$
$\land$ lr-programs-properp-1 $(progs,$ strip-logic-fnames (cdr $(prog\text{-}seg)),$ $table))$
$\rightarrow$ proper-p-temp-var-dclsp (temp-var-dcls $(prog)$,
p-state $(pc,$
$ctrl\text{-}stk,$
$temp\text{-}stk,$
comp-programs $(prog\text{-}seg)$,
$data\text{-}seg,$
$max\text{-}ctrl\text{-}stk\text{-}size,$
$max\text{-}temp\text{-}stk\text{-}size,$
$word\text{-}size,$
$psw))$

THEOREM: member-f-definedp-0
$(\mathbf{f} \in alist) \rightarrow$ definedp (0, $alist$)

THEOREM: member-no-duplicatesp-assoc-equal
$((x \in alist) \land$ no-duplicatesp (strip-cars $(alist)) \land$ good-alistp $(alist))$
$\rightarrow$ (assoc (car (x), $alist) = x)$

EVENT: Disable member-f-definedp-0.

THEOREM: lr-proper-exprp-program-body-not-listp
$(\neg$ listp $(prog))$
$\rightarrow$ $(\neg$ lr-proper-exprp $(\mathbf{t}$, program-body $(prog)$, $pnames,$ $formals,$ $temps,$ $table))$

THEOREM: good-alistp-lr-programs-properp
$(\neg$ good-alistp $(prog\text{-}seg))$
$\rightarrow$ $(\neg$ lr-programs-properp-1 $(prog\text{-}seg,$ $program\text{-}names,$ $table))$

EVENT: Disable lr-proper-exprp-program-body-not-listp.

THEOREM: proper-p-prog-segmentp-comp-programs-1-helper
(lr-proper-heapp $(data\text{-}seg)$
$\land$ lr-proper-p-areasp $(data\text{-}seg)$
$\land$ lr-programs-properp-1 $(prog\text{-}seg,$
strip-logic-fnames (cdr $(prog\text{-}seg))$,
$table)$
$\land$ all-user-fnamesp (cdr (strip-cars $(prog\text{-}seg))$)
$\land$ lr-s-similar-const-table $(table,$ $data\text{-}seg)$
$\land$ $(prog \in prog\text{-}seg)$

$\land$ no-duplicatesp (strip-cars (*prog-seg*)))
$\rightarrow$ proper-labeled-p-instructionsp (comp-body-1 (**t**, program-body (*prog*), 0),
car (*prog*),
p-state (*pc*,
ctrl-stk,
temp-stk,
comp-programs (*prog-seg*),
data-seg,
max-ctrl,
max-temp,
word-size,
psw))

THEOREM: proper-p-prog-segmentp-comp-programs-1
(lr-proper-heapp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ lr-programs-properp-1 (*prog-seg*,
strip-logic-fnames (cdr (*prog-seg*)),
table)
$\land$ all-user-fnamesp (cdr (strip-cars (*prog-seg*)))
$\land$ lr-s-similar-const-table (*table*, *data-seg*)
$\land$ no-duplicatesp (strip-cars (*prog-seg*))
$\land$ subsetp (*progs*, *prog-seg*)
$\land$ all-litatoms (strip-cars (*progs*)))
$\rightarrow$ proper-p-prog-segmentp (comp-programs-1 (*progs*),
p-state (*pc*,
ctrl-stk,
temp-stk,
comp-programs (*prog-seg*),
data-seg,
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))

EVENT: Disable all-litatoms-s-formals-member-lr-programs-properp.

THEOREM: proper-p-instructionp-set-global
adpp (cons (*name*, 0), p-data-segment (l))
$\rightarrow$ proper-p-instructionp (list (`'set-global`, *name*), x, lr->p (l))

THEOREM: proper-p-programp-append-car-prog-segment
(lr-programs-properp-1 (*prog-seg*, strip-logic-fnames (cdr (*prog-seg*)), *table*)
$\land$ definedp (area-name (p-pc (l)), p-prog-segment (l))

$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-proper-p-areasp (p-data-segment (l))
$\wedge$ lr-s-similar-const-table (*table*, p-data-segment (l))
$\wedge$ adpp (untag (LR-ANSWER-ADDR), p-data-segment (l))
$\wedge$ (*prog-seg* = p-prog-segment (l))
$\wedge$ all-litatoms (strip-cars (*prog-seg*))
$\wedge$ all-user-fnamesp (strip-cars (cdr (p-prog-segment (l)))))
$\rightarrow$ proper-p-programp (cons (name (car (*prog-seg*)),
cons (formal-vars (car (*prog-seg*)),
cons (temp-var-dcls (car (*prog-seg*)),
append (label-instrs (comp-body-1 (**t**,
program-body (car (*prog-seg*)),
`0`),
`0`),
list (list (**'dl**,
lr-make-label (*n1*),
coment1,
list (**'set-global**,
identity (area-name (LR-ANSWER-ADDR)))),
list (**'dl**,
lr-make-label (*n2*),
comment2,
'(ret))))))),
lr->p (l))

THEOREM: all-litatoms-all-user-fnamesp-plistp
(all-user-fnamesp (*list*) $\wedge$ plistp (*list*)) $\rightarrow$ all-litatoms (*list*)

THEOREM: plistp-strip-cars
plistp (strip-cars (x))

THEOREM: lr-programs-properp-all-user-fnamesp-strip-cars-cdr
lr-programs-properp (l, *table*)
$\rightarrow$ all-user-fnamesp (strip-cars (cdr (p-prog-segment (l))))

THEOREM: lr-programs-properp-caar-main
lr-programs-properp (l, *table*) $\rightarrow$ (caar (p-prog-segment (l)) = **'main**)

THEOREM: proper-p-prog-segmentp-p-runtime-support-programs
(lr-programs-properp (l, *table*)
$\wedge$ (p-word-size (l) $\in \mathbf{N}$)
$\wedge$ (p-word-size (l) $\not<$ S-MAX-SUBR-REQS)
$\wedge$ lr-proper-heapp (p-data-segment (l))
$\wedge$ lr-proper-p-areasp (p-data-segment (l)))
$\rightarrow$ proper-p-prog-segmentp (P-RUNTIME-SUPPORT-PROGRAMS, lr->p (l))

THEOREM: proper-p-prog-segmentp-comp-programs
(lr-proper-heapp (p-data-segment (l))
 $\wedge$ lr-proper-p-areasp (p-data-segment (l))
 $\wedge$ ($progs$ = p-prog-segment (l))
 $\wedge$ lr-programs-properp (l, $table$)
 $\wedge$ lr-s-similar-const-table ($table$, p-data-segment (l))
 $\wedge$ no-duplicatesp (strip-cars (p-prog-segment (l)))
 $\wedge$ (p-word-size (l) $\in \mathbf{N}$)
 $\wedge$ (p-word-size (l) $\not<$ S-MAX-SUBR-REQS)
 $\wedge$ adpp (untag (LR-ANSWER-ADDR), p-data-segment (l)))
$\rightarrow$ proper-p-prog-segmentp (comp-programs ($progs$), lr->p (l))

EVENT: Disable lr-programs-properp-caar-main.

THEOREM: no-duplicatesp-remove-duplicates
no-duplicatesp (remove-duplicates (x))

THEOREM: all-p-objectps-put
(all-p-objectps (lst, p) $\wedge$ p-objectp ($value$, p) $\wedge$ ($offset$ < length (lst)))
$\rightarrow$ all-p-objectps (put ($value$, $offset$, lst), p)

THEOREM: proper-p-data-segmentp-deposit-helper
(proper-p-data-segmentp ($data\text{-}seg$, p)
 $\wedge$ (offset ($addr$) < length (cdr (assoc (area-name ($addr$), $data\text{-}seg$))))
 $\wedge$ p-objectp ($value$, p))
$\rightarrow$ proper-p-data-segmentp (deposit ($value$, $addr$, $data\text{-}seg$), p)

THEOREM: proper-p-data-segmentp-deposit
(proper-p-data-segmentp ($data\text{-}seg$, p)
 $\wedge$ adpp (untag ($addr$), $data\text{-}seg$)
 $\wedge$ p-objectp ($value$, p))
$\rightarrow$ proper-p-data-segmentp (deposit ($value$, $addr$, $data\text{-}seg$), p)

THEOREM: all-p-objectps-get
(all-p-objectps (lst, p)
 $\wedge$ ($offset$ < length (lst))
 $\wedge$ all-p-objectps (lst, p))
$\rightarrow$ p-objectp (get ($offset$, lst), p)

THEOREM: proper-p-data-segmentp-fetch
(proper-p-data-segmentp ($data\text{-}seg$, p)
 $\wedge$ (offset ($addr$) < length (cdr (assoc (area-name ($addr$), $data\text{-}seg$))))
 $\wedge$ definedp (area-name ($addr$), $data\text{-}seg$))
$\rightarrow$ p-objectp (fetch ($addr$, $data\text{-}seg$), p)

THEOREM: lr-s-similar-const-table-p-objectp-definedp

$$
\begin{array}{l}
(\text{lr-s-similar-const-table}\,(\mathit{table},\ \mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{lr-proper-heapp}\,(\mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{lr-proper-p-areasp}\,(\mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{definedp}\,(\mathbf{f},\ \mathit{table}) \\
\ \wedge\ \ (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \mathit{data\text{-}seg}), \\
\qquad\qquad\qquad\qquad \text{lr-free-list-nodes}\,(\text{lr-max-node}\,(\mathit{data\text{-}seg}), \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{data\text{-}seg}), \\
\qquad\qquad\qquad\qquad \mathit{data\text{-}seg}) \\
\qquad \not< \ \ \text{s-heap-reqs}\,(\mathit{flag},\ \mathit{object2},\ \mathit{data\text{-}seg},\ \mathit{table})) \\
\ \wedge\ \ \text{definedp}\,(\mathit{object1},\ \text{cdr}\,(\text{lr-compile-quote}\,(\mathit{flag},\ \mathit{object2},\ \mathit{data\text{-}seg},\ \mathit{table}))) \\
\ \wedge\ \ \text{same-signature}\,(\text{p-data-segment}\,(p),\ \mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{s-restricted-objectp}\,(\mathit{flag},\ \mathit{object2})) \\
\rightarrow\ \ \text{p-objectp}\,(\text{cdr}\,(\text{assoc}\,(\mathit{object1}, \\
\qquad\qquad\qquad\qquad \text{cdr}\,(\text{lr-compile-quote}\,(\mathit{flag},\ \mathit{object2},\ \mathit{data\text{-}seg},\ \mathit{table})))), \\
\qquad\qquad\quad p)
\end{array}
$$

THEOREM: lessp-x-sub1-facts

$$(x < ((y - 1) - 1)) \rightarrow ((x < y) \wedge (x < (y - 1)))$$

THEOREM: proper-p-data-segmentp-deposit-a-list-cons

$$
\begin{array}{l}
((\text{area-name}\,(\mathit{addr}) = \text{LR-HEAP-NAME}) \\
\ \wedge\ \ \text{adpp}\,(\text{untag}\,(\mathit{addr}),\ \mathit{data\text{-}seg}) \\
\ \wedge\ \ ((\text{offset}\,(\mathit{addr}) + 3) < \text{length}\,(\text{cdr}\,(\text{assoc}\,(\text{LR-HEAP-NAME},\ \mathit{data\text{-}seg})))) \\
\ \wedge\ \ \text{p-objectp}\,(a,\ p) \\
\ \wedge\ \ \text{p-objectp}\,(b,\ p) \\
\ \wedge\ \ \text{p-objectp}\,(c,\ p) \\
\ \wedge\ \ \text{p-objectp}\,(d,\ p) \\
\ \wedge\ \ \text{proper-p-data-segmentp}\,(\mathit{data\text{-}seg},\ p)) \\
\rightarrow\ \ \text{proper-p-data-segmentp}\,(\text{deposit-a-list}\,(\text{list}\,(a,\ b,\ c,\ d),\ \mathit{addr},\ \mathit{data\text{-}seg}),\ p)
\end{array}
$$

THEOREM: proper-p-data-segmentp-deposit-a-list-cons-cons

$$
\begin{array}{l}
\textbf{let}\ \mathit{ds\text{-}tab}\ \ \textbf{be}\ \text{lr-compile-quote}\,(\texttt{'list},\ \text{list}\,(x,\ y),\ \mathit{data\text{-}seg},\ \mathit{table}), \\
\qquad \mathit{count\text{-}nodes}\ \ \textbf{be}\ \text{lr-count-free-nodes}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \mathit{data\text{-}seg}), \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{lr-free-list-nodes}\,(\text{lr-max-node}\,(\mathit{data\text{-}seg}), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{data\text{-}seg}), \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{data\text{-}seg}) \\
\textbf{in} \\
(\text{lr-proper-p-areasp}\,(\mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{lr-proper-heapp}\,(\mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{lr-s-similar-const-table}\,(\mathit{table},\ \mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{definedp}\,(\mathbf{f},\ \mathit{table}) \\
\ \wedge\ \ ((\mathit{count\text{-}nodes} - 1) \\
\qquad \not< \ \ \text{s-heap-reqs}\,(\texttt{'list},\ \text{list}\,(x,\ y),\ \mathit{data\text{-}seg},\ \mathit{table})) \\
\ \wedge\ \ (\mathit{count\text{-}nodes} \neq \texttt{0})
\end{array}
$$

$\wedge$ proper-p-data-segmentp (car (*ds-tab*), *p*)
$\wedge$ ((length (cdr (assoc (LR-HEAP-NAME, *data-seg*))) $-$ 1)
  $\not<$ (offset (fetch (LR-FP-ADDR, car (*ds-tab*)))
    $+$ LR-NODE-SIZE))
$\wedge$ (p-word-size (p) $\in \mathbf{N}$)
$\wedge$ (p-word-size (p) $\not<$ S-MAX-SUBR-REQS)
$\wedge$ same-signature (p-data-segment (p), *data-seg*)
$\wedge$ s-restricted-objectp (`'list`, list (x, y)))
$\rightarrow$ proper-p-data-segmentp (deposit-a-list (list (identity (tag (`'nat`,
    LR-CONS-TAG)),
  identity (tag (`'nat`,
    `1`)),
  cdr (assoc (x,
    cdr (*ds-tab*))),
  cdr (assoc (y,
    cdr (*ds-tab*)))),
  fetch (identity (LR-FP-ADDR),
    car (*ds-tab*)),
  car (*ds-tab*)),
p) **endlet**

THEOREM: proper-p-data-segmentp-deposit-a-list-cons-numberp
(lr-proper-p-areasp (*data-seg*)
$\wedge$ lr-proper-heapp (*data-seg*)
$\wedge$ (p-word-size (p) $\not<$ log (`2`, *number*))
$\wedge$ (*number* $\in \mathbf{N}$)
$\wedge$ proper-p-data-segmentp (*data-seg*, p)
$\wedge$ ((length (cdr (assoc (LR-HEAP-NAME, *data-seg*))) $-$ 1)
  $\not<$ (offset (fetch (LR-FP-ADDR, *data-seg*)) $+$ LR-NODE-SIZE))
$\wedge$ (p-word-size (p) $\in \mathbf{N}$)
$\wedge$ (p-word-size (p) $\not<$ S-MAX-SUBR-REQS)
$\wedge$ same-signature (p-data-segment (p), *data-seg*))
$\rightarrow$ proper-p-data-segmentp (deposit-a-list (list (identity (tag (`'nat`,
    LR-ADD1-TAG)),
  identity (tag (`'nat`, `1`)),
  tag (`'nat`, *number*),
  identity (LR-UNDEF-ADDR)),
  fetch (identity (LR-FP-ADDR),
    data-seg),
  data-seg),
p)

THEOREM: proper-p-data-segmentp-deposit-a-list-cons-truep
(lr-proper-p-areasp (*data-seg*)

```
 ∧   lr-proper-heapp (data-seg)
 ∧   proper-p-data-segmentp (data-seg, p)
 ∧   ((length (cdr (assoc (LR-HEAP-NAME, data-seg))) − 1)
       ≮   (offset (fetch (LR-FP-ADDR, data-seg)) + LR-NODE-SIZE))
 ∧   (p-word-size (p) ∈ N)
 ∧   (p-word-size (p) ≮ log (2, LR-TRUE-TAG))
 ∧   same-signature (p-data-segment (p), data-seg))
 →   proper-p-data-segmentp (deposit-a-list (list (identity (tag ('nat,
                                                                LR-TRUE-TAG)),
                                                   identity (tag ('nat, 1)),
                                                   identity (LR-UNDEF-ADDR),
                                                   identity (LR-UNDEF-ADDR)),
                                              fetch (identity (LR-FP-ADDR),
                                                     data-seg),
                                              data-seg),
                             p)
```

EVENT: Disable proper-p-data-segmentp-deposit-a-list-cons.

THEOREM: s-ws-reqs-flag-not-list-t

```
((flag ≠ 'list) ∧ (¬ definedp (t, table)))
 →   (s-ws-reqs (flag, t, data-seg, table) = identity (log (2, LR-TRUE-TAG)))
```

THEOREM: lr-compile-quote-preserves-proper-p-data-segmentp

```
(lr-s-similar-const-table (table, data-seg)
 ∧   s-restricted-objectp (flag, object)
 ∧   lr-proper-heapp (data-seg)
 ∧   lr-proper-p-areasp (data-seg)
 ∧   definedp (f, table)
 ∧   (p-word-size (p) ∈ N)
 ∧   (p-word-size (p) ≮ s-ws-reqs (flag, object, data-seg, table))
 ∧   (lr-count-free-nodes (fetch (LR-FP-ADDR, data-seg),
                          lr-free-list-nodes (lr-max-node (data-seg),
                                              data-seg),
                          data-seg)
       ≮   s-heap-reqs (flag, object, data-seg, table))
 ∧   proper-p-data-segmentp (data-seg, p)
 ∧   same-signature (p-data-segment (p), data-seg))
 →   proper-p-data-segmentp (car (lr-compile-quote (flag,
                                                    object,
                                                    data-seg,
                                                    table)),
                             p)
```

THEOREM: lr-data-seg-table-body-preserves-proper-p-data-segmentp
(lr-s-similar-const-table (*table*, *data-seg*)
$\land$ s-data-seg-body-restrictedp (*flag*, *body*)
$\land$ lr-proper-heapp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ proper-p-data-segmentp (*data-seg*, *p*)
$\land$ definedp (**f**, *table*)
$\land$ (p-word-size (*p*) $\in$ **N**)
$\land$ (p-word-size (*p*) $\not<$ s-ws-reqs-body (*flag*, *body*, *data-seg*, *table*))
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs-body (*flag*, *body*, *data-seg*, *table*))
$\land$ same-signature (p-data-segment (*p*), *data-seg*))
$\to$ proper-p-data-segmentp (car (lr-data-seg-table-body (*flag*,
body,
data-seg,
table)),
p)

THEOREM: lr-data-seg-table-list-preserves-proper-p-data-segmentp
(lr-s-similar-const-table (*table*, *data-seg*)
$\land$ s-data-seg-list-restrictedp (*progs*)
$\land$ lr-proper-heapp (*data-seg*)
$\land$ lr-proper-p-areasp (*data-seg*)
$\land$ proper-p-data-segmentp (*data-seg*, *p*)
$\land$ definedp (**f**, *table*)
$\land$ (p-word-size (*p*) $\in$ **N**)
$\land$ (p-word-size (*p*) $\not<$ s-ws-reqs-list (*progs*, *data-seg*, *table*))
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs-list (*progs*, *data-seg*, *table*))
$\land$ same-signature (p-data-segment (*p*), *data-seg*))
$\to$ proper-p-data-segmentp (car (lr-data-seg-table-list (*progs*,
data-seg,
table)),
p)

THEOREM: lr-init-data-seg-table-preserves-proper-p-data-segmentp
(lr-s-similar-const-table (*table*, *data-seg*)
$\land$ s-init-data-seg-restrictedp (*params*)

$\wedge$ lr-proper-heapp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ proper-p-data-segmentp (*data-seg*, *p*)
$\wedge$ definedp (**f**, *table*)
$\wedge$ (p-word-size (p) $\in \mathbf{N}$)
$\wedge$ (p-word-size (p) $\not<$ s-init-ws-reqs (*params*, *data-seg*, *table*))
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-init-heap-reqs (*params*, *data-seg*, *table*))
$\wedge$ same-signature (p-data-segment (p), *data-seg*))
$\rightarrow$ proper-p-data-segmentp (car (lr-init-data-seg-table (*params*,
data-seg,
table)),
p)

THEOREM: all-p-objectps-lr-init-heap-contents-helper-helper
(*heap-size* $\not\simeq$ `0`)
$\rightarrow$ ((identity (LR-NODE-SIZE)
$+$ x
$+$ (identity (LR-NODE-SIZE) $*$ (*heap-size* $-$ 1)))
$=$ (x $+$ (identity (LR-NODE-SIZE) $*$ *heap-size*)))

THEOREM: all-p-objectps-lr-init-heap-contents-helper
((p-word-size (p) $\in \mathbf{N}$)
$\wedge$ (p-word-size (p) $\not<$ S-MAX-SUBR-REQS)
$\wedge$ (*heap-size* $\not<$ `4`)
$\wedge$ lr-proper-p-areasp (p-data-segment (p))
$\wedge$ lr-minimum-heapp (p-data-segment (p))
$\wedge$ (type (*addr*) $=$ `'addr`)
$\wedge$ (cddr (*addr*) $=$ **nil**)
$\wedge$ listp (*addr*)
$\wedge$ (offset (*addr*) $\in \mathbf{N}$)
$\wedge$ lr-boundary-nodep (*addr*)
$\wedge$ (area-name (*addr*) $=$ LR-HEAP-NAME)
$\wedge$ definedp (LR-HEAP-NAME, p-data-segment (p))
$\wedge$ (((LR-NODE-SIZE $*$ *heap-size*) $+$ offset (*addr*))
$<$ length (cdr (assoc (LR-HEAP-NAME, p-data-segment (p))))))
$\rightarrow$ all-p-objectps (lr-init-heap-contents (*addr*, *heap-size*), p)

EVENT: Disable all-p-objectps-lr-init-heap-contents-helper-helper.

THEOREM: all-p-objectps-lr-init-heap-contents

```
((p-word-size (p) ∈ N)
 ∧   (p-word-size (p) ≮ S-MAX-SUBR-REQS)
 ∧   (heap-size ≮ 4)
 ∧   lr-proper-p-areasp (p-data-segment (p))
 ∧   same-signature (p-data-segment (p), lr-init-data-seg (heap-size)))
 →   all-p-objectps (lr-init-heap-contents (identity (tag ('addr,
                                                           cons (LR-HEAP-NAME,
                                                                 0))),
                                                   heap-size),
                     p)
```

THEOREM: proper-p-data-segmentp-lr-init-data-seg-helper

```
((p-word-size (p) ∈ N)
 ∧   (p-word-size (p) ≮ S-MAX-SUBR-REQS)
 ∧   (heap-size ≮ 4)
 ∧   lr-proper-p-areasp (p-data-segment (p))
 ∧   same-signature (p-data-segment (p), lr-init-data-seg (heap-size)))
 →   proper-p-data-segmentp (deposit-a-list (list (tag ('nat, LR-FALSE-TAG),
                                                   tag ('nat, 1),
                                                   LR-UNDEF-ADDR,
                                                   LR-UNDEF-ADDR),
                                              LR-F-ADDR,
                                              deposit-a-list (list (tag ('nat,
                                                                         LR-UNDEFINED-TAG),
                                                                    tag ('nat,
                                                                         1),
                                                                    LR-UNDEF-ADDR,
                                                                    LR-UNDEF-ADDR),
                                                              LR-UNDEF-ADDR,
                                                              list (list (area-name (LR-FP-ADDR),
                                                                          add-addr (LR-F-ADDR,
                                                                                    LR-NODE-SIZE)),
                                                                    list (area-name (LR-ANSWER-ADDR),
                                                                          tag ('nat,
                                                                               0)),
                                                                    cons (LR-HEAP-NAME,
                                                                          lr-init-heap-contents (tag ('addr,
                                                                                                      cons (LR-HEAP-
                                                                                                            0)),
                                                                                                 heap-size))))),
                             p)
```

THEOREM: proper-p-data-segmentp-lr-init-data-seg

```
((p-word-size (p) ∈ N)
```

$\wedge$ (p-word-size (p) $\not<$ S-MAX-SUBR-REQS)
$\wedge$ (*heap-size* $\not<$ `4`)
$\wedge$ lr-proper-p-areasp (p-data-segment (p))
$\wedge$ same-signature (p-data-segment (p), lr-init-data-seg (*heap-size*)))
$\rightarrow$ proper-p-data-segmentp (lr-init-data-seg (*heap-size*), p)

THEOREM: proper-p-data-segmentp-lr-init-data-seg-compile-t0
((p-word-size (p) $\in$ **N**)
$\wedge$ (p-word-size (p) $\not<$ S-MAX-SUBR-REQS)
$\wedge$ (*heap-size* $\not<$ `4`)
$\wedge$ lr-proper-p-areasp (p-data-segment (p))
$\wedge$ same-signature (p-data-segment (p), lr-init-data-seg (*heap-size*)))
$\rightarrow$ proper-p-data-segmentp (car (lr-compile-quote (**'list**,
list (**t**, `0`),
lr-init-data-seg (*heap-size*),
list (cons (**f**, *addr*)))),
p)

THEOREM: same-signature-car-lr-data-seg-table-list-reducer
(lr-proper-free-listp (*data-seg2*)
$\wedge$ lr-proper-p-areasp (*data-seg2*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg2*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg2*)))
$\rightarrow$ (same-signature (*data-seg1*,
car (lr-data-seg-table-list (*progs*, *data-seg2*, *table*)))
$\leftrightarrow$ same-signature (*data-seg1*, *data-seg2*))

THEOREM: same-signature-car-lr-init-data-seg-table-reducer
(lr-proper-free-listp (*data-seg2*)
$\wedge$ lr-proper-p-areasp (*data-seg2*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg2*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg2*)))
$\rightarrow$ (same-signature (*data-seg1*,
car (lr-init-data-seg-table (*params*, *data-seg2*, *table*)))
$\leftrightarrow$ same-signature (*data-seg1*, *data-seg2*))

THEOREM: same-signature-car-lr-compile-quote-reducer
(lr-proper-free-listp (*data-seg2*)
$\wedge$ lr-proper-p-areasp (*data-seg2*)
$\wedge$ definedp (LR-HEAP-NAME, *data-seg2*)
$\wedge$ lr-boundary-nodep (lr-max-node (*data-seg2*)))
$\rightarrow$ (same-signature (*data-seg1*,
car (lr-compile-quote (*flag*, *object*, *data-seg2*, *table*)))
$\leftrightarrow$ same-signature (*data-seg1*, *data-seg2*))

THEOREM: proper-p-data-segmentp-lr-data-seg-table

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{p-word-size}\,(p) \in \mathbf{N})} \\
\wedge & (\text{p-word-size}\,(p) \not< \text{S-MAX-SUBR-REQS}) \\
\wedge & \text{lr-proper-p-areasp}\,(\text{p-data-segment}\,(p)) \\
\wedge & \text{s-restrictedp}\,(\textit{progs},\ \textit{params}) \\
\wedge & (\textit{heap-size} \not< \text{s-total-heap-reqs}\,(\textit{progs},\ \textit{params},\ \textit{heap-size})) \\
\wedge & (\text{p-word-size}\,(p) \not< \text{s-total-ws-reqs}\,(\textit{progs},\ \textit{params},\ \textit{heap-size})) \\
\wedge & \text{same-signature}\,(\text{p-data-segment}\,(p), \\
 & \qquad \text{car}\,(\text{lr-data-seg-table}\,(\textit{progs},\ \textit{params},\ \textit{heap-size})))) \\
\rightarrow & \text{proper-p-data-segmentp}\,(\text{car}\,(\text{lr-data-seg-table}\,(\textit{progs},\ \textit{params},\ \textit{heap-size})), \\
 & \qquad p)
\end{array}
$$

THEOREM: adpp-untag-answer-addr-car-lr-data-seg-table

$$
\begin{array}{l}
\text{adpp}\,(\text{identity}\,(\text{untag}\,(\text{LR-ANSWER-ADDR})), \\
\qquad \text{car}\,(\text{lr-data-seg-table}\,(\textit{progs},\ \textit{params},\ \textit{heap-size})))
\end{array}
$$

THEOREM: program-body-assoc-cdr-lr-compile-programs

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\textit{name} \neq \text{caar}\,(\textit{progs}))} \\
\rightarrow & (\text{program-body}\,(\text{assoc}\,(\textit{name},\ \text{cdr}\,(\text{lr-compile-programs}\,(\textit{progs},\ \textit{table})))) \\
 & \quad = \ \text{lr-compile-body}\,(\mathbf{t}, \\
 & \qquad\qquad \text{s-body}\,(\text{assoc}\,(\textit{name},\ \text{cdr}\,(\textit{progs}))), \\
 & \qquad\qquad \text{lr-make-temp-name-alist}\,(\text{s-temp-list}\,(\text{assoc}\,(\textit{name}, \\
 & \qquad\qquad\qquad \text{cdr}\,(\textit{progs}))), \\
 & \qquad\qquad\qquad \text{s-formals}\,(\text{assoc}\,(\textit{name}, \\
 & \qquad\qquad\qquad\qquad \text{cdr}\,(\textit{progs})))), \\
 & \qquad\qquad \textit{table}))
\end{array}
$$

THEOREM: s-total-ws-reqs-not-lessp-s-max-subr-reqs

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\textit{word-size} \not< \text{s-total-ws-reqs}\,(\textit{progs},\ \textit{params},\ \textit{heap-size}))} \\
\rightarrow & (\textit{word-size} \not< \text{S-MAX-SUBR-REQS})
\end{array}
$$

THEOREM: proper-p-statep-lr->p-s->lr

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{s-good-statep}\,(s,\ c)} \\
\wedge & \text{all-litatoms}\,(\text{strip-cars}\,(\text{s-params}\,(s))) \\
\wedge & (\textit{max-ctrl} \not< (2 + \text{length}\,(\text{s-params}\,(s)) + \text{length}\,(\text{s-temps}\,(s)))) \\
\wedge & (\textit{word-size} \in \mathbf{N}) \\
\wedge & (\textit{max-temp} < \exp\,(2,\ \textit{word-size})) \\
\wedge & (\textit{max-ctrl} < \exp\,(2,\ \textit{word-size})) \\
\wedge & (\textit{max-temp} \in \mathbf{N}) \\
\wedge & (\textit{max-ctrl} \in \mathbf{N}) \\
\wedge & (\textit{heap-size} \not< \text{s-total-heap-reqs}\,(\text{s-progs}\,(s),\ \text{s-params}\,(s),\ \textit{heap-size})) \\
\wedge & \text{s-restrict-subrps-progs}\,(\text{s-progs}\,(s)) \\
\wedge & \text{litatom}\,(\text{name}\,(\text{car}\,(\text{s-progs}\,(s)))) \\
\wedge & \text{no-duplicatesp}\,(\text{strip-cars}\,(\text{s-progs}\,(s))) \\
\wedge & (\text{strip-cars}\,(\text{s-params}\,(s)) = \text{s-formals}\,(\text{s-prog}\,(s)))
\end{array}
$$

$\land$ s-restrictedp (s-progs (s), s-params (s))
$\land$ (*word-size* $\not<$ s-total-ws-reqs (s-progs (s), s-params (s), *heap-size*)))
$\rightarrow$ proper-p-statep (lr->p (s->lr (s, *heap-size*, *max-ctrl*, *max-temp*, *word-size*)))

THEOREM: lr-programs-properp-s->lr
(s-good-statep (s, c)
$\land$ s-restrict-subrps-progs (s-progs (s))
$\land$ s-restrictedp (s-progs (s), s-params (s))
$\land$ (*heap-size* $\not<$ s-total-heap-reqs (s-progs (s), s-params (s), *heap-size*))
$\land$ litatom (name (car (s-progs (s)))))
$\rightarrow$ lr-programs-properp (s->lr (s, *heap-size*, *max-ctrl*, *max-temp*, *word-size*),
cdr (lr-data-seg-table (s-progs (s),
s-params (s),
heap-size)))

DEFINITION:
lr-total-heap-reqs (*expr*, *alist*, *program-names*, *heap-size*, c)
= (s-total-heap-reqs (s-progs (logic->s (*expr*, *alist*, *program-names*)),
alist,
heap-size)
$+$ s-eval-heap-r (**t**, logic->s (*expr*, *alist*, *program-names*), c))

DEFINITION:
lr-max-ctrl-reqs (*expr*, *alist*, *program-names*, c)
= (2
$+$ length (s-params (logic->s (*expr*, *alist*, *program-names*)))
$+$ length (s-temps (logic->s (*expr*, *alist*, *program-names*)))
$+$ s-eval-ctrl-r (**t**, logic->s (*expr*, *alist*, *program-names*), c))

DEFINITION:
lr-max-temp-reqs (*expr*, *alist*, *program-names*, c)
= s-eval-temp-r (**t**, logic->s (*expr*, *alist*, *program-names*), c)

DEFINITION:
lr-max-word-size-reqs (*expr*, *alist*, *program-names*, *heap-size*, c)
= max (s-total-ws-reqs (s-progs (logic->s (*expr*, *alist*, *program-names*)),
s-params (logic->s (*expr*, *alist*, *program-names*)),
heap-size),
s-eval-ws-r (**t**, logic->s (*expr*, *alist*, *program-names*), c))

THEOREM: lr-eval-s-eval-equivalence-s->lr
let *s-lr* **be** s->lr (s, *heap-size*, *max-ctrl*, *max-temp*, *word-size*)
in
(proper-p-statep (lr->p (*s-lr*))
$\land$ lr-programs-properp (*s-lr*,

$$
\begin{array}{ll}
 & \qquad\qquad\qquad\qquad \text{cdr}\,(\text{lr-data-seg-table}\,(\text{s-progs}\,(s),\\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{s-params}\,(s),\\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{heap-size})))\\
\land & \text{lr-s-similar-statesp}\,(\text{s-params}\,(s),\\
 & \qquad\qquad\qquad\quad \text{s-temps}\,(s),\\
 & \qquad\qquad\qquad\quad \textit{s-lr},\\
 & \qquad\qquad\qquad\quad \text{cdr}\,(\text{lr-data-seg-table}\,(\text{s-progs}\,(s),\\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{s-params}\,(s),\\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{heap-size})))\\
\land & \text{s-restrictedp}\,(\text{s-progs}\,(s),\ \text{s-params}\,(s))\\
\land & (\textit{heap-size} \not< \text{s-total-heap-reqs}\,(\text{s-progs}\,(s),\\
 & \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{s-params}\,(s),\\
 & \qquad\qquad\qquad\qquad\qquad\qquad\quad \textit{heap-size}))\\
\land & \text{s-good-statep}\,(s,\ c)\\
\land & (\text{p-psw}\,(\text{lr-eval}\,(\mathbf{t},\ \textit{s-lr},\ c)) = \texttt{'run})\\
\land & (\text{s-pname}\,(s) = \text{name}\,(\text{car}\,(\text{s-progs}\,(s))))\\
\land & (\text{s-pos}\,(s) = \mathbf{nil}))\\
\rightarrow & \text{lr-valp}\,(\text{s-ans}\,(\text{s-eval}\,(\mathbf{t},\ s,\ c)),\\
 & \qquad\quad \text{car}\,(\text{p-temp-stk}\,(\text{lr-eval}\,(\mathbf{t},\ \textit{s-lr},\ c))),\\
 & \qquad\quad \text{p-data-segment}\,(\text{lr-eval}\,(\mathbf{t},\ \textit{s-lr},\ c)))\ \mathbf{endlet}
\end{array}
$$

THEOREM: same-signature-car-lr-data-seg-table

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\textit{heap-size} \not< \text{s-total-heap-reqs}\,(\textit{progs},\ \textit{params},\ \textit{heap-size}))}\\
\rightarrow & \text{same-signature}\,(\text{lr-init-data-seg}\,(\textit{heap-size}),\\
 & \qquad\qquad\qquad \text{car}\,(\text{lr-data-seg-table}\,(\textit{progs},\ \textit{params},\ \textit{heap-size})))
\end{array}
$$

THEOREM: lr-max-node-car-lr-data-seg-table

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\textit{heap-size} \not< \text{s-total-heap-reqs}\,(\textit{progs},\ \textit{params},\ \textit{heap-size}))}\\
\rightarrow & (\text{lr-max-node}\,(\text{car}\,(\text{lr-data-seg-table}\,(\textit{progs},\ \textit{params},\ \textit{heap-size})))\\
 & \ = \ \text{tag}\,(\texttt{'addr},\\
 & \qquad\qquad \text{cons}\,(\text{identity}\,(\text{LR-HEAP-NAME}),\\
 & \qquad\qquad\qquad \text{identity}\,(\text{LR-NODE-SIZE}) * \textit{heap-size})))
\end{array}
$$

THEOREM: lr-count-free-nodes-lr-data-seg-table-list-s-heap-reqs-help

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-proper-free-listp}\,(\textit{data-seg})}\\
\land & \text{lr-proper-p-areasp}\,(\textit{data-seg})\\
\land & \text{definedp}\,(\text{LR-HEAP-NAME},\ \textit{data-seg})\\
\land & \text{lr-minimum-heapp}\,(\textit{data-seg})\\
\land & \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\textit{data-seg}))\\
\land & (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \textit{data-seg}),\\
 & \qquad\qquad\qquad\qquad \text{lr-free-list-nodes}\,(\text{lr-max-node}\,(\textit{data-seg}),\\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{data-seg}),\\
 & \qquad\qquad\qquad\qquad \textit{data-seg})\\
 & \ \not< \ \text{s-heap-reqs-body}\,(\mathbf{t},\ \textit{body},\ \textit{data-seg},\ \textit{table})))\\
\rightarrow & ((\text{s-heap-reqs-body}\,(\mathbf{t},\ \textit{body},\ \textit{data-seg},\ \textit{table})
\end{array}
$$

```
    +   lr-count-free-nodes (fetch (identity (LR-FP-ADDR),
                                   car (lr-data-seg-table-body (t,
                                                                body,
                                                                data-seg,
                                                                table))),
                             lr-free-list-nodes (lr-max-node (data-seg),
                                                 car (lr-data-seg-table-body (t,
                                                                              body,
                                                                              data-seg,
                                                                              table))),
                             car (lr-data-seg-table-body (t,
                                                          body,
                                                          data-seg,
                                                          table))))
    =   lr-count-free-nodes (fetch (identity (LR-FP-ADDR), data-seg),
                             lr-free-list-nodes (lr-max-node (data-seg),
                                                 data-seg),
                             data-seg))
```

THEOREM: lr-count-free-nodes-lr-data-seg-table-list-s-heap-reqs

```
(lr-proper-free-listp (data-seg)
 ∧  lr-proper-p-areasp (data-seg)
 ∧  definedp (LR-HEAP-NAME, data-seg)
 ∧  lr-minimum-heapp (data-seg)
 ∧  lr-boundary-nodep (lr-max-node (data-seg))
 ∧  (lr-count-free-nodes (fetch (LR-FP-ADDR, data-seg),
                          lr-free-list-nodes (lr-max-node (data-seg),
                                              data-seg),
                          data-seg)
     ≮  s-heap-reqs-list (progs, data-seg, table)))
→  ((lr-count-free-nodes (fetch (LR-FP-ADDR,
                                 car (lr-data-seg-table-list (progs,
                                                              data-seg,
                                                              table))),
                          lr-free-list-nodes (lr-max-node (data-seg),
                                              car (lr-data-seg-table-list (progs,
                                                                           data-seg,
                                                                           table))),
                          car (lr-data-seg-table-list (progs, data-seg, table)))
     +  s-heap-reqs-list (progs, data-seg, table))
    =  lr-count-free-nodes (fetch (LR-FP-ADDR, data-seg),
                            lr-free-list-nodes (lr-max-node (data-seg),
                                                data-seg),
                            data-seg))
```

THEOREM: lr-max-node-car-lr-data-seg-table-list

$$
\begin{array}{l}
(\text{lr-proper-free-listp}\,(\mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{lr-proper-p-areasp}\,(\mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{definedp}\,(\text{LR-HEAP-NAME},\ \mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{lr-boundary-nodep}\,(\text{lr-max-node}\,(\mathit{data\text{-}seg}))) \\
\rightarrow\ \ (\text{lr-max-node}\,(\text{car}\,(\text{lr-data-seg-table-list}\,(\mathit{progs},\ \mathit{data\text{-}seg},\ \mathit{table}))) \\
\qquad =\ \ \text{lr-max-node}\,(\mathit{data\text{-}seg}))
\end{array}
$$

THEOREM: lr-count-free-nodes-lr-data-seg-table-list-s-heap-reqs-1

$$
\begin{array}{l}
(\text{lr-proper-heapp}\,(\mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{lr-proper-p-areasp}\,(\mathit{data\text{-}seg}) \\
\ \wedge\ \ (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \mathit{data\text{-}seg}), \\
\qquad\qquad \text{lr-free-list-nodes}\,(\text{lr-max-node}\,(\mathit{data\text{-}seg}), \\
\qquad\qquad\qquad \mathit{data\text{-}seg}), \\
\qquad\qquad \mathit{data\text{-}seg}) \\
\qquad \not< \ \ \text{s-heap-reqs-list}\,(\mathit{progs},\ \mathit{data\text{-}seg},\ \mathit{table})) \\
\ \wedge\ \ (\mathit{max\text{-}addr} = \text{lr-max-node}\,(\text{car}\,(\text{lr-data-seg-table-list}\,(\mathit{progs}, \\
\qquad\qquad\qquad \mathit{data\text{-}seg}, \\
\qquad\qquad\qquad \mathit{table}))))) \\
\rightarrow\ \ (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{identity}\,(\text{LR-FP-ADDR}), \\
\qquad\qquad\qquad \text{car}\,(\text{lr-data-seg-table-list}\,(\mathit{progs}, \\
\qquad\qquad\qquad\qquad \mathit{data\text{-}seg}, \\
\qquad\qquad\qquad\qquad \mathit{table}))), \\
\qquad\qquad \text{lr-free-list-nodes}\,(\mathit{max\text{-}addr}, \\
\qquad\qquad\qquad \text{car}\,(\text{lr-data-seg-table-list}\,(\mathit{progs}, \\
\qquad\qquad\qquad\qquad \mathit{data\text{-}seg}, \\
\qquad\qquad\qquad\qquad \mathit{table}))), \\
\qquad\qquad \text{car}\,(\text{lr-data-seg-table-list}\,(\mathit{progs},\ \mathit{data\text{-}seg},\ \mathit{table}))) \\
\qquad =\ \ (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \mathit{data\text{-}seg}), \\
\qquad\qquad\qquad \text{lr-free-list-nodes}\,(\text{lr-max-node}\,(\mathit{data\text{-}seg}), \\
\qquad\qquad\qquad\qquad \mathit{data\text{-}seg}), \\
\qquad\qquad\qquad \mathit{data\text{-}seg}) \\
\qquad\qquad -\ \ \text{s-heap-reqs-list}\,(\mathit{progs},\ \mathit{data\text{-}seg},\ \mathit{table})))
\end{array}
$$

THEOREM: lr-count-free-nodes-lr-init-data-seg-table-s-heap-reqs-1

$$
\begin{array}{l}
(\text{lr-proper-heapp}\,(\mathit{data\text{-}seg}) \\
\ \wedge\ \ \text{lr-proper-p-areasp}\,(\mathit{data\text{-}seg}) \\
\ \wedge\ \ (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \mathit{data\text{-}seg}), \\
\qquad\qquad \text{lr-free-list-nodes}\,(\text{lr-max-node}\,(\mathit{data\text{-}seg}), \\
\qquad\qquad\qquad \mathit{data\text{-}seg}), \\
\qquad\qquad \mathit{data\text{-}seg}) \\
\qquad \not< \ \ \text{s-init-heap-reqs}\,(\mathit{params},\ \mathit{data\text{-}seg},\ \mathit{table})) \\
\ \wedge\ \ (\mathit{max\text{-}addr} = \text{lr-max-node}\,(\text{car}\,(\text{lr-init-data-seg-table}\,(\mathit{params}, \\
\qquad\qquad\qquad \mathit{data\text{-}seg},
\end{array}
$$

table)))))

→ (lr-count-free-nodes (fetch (identity (LR-FP-ADDR), car (lr-init-data-seg-table (*params*, *data-seg*, *table*))), lr-free-list-nodes (*max-addr*, car (lr-init-data-seg-table (*params*, *data-seg*, *table*))), car (lr-init-data-seg-table (*params*, *data-seg*, *table*)))
= (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*), lr-free-list-nodes (lr-max-node (*data-seg*), *data-seg*), *data-seg*)
− s-init-heap-reqs (*params*, *data-seg*, *table*)))

THEOREM: not-lessp-lr-count-free-nodes-lr-data-seg-table-heap-r
((*heap-size* $\not<$ (s-total-heap-reqs (*progs*, *params*, *heap-size*) + *x*))
∧ s-restrictedp (*progs*, *params*))
→ (lr-count-free-nodes (fetch (identity (LR-FP-ADDR), car (lr-data-seg-table (*progs*, *params*, *heap-size*))), lr-free-list-nodes (lr-max-node (car (lr-data-seg-table (*progs*, *params*, *heap-size*))), car (lr-data-seg-table (*progs*, *params*, *heap-size*))), car (lr-data-seg-table (*progs*, *params*, *heap-size*)))
$\not<$ *x*)

THEOREM: lr-eval-s-eval-flag-run-s->lr
let *s-lr* **be** s->lr (*s*, *heap-size*, *max-ctrl*, *max-temp*, *word-size*)
in
(proper-p-statep (lr->p (*s-lr*))
∧ lr-programs-properp (*s-lr*, cdr (lr-data-seg-table (s-progs (*s*), s-params (*s*), *heap-size*)))
∧ lr-s-similar-statesp (s-params (*s*), s-temps (*s*), *s-lr*, cdr (lr-data-seg-table (s-progs (*s*),

s-params (s),
heap-size)))
$\wedge$ s-restrictedp (s-progs (s), s-params (s))
$\wedge$ s-good-statep (s, c)
$\wedge$ s-all-temps-setp (**t**,
s-body (car (s-progs (s))),
temp-alist-to-set (s-temps (s)))
$\wedge$ s-all-progs-temps-setp (s-progs (s))
$\wedge$ s-check-temps-setp (s-temps (s))
$\wedge$ (s-err-flag (s-eval (**t**, s, c)) = `'run`)
$\wedge$ (*heap-size* $\not<$ (s-total-heap-reqs (s-progs (s),
s-params (s),
heap-size)
\+ s-eval-heap-r (**t**, s, c)))
$\wedge$ (*max-ctrl* $\not<$ (p-ctrl-stk-size (p-ctrl-stk (*s-lr*))
\+ s-eval-ctrl-r (**t**, s, c)))
$\wedge$ (*max-temp* $\not<$ s-eval-temp-r (**t**, s, c))
$\wedge$ (*word-size* $\not<$ s-eval-ws-r (**t**, s, c))
$\wedge$ (*word-size* $\not<$ S-MAX-SUBR-REQS)
$\wedge$ (s-pname (s) = name (car (s-progs (s))))
$\wedge$ (s-pos (s) = **nil**))
$\rightarrow$ (p-psw (lr-eval (**t**, *s-lr*, c)) = `'run`) **endlet**

THEOREM: all-undef-addr-strip-cdrs-lr-make-initial-temps
all-undef-addrs (strip-cdrs (lr-make-initial-temps (x)))

THEOREM: lr-s-similar-temps-make-temps-entries-initial-temps
(all-undef-addrs (strip-cdrs (*lr-temps*))
$\wedge$ (length (*s-temps*) = length (*lr-temps*)))
$\rightarrow$ lr-s-similar-temps (make-temps-entries (*s-temps*), *lr-temps*, *data-seg*)

DEFINITION:
object-addrs (*object-list*, *table*)
= **if** listp (*object-list*)
then cons (cdr (assoc (car (*object-list*), *table*)),
object-addrs (cdr (*object-list*), *table*))
else nil endif

THEOREM: lr-valp-lr-compile-quote
(lr-proper-p-areasp (*data-seg*)
$\wedge$ lr-s-similar-const-table (*table*, *data-seg*)
$\wedge$ s-restricted-objectp (*flag*, *object*)
$\wedge$ lr-proper-heapp (*data-seg*)
$\wedge$ definedp (**f**, *table*)
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),

lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs (*flag*, *object*, *data-seg*, *table*)))
$\rightarrow$ **if** *flag* = `'list`
then lr-check-result1 (*object*,
object-addrs (*object*,
cdr (lr-compile-quote (*flag*,
object,
data-seg,
table))),
car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))
else lr-valp (*object*,
cdr (assoc (*object*,
cdr (lr-compile-quote (*flag*,
object,
data-seg,
table)))),
car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*))) **endif**

THEOREM: lr-init-data-seg-table-preserves-lr-valp
(lr-proper-p-areasp (*data-seg*)
$\land$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\land$ s-init-data-seg-restrictedp (*params*)
$\land$ lr-proper-heapp (*data-seg*)
$\land$ definedp (**f**, *table*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-init-heap-reqs (*params*, *data-seg*, *table*))
$\land$ lr-valp (*value*, *addr*, *data-seg*))
$\rightarrow$ lr-valp (*value*, *addr*, car (lr-init-data-seg-table (*params*, *data-seg*, *table*)))

THEOREM: lr-valp-lr-compile-quote-flag-t
(lr-proper-p-areasp (*data-seg*)
$\land$ lr-s-similar-const-table (*table*, *data-seg*)
$\land$ s-restricted-objectp (**t**, *object*)
$\land$ lr-proper-heapp (*data-seg*)
$\land$ definedp (**f**, *table*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)

$\not<$ s-heap-reqs (**t**, *object*, *data-seg*, *table*)))
$\rightarrow$ lr-valp (*object*,
cdr (assoc (*object*,
cdr (lr-compile-quote (**t**, *object*, *data-seg*, *table*)))),
car (lr-compile-quote (**t**, *object*, *data-seg*, *table*)))

THEOREM: lr-s-similar-params-pair-formals-lr-init-data-seg
(lr-proper-p-areasp (*data-seg*)
$\wedge$ lr-s-similar-const-table (*table*, *data-seg*)
$\wedge$ s-init-data-seg-restrictedp (*params*)
$\wedge$ lr-proper-heapp (*data-seg*)
$\wedge$ definedp (**f**, *table*)
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-init-heap-reqs (*params*, *data-seg*, *table*)))
$\rightarrow$ lr-s-similar-params (*params*,
pair-formals-with-addresses (*params*,
cdr (lr-init-data-seg-table (*params*,
data-seg,
table))),
car (lr-init-data-seg-table (*params*, *data-seg*, *table*)))

THEOREM: lr-data-seg-table-body-preserves-lr-valp
(lr-proper-p-areasp (*data-seg*)
$\wedge$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\wedge$ s-data-seg-body-restrictedp (*flag*, *body*)
$\wedge$ lr-proper-heapp (*data-seg*)
$\wedge$ definedp (**f**, *table*)
$\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs-body (*flag*, *body*, *data-seg*, *table*))
$\wedge$ lr-valp (*value*, *addr*, *data-seg*))
$\rightarrow$ lr-valp (*value*,
addr,
car (lr-data-seg-table-body (*flag*, *body*, *data-seg*, *table*)))

THEOREM: lr-data-seg-table-list-preserves-lr-valp
(lr-proper-p-areasp (*data-seg*)
$\wedge$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\wedge$ s-data-seg-list-restrictedp (*progs*)
$\wedge$ lr-proper-heapp (*data-seg*)

$\land$ definedp (**f**, *table*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs-list (*progs*, *data-seg*, *table*))
$\land$ lr-valp (*value*, *addr*, *data-seg*))
$\rightarrow$ lr-valp (*value*, *addr*, car (lr-data-seg-table-list (*progs*, *data-seg*, *table*)))

THEOREM: lr-data-seg-table-list-preserves-lr-s-similar-params
(lr-proper-p-areasp (*data-seg*)
$\land$ lr-good-pointerp-tablep (*table*, *data-seg*)
$\land$ s-data-seg-list-restrictedp (*progs*)
$\land$ lr-proper-heapp (*data-seg*)
$\land$ definedp (**f**, *table*)
$\land$ (lr-count-free-nodes (fetch (LR-FP-ADDR, *data-seg*),
lr-free-list-nodes (lr-max-node (*data-seg*),
data-seg),
data-seg)
$\not<$ s-heap-reqs-list (*progs*, *data-seg*, *table*))
$\land$ lr-s-similar-params (*s-params*, *lr-params*, *data-seg*))
$\rightarrow$ lr-s-similar-params (*s-params*,
lr-params,
car (lr-data-seg-table-list (*progs*, *data-seg*, *table*)))

DEFINITION:
all-definedp (*list*, *alist*)
= **if** listp (*list*)
then definedp (car (*list*), *alist*) $\land$ all-definedp (cdr (*list*), *alist*)
else t endif

THEOREM: assoc-definedp-table-lr-data-seg-table-body
definedp (*object*, *table*)
$\rightarrow$ (assoc (*object*, cdr (lr-data-seg-table-body (*flag*, *expr*, *data-seg*, *table*)))
= assoc (*object*, *table*))

THEOREM: assoc-definedp-table-lr-data-seg-table-list
definedp (*object*, *table*)
$\rightarrow$ (assoc (*object*, cdr (lr-data-seg-table-list (*progs*, *data-seg*, *table*)))
= assoc (*object*, *table*))

THEOREM: pair-formals-with-addresses-lr-data-seg-table-list
all-definedp (strip-cdrs (*params*), *table*)
$\rightarrow$ (pair-formals-with-addresses (*params*,
cdr (lr-data-seg-table-list (*progs*,

$$\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{data-seg}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{table}))) \\
\quad = \quad \text{pair-formals-with-addresses}\,(\textit{params},\ \textit{table}))
\end{array}$$

THEOREM: all-definedp-strip-cdrs-lr-init-data-seg-table

$$\begin{array}{ll}
\multicolumn{2}{l}{(\text{lr-s-similar-const-table}\,(\textit{table},\ \textit{data-seg})} \\
\land & \text{lr-proper-p-areasp}\,(\textit{data-seg}) \\
\land & \text{lr-proper-heapp}\,(\textit{data-seg}) \\
\land & \text{definedp}\,(\mathbf{f},\ \textit{table}) \\
\land & \text{s-init-data-seg-restrictedp}\,(\textit{params}) \\
\land & (\text{lr-count-free-nodes}\,(\text{fetch}\,(\text{LR-FP-ADDR},\ \textit{data-seg}), \\
& \qquad\qquad\qquad\qquad \text{lr-free-list-nodes}\,(\text{lr-max-node}\,(\textit{data-seg}), \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{data-seg}), \\
& \qquad\qquad\qquad\qquad \textit{data-seg}) \\
& \quad \not< \quad \text{s-init-heap-reqs}\,(\textit{params},\ \textit{data-seg},\ \textit{table}))) \\
\rightarrow & \text{all-definedp}\,(\text{strip-cdrs}\,(\textit{params}), \\
& \qquad\qquad\quad \text{cdr}\,(\text{lr-init-data-seg-table}\,(\textit{params},\ \textit{data-seg},\ \textit{table})))
\end{array}$$

THEOREM: lr-s-similar-params-pair-formals-with-addresses

$$\begin{array}{ll}
\multicolumn{2}{l}{((\textit{heap-size} \not< \text{s-total-heap-reqs}\,(\textit{progs},\ \textit{params},\ \textit{heap-size}))} \\
\land & \text{s-restrictedp}\,(\textit{progs},\ \textit{params})) \\
\rightarrow & \text{lr-s-similar-params}\,(\textit{params}, \\
& \qquad\qquad\qquad \text{pair-formals-with-addresses}\,(\textit{params}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{cdr}\,(\text{lr-data-seg-table}\,(\textit{progs}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{params}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{heap-size}))), \\
& \qquad\qquad\qquad \text{car}\,(\text{lr-data-seg-table}\,(\textit{progs},\ \textit{params},\ \textit{heap-size})))
\end{array}$$

THEOREM: lr-s-similar-statesp-s->lr-lr-data-seg-table

$$\begin{array}{ll}
\multicolumn{2}{l}{((\textit{heap-size} \not< \text{s-total-heap-reqs}\,(\text{s-progs}\,(s),\ \text{s-params}\,(s),\ \textit{heap-size}))} \\
\land & \text{s-restrictedp}\,(\text{s-progs}\,(s),\ \text{s-params}\,(s)) \\
\land & (\text{strip-cars}\,(\text{s-params}\,(s)) = \text{s-formals}\,(\text{car}\,(\text{s-progs}\,(s)))) \\
\land & (\text{s-pname}\,(s) = \text{name}\,(\text{car}\,(\text{s-progs}\,(s)))) \\
\land & (\text{s-temps}\,(s) = \text{make-temps-entries}\,(\text{s-temp-list}\,(\text{car}\,(\text{s-progs}\,(s)))))) \\
\rightarrow & \text{lr-s-similar-statesp}\,(\text{s-params}\,(s), \\
& \qquad\qquad\qquad \text{s-temps}\,(s), \\
& \qquad\qquad\qquad \text{s->lr}\,(s,\ \textit{heap-size},\ \textit{max-ctrl},\ \textit{max-temp},\ \textit{word-size}), \\
& \qquad\qquad\qquad \text{cdr}\,(\text{lr-data-seg-table}\,(\text{s-progs}\,(s), \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{s-params}\,(s), \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{heap-size})))
\end{array}$$

THEOREM: p-ctrl-stk-size-p-ctrl-stk-s->lr

$$\begin{array}{ll}
\multicolumn{2}{l}{\text{p-ctrl-stk-size}\,(\text{p-ctrl-stk}\,(\text{s->lr}\,(s,\ \textit{heap-size},\ \textit{max-ctrl},\ \textit{max-temp},\ \textit{word-size})))} \\
= & (2 + \text{length}\,(\text{s-params}\,(s)) + \text{length}\,(\text{s-temps}\,(s)))
\end{array}$$

THEOREM: s->lr-ok
let *s-lr* **be** s->lr (s, *heap-size*, *max-ctrl*, *max-temp*, *word-size*)
in
(s-good-statep (s, c)
$\land$ s-all-temps-setp (**t**,
s-body (car (s-progs (s))),
temp-alist-to-set (s-temps (s)))
$\land$ s-all-progs-temps-setp (s-progs (s))
$\land$ (s-temps (s)
= make-temps-entries (s-temp-list (car (s-progs (s)))))
$\land$ s-check-temps-setp (s-temps (s))
$\land$ s-restrictedp (s-progs (s), s-params (s))
$\land$ (*heap-size* $\not<$ (s-total-heap-reqs (s-progs (s),
s-params (s),
heap-size)
$+$ s-eval-heap-r (**t**, s, c)))
$\land$ s-restrict-subrps-progs (s-progs (s))
$\land$ (*max-ctrl* $\not<$ (2
$+$ length (s-params (s))
$+$ length (s-temps (s))
$+$ s-eval-ctrl-r (**t**, s, c)))
$\land$ (*max-ctrl* $\in \mathbf{N}$)
$\land$ (*max-ctrl* $<$ exp (2, *word-size*))
$\land$ (*max-temp* $\in \mathbf{N}$)
$\land$ (*max-temp* $\not<$ s-eval-temp-r (**t**, s, c))
$\land$ (*max-temp* $<$ exp (2, *word-size*))
$\land$ (*word-size* $\not<$ max (s-total-ws-reqs (s-progs (s),
s-params (s),
heap-size),
s-eval-ws-r (**t**, s, c)))
$\land$ (*word-size* $\in \mathbf{N}$)
$\land$ litatom (name (car (s-progs (s))))
$\land$ (s-pname (s) = name (car (s-progs (s))))
$\land$ (s-pos (s) = **nil**)
$\land$ all-litatoms (strip-cars (s-params (s)))
$\land$ no-duplicatesp (strip-cars (s-progs (s)))
$\land$ (strip-cars (s-params (s)) = s-formals (car (s-progs (s))))
$\land$ (s-err-flag (s-eval (**t**, s, c)) = `'run`))
$\rightarrow$ lr-valp (s-ans (s-eval (**t**, s, c)),
car (p-temp-stk (lr-eval (**t**, *s-lr*, c))),
p-data-segment (lr-eval (**t**, *s-lr*, c))) **endlet**

THEOREM: s-good-state-logic->s
(l-proper-programsp (*prog-names*)

$\wedge$ l-proper-exprp (**t**, *expr*, *prog-names*, strip-cars (*alist*))
$\wedge$ all-litatoms (strip-cars (*alist*)))
$\rightarrow$ s-good-statep (logic->s (*expr*, *alist*, *prog-names*), *c*)

THEOREM: s-body-car-s-progs-logic->s
s-body (car (s-progs (logic->s (*expr*, *alist*, *pnames*)))) = *expr*

THEOREM: l-proper-programsp-s-progs-logic->s
(l-proper-programsp (*pnames*) $\wedge$ l-proper-exprp (**t**, *expr*, *pnames*, *formals*))
$\rightarrow$ s-all-progs-temps-setp (s-progs (logic->s (*expr*, *alist*, *pnames*)))

THEOREM: s-temps-logic->s
s-temps (logic->s (*expr*, *alist*, *pnames*)) = **nil**

THEOREM: s-temp-list-car-s-progs-logic->s
s-temp-list (car (s-progs (logic->s (*expr*, *alist*, *pnames*)))) = **nil**

THEOREM: s-params-logic->s
s-params (logic->s (*expr*, *alist*, *pnames*)) = *alist*

DEFINITION:
l-data-seg-body-restrictedp (*flag*, *expr*)
= **if** *flag* = `'list`
  then if listp (*expr*)
    then l-data-seg-body-restrictedp (**t**, car (*expr*))
      $\wedge$ l-data-seg-body-restrictedp (`'list`, cdr (*expr*))
    else t endif
  elseif listp (*expr*)
  then if car (*expr*) = `'quote`
    then s-restricted-objectp (**t**, cadr (*expr*))
    else l-data-seg-body-restrictedp (`'list`, cdr (*expr*)) **endif**
  else t endif

DEFINITION:
l-data-seg-list-restrictedp (*fun-names*)
= **if** listp (*fun-names*)
  then l-data-seg-body-restrictedp (**t**, body (car (*fun-names*)))
    $\wedge$ l-data-seg-list-restrictedp (cdr (*fun-names*))
  else t endif

DEFINITION:
l-restrictedp (*fun-names*, *alist*)
= (s-init-data-seg-restrictedp (*alist*)
  $\wedge$ l-data-seg-list-restrictedp (*fun-names*))

THEOREM: l-data-seg-body-restrictedp-s-data-seg-body-restrictedp
l-data-seg-body-restrictedp (*flag*, *body*)
$\rightarrow$ s-data-seg-body-restrictedp (*flag*, *body*)

THEOREM: l-data-seg-body-restrictedp-delete-all
l-data-seg-list-restrictedp (*pnames*)
$\rightarrow$ l-data-seg-list-restrictedp (delete-all (*name*, *pnames*))

THEOREM: l-data-seg-list-restrictedp-s-data-seg-list-restrictedp
(l-data-seg-list-restrictedp (*pnames*) $\land$ l-data-seg-body-restrictedp (**t**, *expr*))
$\rightarrow$ s-data-seg-list-restrictedp (s-progs (logic->s (*expr*, *alist*, *pnames*)))

THEOREM: l-restrict-subrps-s-restrict-subrps
(l-restrict-subrps (*flag*, *expr*)
$\land$ l-proper-exprp (*flag*, *expr*, *program-names*, *formals*))
$\rightarrow$ s-restrict-subrps (*flag*, *expr*)

THEOREM: l-restrict-subrps-progs-delete-all
l-restrict-subrps-progs (*pnames*)
$\rightarrow$ l-restrict-subrps-progs (delete-all (*name*, *pnames*))

THEOREM: l-restrict-subrps-progs-s-restrict-subrps-progs
(l-restrict-subrps-progs (*pnames*)
$\land$ l-proper-programsp-1 (*pnames*, *program-names*))
$\rightarrow$ s-restrict-subrps-progs (s-construct-programs (remove-duplicates (*pnames*)))

THEOREM: l-restrict-subrps-progs-s-restrict-subrps-progs-logic->s
(l-restrict-subrps-progs (*pnames*)
$\land$ l-restrict-subrps (**t**, *expr*)
$\land$ l-proper-exprp (**t**, *expr*, *program-names*, *formals*)
$\land$ l-proper-programsp (*pnames*))
$\rightarrow$ s-restrict-subrps-progs (s-progs (logic->s (*expr*, *alist*, *pnames*)))

THEOREM: name-car-s-progs-logic->s
name (car (s-progs (logic->s (*expr*, *alist*, *pnames*)))) = `'main`

THEOREM: s-pname-logic->s
s-pname (logic->s (*expr*, *alist*, *pnames*)) = `'main`

THEOREM: s-pos-logic->s
s-pos (logic->s (*expr*, *alist*, *pnames*)) = **nil**

THEOREM: definedp-user-fname-s-construct-programs
(litatom (*name*) $\land$ all-litatoms-not-plist (*pnames*))
$\rightarrow$ (definedp (user-fname (*name*), s-construct-programs (*pnames*))
= (*name* $\in$ *pnames*))

THEOREM: no-duplicatesp-strip-cars-s-construct-programs
all-litatoms-not-plist (*pnames*)
$\rightarrow$ (no-duplicatesp (strip-cars (s-construct-programs (*pnames*)))
$=$ no-duplicatesp (*pnames*))

THEOREM: all-user-fnamesp-strip-cars-s-construct-programs
all-litatoms-not-plist (*pnames*)
$\rightarrow$ all-user-fnamesp (strip-cars (s-construct-programs (*pnames*)))

THEOREM: no-duplicatesp-strip-cars-s-progs-logic->s
all-litatoms-not-plist (*pnames*)
$\rightarrow$ no-duplicatesp (strip-cars (s-progs (logic->s (*expr*, *alist*, *pnames*))))

THEOREM: s-formals-car-s-progs-logic->s
s-formals (car (s-progs (logic->s (*expr*, *alist*, *pnames*)))) $=$ strip-cars (*alist*)

THEOREM: s-expr-logic->s
s-expr (logic->s (*expr*, *alist*, *pnames*)) $=$ *expr*

THEOREM: all-litatoms-not-plist-lr-proper-programsp
l-proper-programsp (*prog-names*) $\rightarrow$ all-litatoms-not-plist (*prog-names*)

THEOREM: logic->lr-ok-really
(l-proper-exprp (**t**, *expr*, *pnames*, strip-cars (*alist*))
$\land$ l-proper-programsp (*pnames*)
$\land$ all-litatoms (strip-cars (*alist*))
$\land$ l-data-seg-body-restrictedp (**t**, *expr*)
$\land$ l-restrictedp (*pnames*, *alist*)
$\land$ v&c$ (**t**, *expr*, *alist*)
$\land$ (cdr (v&c$ (**t**, *expr*, *alist*)) $<$ *c*)
$\land$ l-restrict-subrps (**t**, *expr*)
$\land$ l-restrict-subrps-progs (*pnames*)
$\land$ (*heap-size* $\not<$ lr-total-heap-reqs (*expr*, *alist*, *pnames*, *heap-size*, *c*))
$\land$ (*max-ctrl* $\not<$ lr-max-ctrl-reqs (*expr*, *alist*, *pnames*, *c*))
$\land$ (*max-ctrl* $<$ exp (2, *word-size*))
$\land$ (*max-ctrl* $\in \mathbf{N}$)
$\land$ (*max-temp* $\not<$ lr-max-temp-reqs (*expr*, *alist*, *pnames*, *c*))
$\land$ (*max-temp* $<$ exp (2, *word-size*))
$\land$ (*max-temp* $\in \mathbf{N}$)
$\land$ (*word-size* $\not<$ lr-max-word-size-reqs (*expr*, *alist*, *pnames*, *heap-size*, *c*))
$\land$ (*word-size* $\in \mathbf{N}$))
$\rightarrow$ lr-valp (car (v&c$ (**t**, *expr*, *alist*)),
car (p-temp-stk (lr-eval (**t**,
s->lr (logic->s (*expr*, *alist*, *pnames*),
heap-size,

```
                                        max-ctrl,
                                        max-temp,
                                        word-size),
                                   c))),
            p-data-segment (lr-eval (t,
                                   s->lr (logic->s (expr, alist, pnames),
                                        heap-size,
                                        max-ctrl,
                                        max-temp,
                                        word-size),
                                   c)))
```

```
; -------------------------------------------------------------
; was p1.events
; -------------------------------------------------------------
```

DEFINITION:
logic->p (*expr*, *alist*, *pnames*, *heap-size*, *max-ctrl*, *max-temp*, *word-size*)
= lr->p (s->lr (logic->s (*expr*, *alist*, *pnames*),
heap-size,
max-ctrl,
max-temp,
word-size))

DEFINITION:
p-run-subr-clock (*l*, *new-l*)
= **case on** car (lr-expr (*l*)):
case = *car*
then p-car-clock (p-set-pc (lr->p (*new-l*), lr-return-pc (*l*)))
case = *cdr*
then p-cdr-clock (p-set-pc (lr->p (*new-l*), lr-return-pc (*l*)))
case = *cons*
then p-cons-clock (p-set-pc (lr->p (*new-l*), lr-return-pc (*l*)))
case = *false*
then p-false-clock (p-set-pc (lr->p (*new-l*), lr-return-pc (*l*)))
case = *falsep*
then p-falsep-clock (p-set-pc (lr->p (*new-l*), lr-return-pc (*l*)))
case = *listp*
then p-listp-clock (p-set-pc (lr->p (*new-l*), lr-return-pc (*l*)))
case = *nlistp*
then p-nlistp-clock (p-set-pc (lr->p (*new-l*), lr-return-pc (*l*)))
case = *true*
then p-true-clock (p-set-pc (lr->p (*new-l*), lr-return-pc (*l*)))
case = *truep*

```
    then p-truep-clock (p-set-pc (lr->p (new-l), lr-return-pc (l)))
  otherwise 0 endcase
```

EVENT: Disable p-run-subr-clock.

DEFINITION:

```
p-clock1 (flag, l, c)
=  if p-psw (l) ≠ 'run then 0
   elseif flag = 'list
   then if offset (p-pc (l)) ≃ nil then 0
        elseif listp (lr-expr-list (l))
        then p-clock1 (t, l, c)
             +  p-clock1 ('list,
                         lr-set-expr (lr-eval (t, l, c),
                                      l,
                                      nx (offset (p-pc (l)))),
                         c)
        else 0 endif
   elseif c ≃ 0 then 0
   elseif litatom (lr-expr (l)) then 1
   elseif lr-expr (l) ≃ nil then 0
   elseif car (lr-expr (l)) = 'if
   then let test be lr-if-ok (lr-eval (t,
                                       lr-set-pos (l,
                                                   dv (offset (p-pc (l)), 1)),
                                       c))
        in
        if p-psw (test) = 'run
        then if top (p-temp-stk (test)) ≠ LR-F-ADDR
             then p-clock1 (t,
                            lr-set-pos (l, dv (offset (p-pc (l)), 1)),
                            c)
                  +  3
                  +  p-clock1 (t,
                               lr-set-expr (lr-pop-tstk (test),
                                            l,
                                            dv (offset (p-pc (l)),
                                                2)),
                               c)
                  +  1
             else p-clock1 (t,
                            lr-set-pos (l,
                                        dv (offset (p-pc (l)), 1)),
                            c)
```

```
                  +   3
                  +   p-clock1 (t,
                                lr-set-expr (lr-pop-tstk (test),
                                             l,
                                             dv (offset (p-pc (l)),
                                                 3)),
                                c) endif
      else p-clock1 (t,
                     lr-set-pos (l, dv (offset (p-pc (l)), 1)),
                     c) endif endlet
elseif car (lr-expr (l)) = S-TEMP-EVAL
then p-clock1 (t, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c) + 1
elseif car (lr-expr (l)) = S-TEMP-TEST
then if lr-eval-temp-setp (l)  then 5
      else 4
            +   p-clock1 (t,
                          lr-set-pos (l, dv (offset (p-pc (l)), 1)),
                          c)
            +   2 endif
elseif car (lr-expr (l)) = S-TEMP-FETCH  then 1
elseif car (lr-expr (l)) = 'quote then 1
elseif p-psw (lr-eval ('list, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c))
      ≠   'run
then p-clock1 ('list, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c)
elseif subrp (car (lr-expr (l)))
then p-clock1 ('list, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c)
      +   p-run-subr-clock (l,
                            lr-eval ('list,
                                     lr-set-pos (l,
                                                 dv (offset (p-pc (l)), 1)),
                                     c))
elseif litatom (car (lr-expr (l)))
then let fs  be  lr-funcall (l,
                             lr-eval ('list,
                                      lr-set-pos (l,
                                                  dv (offset (p-pc (l)), 1)),
                                      c))
      in
      p-clock1 ('list, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c)
      +   1
      +   p-clock1 (t, fs, c − 1)
      +   1 endlet
else 0 endif
```

DEFINITION:
lr-good-pointerps (*list*, *data-seg*)
= **if** listp (*list*)
  then lr-good-pointerp (car (*list*), *data-seg*)
    $\land$ lr-good-pointerps (cdr (*list*), *data-seg*)
  else t endif

DEFINITION:
lr-proper-ctrl-stkp (*ctrl-stk*, *data-seg*)
= **if** *ctrl-stk* $\simeq$ **nil then** *ctrl-stk* = **nil**
  else lr-good-pointerps (strip-cdrs (bindings (top (*ctrl-stk*))),
    data-seg)
    $\land$ lr-proper-ctrl-stkp (pop (*ctrl-stk*), *data-seg*) **endif**

DEFINITION:
lr-p-proper-statep (*temp-stk*, *ctrl-stk*, *data-seg*, *table*)
= (lr-proper-ctrl-stkp (*ctrl-stk*, *data-seg*)
  $\land$ lr-good-pointerps (*temp-stk*, *data-seg*)
  $\land$ lr-proper-heapp (*data-seg*)
  $\land$ lr-good-pointerp-tablep (*table*, *data-seg*))

EVENT: Disable lr-p-proper-statep.

THEOREM: definedp-cdr-assoc-lr-good-pointerps
((*addr* $\in$ *list*) $\land$ lr-good-pointerps (*list*, *data-seg*))
$\rightarrow$ lr-good-pointerp (*addr*, *data-seg*)

THEOREM: lr-p-proper-statep-lr-push-tstk-cdr-assoc-lr-expr
(litatom (lr-expr (l))
$\land$ proper-p-statep (lr->p (l))
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ lr-p-proper-statep (p-temp-stk (l),
    p-ctrl-stk (l),
    p-data-segment (l),
    table)
$\land$ (p-psw (lr-push-tstk (l,
    cdr (assoc (lr-expr (l),
      bindings (car (p-ctrl-stk (l)))))))
  = `'run`))
$\rightarrow$ lr-p-proper-statep (p-temp-stk (lr-push-tstk (l,
    cdr (assoc (lr-expr (l),
      bindings (car (p-ctrl-stk (l))))))),
    p-ctrl-stk (l),

p-data-segment (l),
table)

THEOREM: lr-p-proper-statep-cdr-temp-stk
lr-p-proper-statep (*temp-stk*, *ctrl-stk*, *data-seg*, *table*)
$\rightarrow$ lr-p-proper-statep (cdr (*temp-stk*), *ctrl-stk*, *data-seg*, *table*)

THEOREM: lr-good-pointerps-put-assoc
(lr-good-pointerp (*addr*, *data-seg*)
$\land$ lr-good-pointerps (strip-cdrs (*bindings*), *data-seg*))
$\rightarrow$ lr-good-pointerps (strip-cdrs (put-assoc (*addr*, *expr*, *bindings*)), *data-seg*)

THEOREM: lr-p-proper-statep-cons-p-frame-put-assoc
(lr-p-proper-statep (*temp-stk*, *ctrl-stk*, *data-seg*, *table*)
$\land$ listp (*temp-stk*)
$\land$ listp (*ctrl-stk*)
$\land$ (*cdr-ctrl-stk* = cdr (*ctrl-stk*)))
$\rightarrow$ lr-p-proper-statep (*temp-stk*,
cons (p-frame (put-assoc (car (*temp-stk*),
expr,
bindings (car (*ctrl-stk*))),
ret-pc),
cdr-ctrl-stk),
data-seg,
table)

THEOREM: lr-eval-leaves-listp-p-ctrl-stk-lr->p-lr-set-pos
(proper-p-statep (lr->p (l))
$\land$ good-posp (*flag*, *pos*, program-body (p-current-program (l)))
$\land$ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ (p-psw (lr-eval (*flag*, lr-set-pos (l, *pos*), c)) = `'run`))
$\rightarrow$ listp (p-ctrl-stk (lr-eval (*flag*, lr-set-pos (l, *pos*), c)))

THEOREM: lr-p-proper-statep-cdr-assoc-caddr-lr-expr-bindings
(proper-p-statep (lr->p (l))
$\land$ lr-p-proper-statep (*temp-stk*, p-ctrl-stk (l), *data-seg*, *table*)
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ listp (lr-expr (l))
$\land$ ((car (lr-expr (l)) = S-TEMP-TEST)
$\lor$ (car (lr-expr (l)) = S-TEMP-FETCH)))
$\rightarrow$ lr-p-proper-statep (cons (cdr (assoc (caddr (lr-expr (l)),
bindings (car (p-ctrl-stk (l))))),
temp-stk),

p-ctrl-stk (l),
data-seg,
table)

THEOREM: member-strip-cdrs-lr-good-pointerp-tablep
((*object* $\in$ strip-cdrs (*table*)) $\wedge$ lr-good-pointerp-tablep (*table*, *data-seg*))
$\rightarrow$ lr-good-pointerp (*object*, *data-seg*)

THEOREM: lr-p-proper-statep-p-temps-stk-lr-push-tstk-quote
(listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) = `'quote`)
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (lr-push-tstk (l, cadr (lr-expr (l)))) = `'run`)
$\wedge$ lr-p-proper-statep (p-temp-stk (l), *ctrl-stk*, *data-seg*, *table*))
$\rightarrow$ lr-p-proper-statep (p-temp-stk (lr-push-tstk (l, cadr (lr-expr (l)))),
ctrl-stk,
data-seg,
table)

THEOREM: p-run-subr-preserves-lr-good-pointerp-tablep
(listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
$\wedge$ (car (lr-expr (l)) $\neq$ `'quote`)
$\wedge$ (p-psw (*new-l*) = `'run`)
$\wedge$ subrp (car (lr-expr (l)))
$\wedge$ lr-good-pointerp-tablep (*table2*, p-data-segment (*new-l*))
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table1*)
$\wedge$ lr-programs-properp (*new-l*, *table1*)
$\wedge$ (p-psw (p-run-subr (car (lr-expr (l)),
p-set-pc (lr->p (*new-l*), lr-return-pc (l))))
= `'run`)
$\wedge$ lr-proper-heapp (p-data-segment (*new-l*))
$\wedge$ proper-p-statep (lr->p (*new-l*))
$\wedge$ (p-prog-segment (*new-l*) = p-prog-segment (l)))
$\rightarrow$ lr-good-pointerp-tablep (*table2*,
p-data-segment (p-run-subr (car (lr-expr (l)),
p-set-pc (lr->p (*new-l*),
lr-return-pc (l)))))

THEOREM: lr-good-pointerps-deposit-free-ptr
lr-good-pointerps (*list*, deposit (*anything*, identity (LR-FP-ADDR), *data-seg*))
= lr-good-pointerps (*list*, *data-seg*)

THEOREM: lr-proper-ctrl-stkp-deposit-free-ptr
lr-proper-ctrl-stkp (*ctrl-stk*, deposit (*anything*, identity (LR-FP-ADDR), *data-seg*))
= lr-proper-ctrl-stkp (*ctrl-stk*, *data-seg*)

THEOREM: lr-good-pointerp-deposit-a-list-node
(lr-good-pointerp (*addr1*, *data-seg*)
∧ (type (*addr2*) = `'addr`)
∧ (cddr (*addr2*) = **nil**)
∧ listp (*addr2*)
∧ adpp (untag (*addr2*), *data-seg*)
∧ lr-boundary-nodep (*addr2*)
∧ (area-name (*addr2*) = `'heap`)
∧ (type (*ref-count*) = `'nat`))
→ lr-good-pointerp (*addr1*,
deposit-a-list (list (x, *ref-count*, y, z), *addr2*, *data-seg*))

THEOREM: lr-good-pointerps-deposit-a-list-node
(lr-good-pointerps (*list*, *data-seg*)
∧ (type (*addr*) = `'addr`)
∧ (cddr (*addr*) = **nil**)
∧ listp (*addr*)
∧ adpp (untag (*addr*), *data-seg*)
∧ lr-boundary-nodep (*addr*)
∧ (area-name (*addr*) = `'heap`)
∧ (type (*ref-count*) = `'nat`))
→ lr-good-pointerps (*list*,
deposit-a-list (list (x, *ref-count*, y, z), *addr*, *data-seg*))

THEOREM: lr-proper-ctrl-stkp-deposit-a-list-node
(lr-proper-ctrl-stkp (*list*, *data-seg*)
∧ (type (*addr*) = `'addr`)
∧ (cddr (*addr*) = **nil**)
∧ listp (*addr*)
∧ adpp (untag (*addr*), *data-seg*)
∧ lr-boundary-nodep (*addr*)
∧ (area-name (*addr*) = `'heap`)
∧ (type (*ref-count*) = `'nat`))
→ lr-proper-ctrl-stkp (*list*,
deposit-a-list (list (x, *ref-count*, y, z),
addr,
data-seg))

THEOREM: p-run-subr-preserves-lr-proper-ctrl-stkp
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
∧ lr-programs-properp (l, *table*)

$\wedge$ listp (lr-expr (l))
$\wedge$ subrp (car (lr-expr (l)))
$\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
$\wedge$ (car (lr-expr (l)) $\neq$ `'quote`)
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ lr-proper-heapp (p-data-segment (lr-eval (`'list`, lr-set-pos (l, pos), c)))
$\wedge$ lr-proper-ctrl-stkp (p-ctrl-stk (lr-eval (`'list`, lr-set-pos (l, pos), c)),
p-data-segment (lr-eval (`'list`,
lr-set-pos (l, pos),
c)))
$\wedge$ (p-psw (lr-eval (`'list`, lr-set-pos (l, pos), c)) = `'run`)
$\wedge$ (p-psw (p-run-subr (car (lr-expr (l)),
p-set-pc (lr->p (lr-eval (`'list`, lr-set-pos (l, pos), c)),
lr-return-pc (l))))
= `'run`)
$\wedge$ (pos = dv (offset (p-pc (l)), `1`)))
$\rightarrow$ lr-proper-ctrl-stkp (p-ctrl-stk (lr-eval (`'list`, lr-set-pos (l, pos), c)),
p-data-segment (p-run-subr (car (lr-expr (l)),
p-set-pc (lr->p (lr-eval (`'list`,
lr-set-pos (l,
pos),
c)),
lr-return-pc (l)))))

THEOREM: lr-good-pointerps-cons-lr-f-addr-lr-proper-heapp
(lr-good-pointerps ($list$, $data\text{-}seg$)
$\wedge$ lr-proper-heapp ($data\text{-}seg$)
$\wedge$ lr-proper-p-areasp ($data\text{-}seg$))
$\rightarrow$ lr-good-pointerps (cons (identity (LR-F-ADDR), $list$), $data\text{-}seg$)

THEOREM: lr-good-pointerps-cons-lr-t-addr-lr-proper-heapp
(lr-good-pointerps ($list$, $data\text{-}seg$)
$\wedge$ lr-proper-heapp ($data\text{-}seg$)
$\wedge$ lr-proper-p-areasp ($data\text{-}seg$))
$\rightarrow$ lr-good-pointerps (cons (identity (LR-T-ADDR), $list$), $data\text{-}seg$)

THEOREM: lr-good-pointerps-cons-lr-0-addr-lr-proper-heapp
(lr-good-pointerps ($list$, $data\text{-}seg$)
$\wedge$ lr-proper-heapp ($data\text{-}seg$)
$\wedge$ lr-proper-p-areasp ($data\text{-}seg$))
$\rightarrow$ lr-good-pointerps (cons (identity (LR-0-ADDR), $list$), $data\text{-}seg$)

THEOREM: lr-good-pointerps-cdr
lr-good-pointerps ($list$, $data\text{-}seg$) $\rightarrow$ lr-good-pointerps (cdr ($list$), $data\text{-}seg$)

THEOREM: lr-good-pointerps-cons-fetch-car-temp-stk-cdr-car

(lr-good-pointerps (*temp-stk*, *data-seg*)
∧ lr-proper-heapp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*)
∧ (length (*temp-stk*) $\not<$ `1`)
∧ (fetch (car (*temp-stk*), *data-seg*) = tag (`'nat`, LR-CONS-TAG)))
→ lr-good-pointerps (cons (fetch (add-addr (car (*temp-stk*),
identity (LR-CAR-OFFSET)),
data-seg),
cdr (*temp-stk*)),
data-seg)

THEOREM: lr-good-pointerps-cons-fetch-car-temp-stk-cdr-cdr

(lr-good-pointerps (*temp-stk*, *data-seg*)
∧ lr-proper-heapp (*data-seg*)
∧ lr-proper-p-areasp (*data-seg*)
∧ (length (*temp-stk*) $\not<$ `1`)
∧ (fetch (car (*temp-stk*), *data-seg*) = tag (`'nat`, LR-CONS-TAG)))
→ lr-good-pointerps (cons (fetch (add-addr (car (*temp-stk*),
identity (LR-CDR-OFFSET)),
data-seg),
cdr (*temp-stk*)),
data-seg)

THEOREM: lr-good-pointerps-cons-fetch-fp-addr-deposit-a-list-cons

(lr-proper-heapp (*data-seg*)
∧ lr-good-pointerps (*temp-stk*, *data-seg*)
∧ (*fp-addr* = fetch (identity (LR-FP-ADDR), *data-seg*))
∧ (type (*ref-count*) = `'nat`))
→ lr-good-pointerps (cons (fetch (identity (LR-FP-ADDR), *data-seg*), *temp-stk*),
deposit-a-list (list (x, *ref-count*, y, z),
fp-addr,
data-seg))

THEOREM: p-run-subr-preserves-lr-good-pointerps

let *lr-eval* **be** lr-eval (`'list`, lr-set-pos (l, *pos*), c)
in
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
∧ lr-programs-properp (l, *table*)
∧ listp (lr-expr (l))
∧ subrp (car (lr-expr (l)))
∧ (car (lr-expr (l)) ≠ `'if`)
∧ (car (lr-expr (l)) ≠ `'quote`)
∧ proper-p-statep (lr->p (l))
∧ lr-proper-heapp (p-data-segment (*lr-eval*))

$\land$ lr-good-pointerps (p-temp-stk (*lr-eval*), p-data-segment (*lr-eval*))
$\land$ (p-psw (*lr-eval*) = `'run`)
$\land$ (p-psw (p-run-subr (car (lr-expr (*l*)),
p-set-pc (lr->p (*lr-eval*), lr-return-pc (*l*))))
= `'run`)
$\land$ (length (p-temp-stk (*lr-eval*)) $\not<$ arity (car (lr-expr (*l*))))
$\land$ (*pos* = dv (offset (p-pc (*l*)), `1`)))
$\rightarrow$ lr-good-pointerps (p-temp-stk (p-run-subr (car (lr-expr (*l*)),
p-set-pc (lr->p (*lr-eval*),
lr-return-pc (*l*)))),
p-data-segment (p-run-subr (car (lr-expr (*l*)),
p-set-pc (lr->p (*lr-eval*),
lr-return-pc (*l*))))) **endlet**

THEOREM: p-run-subr-preserves-lr-proper-heapp2-alt
(good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\land$ lr-programs-properp (*l*, *table*)
$\land$ lr-programs-properp (*new-l*, *table*)
$\land$ listp (lr-expr (*l*))
$\land$ proper-p-statep (lr->p (*new-l*))
$\land$ lr-proper-free-listp (p-data-segment (*new-l*))
$\land$ adpp (untag (lr-max-node (p-data-segment (*new-l*))), p-data-segment (*new-l*))
$\land$ lr-boundary-nodep (lr-max-node (p-data-segment (*new-l*)))
$\land$ (p-psw (*new-l*) = `'run`)
$\land$ (p-psw (p-run-subr (car (lr-expr (*l*)),
p-set-pc (lr->p (*new-l*), lr-return-pc (*l*))))
= `'run`)
$\land$ lr-good-pointerps (p-temp-stk (*new-l*), p-data-segment (*new-l*))
$\land$ (length (p-temp-stk (*new-l*)) $\not<$ arity (car (lr-expr (*l*))))
$\land$ lr-proper-heapp2 (*addr*, p-data-segment (*new-l*))
$\land$ lr-nodep (*addr*, p-data-segment (*new-l*))
$\land$ (p-prog-segment (*l*) = p-prog-segment (*new-l*)))
$\rightarrow$ lr-proper-heapp2 (*addr*,
p-data-segment (p-run-subr (car (lr-expr (*l*)),
p-set-pc (lr->p (*new-l*),
lr-return-pc (*l*)))))

THEOREM: p-run-subr-preserves-lr-proper-heapp
(good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\land$ lr-programs-properp (*l*, *table*)
$\land$ lr-programs-properp (*new-l*, *table*)
$\land$ listp (lr-expr (*l*))
$\land$ (car (lr-expr (*l*)) $\neq$ `'if`)
$\land$ subrp (car (lr-expr (*l*)))

$\wedge$ proper-p-statep (lr->p (*new-l*))
$\wedge$ (p-psw (*new-l*) = `'run`)
$\wedge$ (p-psw (p-run-subr (car (lr-expr (l)),
    p-set-pc (lr->p (*new-l*), lr-return-pc (l))))
  = `'run`)
$\wedge$ (p-prog-segment (l) = p-prog-segment (*new-l*))
$\wedge$ (area-name (p-pc (l)) = area-name (p-pc (*new-l*)))
$\wedge$ lr-proper-heapp (p-data-segment (*new-l*))
$\wedge$ lr-good-pointerps (p-temp-stk (*new-l*), p-data-segment (*new-l*))
$\wedge$ (length (p-temp-stk (*new-l*)) $\not<$ arity (car (lr-expr (l)))))
$\rightarrow$ lr-proper-heapp (p-data-segment (p-run-subr (car (lr-expr (l)),
    p-set-pc (lr->p (*new-l*),
      lr-return-pc (l)))))

THEOREM: lr-apply-subr-preserves-lr-p-proper-statep
let *lr-eval* **be** lr-eval (`'list`, lr-set-pos (l, *pos*), c)
in
(listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
$\wedge$ (car (lr-expr (l)) $\neq$ `'quote`)
$\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-EVAL)
$\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-TEST)
$\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-FETCH)
$\wedge$ (p-psw (lr-eval (`'list`, lr-set-pos (l, *pos*), c)) = `'run`)
$\wedge$ subrp (car (lr-expr (l)))
$\wedge$ lr-p-proper-statep (p-temp-stk (*lr-eval*),
    p-ctrl-stk (*lr-eval*),
    p-data-segment (*lr-eval*),
    table)
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (lr-apply-subr (l, *lr-eval*)) = `'run`)
$\wedge$ lr-proper-formalsp (cdr (p-prog-segment (l)))
$\wedge$ (*pos* = dv (offset (p-pc (l)), `1`)))
$\rightarrow$ lr-p-proper-statep (p-temp-stk (lr-apply-subr (l, *lr-eval*)),
    p-ctrl-stk (*lr-eval*),
    p-data-segment (lr-apply-subr (l, *lr-eval*)),
    table) **endlet**

THEOREM: strip-cdrs-append
strip-cdrs (append (x, y)) = append (strip-cdrs (x), strip-cdrs (y))

THEOREM: strip-cdrs-pairlist
(length (x) $\not<$ length (y))

$\rightarrow$ (strip-cdrs (pairlist (x, y)) = first-n (length (x), y))

THEOREM: lr-good-pointerps-append
lr-good-pointerps (append (x, y), *data-seg*)
$=$ (lr-good-pointerps (x, *data-seg*) $\wedge$ lr-good-pointerps (y, *data-seg*))

THEOREM: lr-good-pointerps-reverse
lr-good-pointerps (reverse (x), *data-seg*) = lr-good-pointerps (x, *data-seg*)

THEOREM: lr-good-pointerps-first-n
(lr-good-pointerps (*list*, *data-seg*) $\wedge$ (length (*list*) $\not<$ n))
$\rightarrow$ lr-good-pointerps (first-n (n, *list*), *data-seg*)

DEFINITION:
all-numberps (*list*)
$=$ **if** listp (*list*) **then** (car (*list*) $\in \mathbf{N}$) $\wedge$ all-numberps (cdr (*list*))
else t endif

THEOREM: all-numberps-strip-cadrs-numberp-cdr-assoc
all-numberps (strip-cadrs (*list*)) $\rightarrow$ (cadr (assoc (x, *list*)) $\in \mathbf{N}$)

THEOREM: all-numberps-strip-cadrs-subr-arity-alist
all-numberps (strip-cadrs (SUBR-ARITY-ALIST))

THEOREM: numberp-arity
arity (x) $\in \mathbf{N}$

THEOREM: strip-cdrs-pair-temps-with-initial-values
strip-cdrs (pair-temps-with-initial-values (*temp-var-dcls*))
$=$ strip-cadrs (*temp-var-dcls*)

THEOREM: lr-good-pointerps-all-undef-addrs
(lr-proper-heapp (*data-seg*)
$\wedge$ lr-proper-p-areasp (*data-seg*)
$\wedge$ all-undef-addrs (*list*))
$\rightarrow$ lr-good-pointerps (*list*, *data-seg*)

THEOREM: all-undef-addrs-strip-cadrs-temp-vars-programs-properp-1
(lr-programs-properp-1 (*programs*, *program-names*, *table*)
$\wedge$ definedp (*name*, *programs*))
$\rightarrow$ all-undef-addrs (strip-cadrs (temp-var-dcls (assoc (*name*, *programs*))))

THEOREM: all-undef-addrs-strip-cadrs-temp-vars-programs-properp
(lr-programs-properp (l, *table*) $\wedge$ definedp (*name*, cdr (p-prog-segment (l))))
$\rightarrow$ all-undef-addrs (strip-cadrs (temp-var-dcls (assoc (*name*,
cdr (p-prog-segment (l))))))

THEOREM: lr-good-pointerps-popn
(lr-good-pointerps (*list*, *data-seg*) $\wedge$ (length (*list*) $\not<$ *n*))
$\rightarrow$ lr-good-pointerps (popn (*n*, *list*), *data-seg*)

THEOREM: lr-p-proper-statep-lr-funcall
((p-psw (lr-eval (**t**, lr-funcall (*l*, *new-l*), *c* $-$ 1)) = `'run`)
$\wedge$ listp (lr-expr (*l*))
$\wedge$ (car (lr-expr (*l*)) $\neq$ `'if`)
$\wedge$ (car (lr-expr (*l*)) $\neq$ `'quote`)
$\wedge$ litatom (car (lr-expr (*l*)))
$\wedge$ ($\neg$ subrp (car (lr-expr (*l*))))
$\wedge$ proper-p-statep (lr->p (*new-l*))
$\wedge$ good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\wedge$ lr-p-proper-statep (p-temp-stk (*l*),
p-ctrl-stk (*l*),
p-data-segment (*l*),
table)
$\wedge$ lr-p-proper-statep (p-temp-stk (*new-l*),
p-ctrl-stk (*new-l*),
p-data-segment (*new-l*),
table)
$\wedge$ (length (p-temp-stk (*new-l*)) $\not<$ arity (car (lr-expr (*l*))))
$\wedge$ lr-proper-formalsp (cdr (p-prog-segment (*l*)))
$\wedge$ lr-programs-properp (*l*, *table*))
$\rightarrow$ lr-p-proper-statep (p-temp-stk (lr-funcall (*l*, *new-l*)),
p-ctrl-stk (lr-funcall (*l*, *new-l*)),
p-data-segment (*new-l*),
table)

THEOREM: length-p-temp-stk-lr-eval-flag-list-alt
(proper-p-statep (lr->p (*l*))
$\wedge$ listp (lr-expr (*l*))
$\wedge$ (car (lr-expr (*l*)) $\neq$ `'if`)
$\wedge$ (car (lr-expr (*l*)) $\neq$ `'quote`)
$\wedge$ ($\neg$ subrp (car (lr-expr (*l*))))
$\wedge$ litatom (car (lr-expr (*l*)))
$\wedge$ good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\wedge$ lr-programs-properp (*l*, *table*)
$\wedge$ lr-proper-formalsp (cdr (p-prog-segment (*l*)))
$\wedge$ (p-psw (lr-eval (`'list`, lr-set-pos (*l*, dv (offset (p-pc (*l*)), `1`)), *c*))
= `'run`))
$\rightarrow$ (length (p-temp-stk (lr-eval (`'list`,
lr-set-pos (*l*, dv (offset (p-pc (*l*)), `1`)),
c)))
= (arity (car (lr-expr (*l*))) + length (p-temp-stk (*l*))))

THEOREM: lr-p-proper-statep-cdr-lr-ctrl-stk

$$
\begin{array}{l}
(\text{listp}\,(\text{lr-expr}\,(l)) \\
\land\quad (\text{car}\,(\text{lr-expr}\,(l)) \neq \texttt{'if}) \\
\land\quad (\neg\ \text{subrp}\,(\text{car}\,(\text{lr-expr}\,(l)))) \\
\land\quad (\text{car}\,(\text{lr-expr}\,(l)) \neq \texttt{'quote}) \\
\land\quad \text{litatom}\,(\text{car}\,(\text{lr-expr}\,(l))) \\
\land\quad \text{lr-p-proper-statep}\,(\textit{temp-stk}, \\
\qquad\qquad \text{p-ctrl-stk}\,(\text{lr-eval}\,(\mathbf{t}, \text{lr-funcall}\,(l, \text{lr-eval}\,(\texttt{'list}, \text{lr-set-pos}\,(l, \textit{pos}), c)), c - 1)), \\
\qquad\qquad \textit{data-segment}, \\
\qquad\qquad \textit{table}) \\
\land\quad \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
\land\quad \text{lr-programs-properp}\,(l,\ \textit{table}) \\
\land\quad \text{proper-p-statep}\,(\text{lr->p}\,(l)) \\
\land\quad (\text{p-psw}\,(\text{lr-eval}\,(\mathbf{t}, \\
\qquad\qquad \text{lr-funcall}\,(l, \text{lr-eval}\,(\texttt{'list}, \text{lr-set-pos}\,(l,\ \textit{pos}),\ c)), \\
\qquad\qquad c - 1)) \\
\qquad = \quad \texttt{'run}) \\
\land\quad (\text{p-psw}\,(\text{lr-eval}\,(\texttt{'list}, \text{lr-set-pos}\,(l,\ \textit{pos}),\ c)) = \texttt{'run}) \\
\land\quad (\textit{pos} = \text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \texttt{1}))) \\
\to\quad \text{lr-p-proper-statep}\,(\textit{temp-stk}, \\
\qquad\qquad \text{cdr}\,(\text{p-ctrl-stk}\,(\text{lr-eval}\,(\mathbf{t}, \text{lr-funcall}\,(l, \text{lr-eval}\,(\texttt{'list}, \text{lr-set-pos}\,(l, \textit{pos}), c)), c - 1))), \\
\qquad\qquad \textit{data-segment}, \\
\qquad\qquad \textit{table})
\end{array}
$$

THEOREM: lr-eval-preserves-lr-p-proper-statep

$$
\begin{array}{l}
(\text{proper-p-statep}\,(\text{lr->p}\,(l)) \\
\land\quad \text{good-posp}\,(\textit{flag},\ \text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
\land\quad (\text{p-psw}\,(\text{lr-eval}\,(\textit{flag},\ l,\ c)) = \texttt{'run}) \\
\land\quad \text{lr-p-proper-statep}\,(\text{p-temp-stk}\,(l), \\
\qquad\qquad \text{p-ctrl-stk}\,(l), \\
\qquad\qquad \text{p-data-segment}\,(l), \\
\qquad\qquad \textit{table})
\end{array}
$$

$\land$ lr-programs-properp (l, *table*)
$\land$ lr-proper-formalsp (cdr (p-prog-segment (l))))
$\rightarrow$ lr-p-proper-statep (p-temp-stk (lr-eval (*flag*, l, c)),
p-ctrl-stk (lr-eval (*flag*, l, c)),
p-data-segment (lr-eval (*flag*, l, c)),
table)

THEOREM: p-plus
p (p, *c1* + *c2*) = p (p (p, *c1*), *c2*)

THEOREM: p-set-pc-lr->p-lr-set-expr
(p-prog-segment (*l1*) = p-prog-segment (*l2*))
$\rightarrow$ (lr->p (lr-set-expr (*l1*, *l2*, *pos*))
= p-set-pc (lr->p (*l1*), lr-p-pc (lr-set-expr (*l1*, *l2*, *pos*))))

EVENT: Disable p-set-pc-lr->p-lr-set-expr.

THEOREM: member-assoc-area-name-cdr-lr-programs-properp
((assoc (area-name (p-pc (l)), cdr (p-prog-segment (l))) $\notin$ p-prog-segment (l))
$\land$ (area-name (p-pc (l)) $\neq$ caar (p-prog-segment (l))))
$\rightarrow$ ($\neg$ lr-programs-properp (l, *table*))

EVENT: Disable member-assoc-area-name-cdr-lr-programs-properp.

THEOREM: not-listp-prog-segment-not-lr-programs-properp
($\neg$ listp (p-prog-segment (l))) $\rightarrow$ ($\neg$ lr-programs-properp (l, *table*))

EVENT: Disable not-listp-prog-segment-not-lr-programs-properp.

THEOREM: unlabel-get-lr-p-pc-program-body-assoc-comp-programs
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*))
$\rightarrow$ (get (lr-p-pc-1 (program-body (p-current-program (l)), offset (p-pc (l))),
program-body (assoc (area-name (p-pc (l)),
comp-programs (p-prog-segment (l)))))
= list (`'dl`,
lr-make-label (offset (lr-p-pc (l))),
nil,
car (comp-body-1 (**t**, lr-expr (l), offset (lr-p-pc (l))))))

THEOREM: car-comp-body-1-litatom
litatom (*body*)
$\rightarrow$ (car (comp-body-1 (**t**, *body*, n)) = list (`'push-local`, *body*))

THEOREM: lr-p-pc-1-append-helper-1

$$
\begin{array}{l}
(\text{listp}\,(\mathit{body}) \land (\text{car}\,(\mathit{body}) = \texttt{'if}) \land (n \not\simeq \texttt{0})) \\
\to \quad (\text{lr-p-pc-1}\,(\mathit{body},\ \text{cons}\,(n,\ \mathit{pos})) \\
\qquad = \quad \textbf{if } n = \texttt{1} \ \textbf{ then } \text{lr-p-pc-1}\,(\text{cadr}\,(\mathit{body}),\ \mathit{pos}) \\
\qquad\qquad \textbf{elseif } n = \texttt{2} \\
\qquad\qquad \textbf{then } \texttt{3} \\
\qquad\qquad\qquad + \quad \text{lr-p-c-size}\,(\texttt{t},\ \text{cadr}\,(\mathit{body})) \\
\qquad\qquad\qquad + \quad \text{lr-p-pc-1}\,(\text{caddr}\,(\mathit{body}),\ \mathit{pos}) \\
\qquad\qquad \textbf{else } \text{lr-p-c-size}\,(\texttt{t},\ \text{cadr}\,(\mathit{body})) \\
\qquad\qquad\quad + \quad \text{lr-p-c-size}\,(\texttt{t},\ \text{caddr}\,(\mathit{body})) \\
\qquad\qquad\quad + \quad \text{lr-p-pc-1}\,(\text{cadddr}\,(\mathit{body}),\ \mathit{pos}) \\
\qquad\qquad\quad + \quad \texttt{4}\ \textbf{endif})
\end{array}
$$

THEOREM: lr-p-pc-1-append-helper-2

$$
\begin{array}{l}
(\text{listp}\,(\mathit{body}) \land (\text{car}\,(\mathit{body}) = \textsc{s-temp-eval})) \\
\to \quad (\text{lr-p-pc-1}\,(\mathit{body},\ \text{cons}\,(\texttt{1},\ \mathit{pos})) = \text{lr-p-pc-1}\,(\text{cadr}\,(\mathit{body}),\ \mathit{pos}))
\end{array}
$$

THEOREM: lr-p-pc-1-append-helper-3

$$
\begin{array}{l}
(\text{listp}\,(\mathit{body}) \land (\text{car}\,(\mathit{body}) = \textsc{s-temp-test})) \\
\to \quad (\text{lr-p-pc-1}\,(\mathit{body},\ \text{cons}\,(\texttt{1},\ \mathit{pos})) = (\text{lr-p-pc-1}\,(\text{cadr}\,(\mathit{body}),\ \mathit{pos}) + \texttt{4}))
\end{array}
$$

THEOREM: lr-p-pc-1-append-helper-4

$$
\begin{array}{l}
(\text{listp}\,(\mathit{body}) \\
\ \land \quad (\text{car}\,(\mathit{body}) \neq \textsc{s-temp-fetch}) \\
\ \land \quad (\text{car}\,(\mathit{body}) \neq \textsc{s-temp-eval}) \\
\ \land \quad (\text{car}\,(\mathit{body}) \neq \textsc{s-temp-test}) \\
\ \land \quad (\text{car}\,(\mathit{body}) \neq \texttt{'quote}) \\
\ \land \quad (\text{car}\,(\mathit{body}) \neq \texttt{'if}) \\
\ \land \quad (n \not\simeq \texttt{0})) \\
\to \quad (\text{lr-p-pc-1}\,(\mathit{body},\ \text{cons}\,(n,\ \mathit{pos})) \\
\qquad = \quad (\text{lr-p-c-size-list}\,(n - 1,\ \mathit{body}) + \text{lr-p-pc-1}\,(\text{get}\,(n,\ \mathit{body}),\ \mathit{pos})))
\end{array}
$$

THEOREM: lr-p-pc-1-append

$$
\begin{array}{l}
(\text{good-posp1}\,(\mathit{pos1},\ \mathit{body}) \\
\ \land \quad \text{lr-proper-exprp}\,(\texttt{t},\ \mathit{body},\ \mathit{pnames},\ \mathit{formals},\ \mathit{temps},\ \mathit{table})) \\
\to \quad (\text{lr-p-pc-1}\,(\mathit{body},\ \text{append}\,(\mathit{pos1},\ \mathit{pos2})) \\
\qquad = \quad (\text{lr-p-pc-1}\,(\mathit{body},\ \mathit{pos1}) + \text{lr-p-pc-1}\,(\text{cur-expr}\,(\mathit{pos1},\ \mathit{body}),\ \mathit{pos2})))
\end{array}
$$

THEOREM: append-butlast-list-car-last

$$\text{listp}\,(x) \to (\text{append}\,(\text{butlast}\,(x),\ \text{list}\,(\text{car}\,(\text{last}\,(x)))) = \text{plist}\,(x))$$

THEOREM: listp-plist-car

$$\text{listp}\,(x) \to (\text{car}\,(\text{plist}\,(x)) = \text{car}\,(x))$$

THEOREM: lr-p-pc-1-plist

$$\text{lr-p-pc-1}\,(\mathit{body},\ \text{plist}\,(\mathit{pos})) = \text{lr-p-pc-1}\,(\mathit{body},\ \mathit{pos})$$

THEOREM: lr-p-pc-1-listp-offset
$$
\begin{aligned}
&(\text{listp}\,(\mathit{pos})\\
&\ \wedge\ \ \text{good-posp1}\,(\text{butlast}\,(\mathit{pos}),\ \mathit{body})\\
&\ \wedge\ \ \text{lr-proper-exprp}\,(\mathbf{t},\ \mathit{body},\ \mathit{pnames},\ \mathit{formals},\ \mathit{temps},\ \mathit{table}))\\
&\rightarrow\ \ (\text{lr-p-pc-1}\,(\mathit{body},\ \mathit{pos})\\
&\qquad =\ \ (\text{lr-p-pc-1}\,(\mathit{body},\ \text{butlast}\,(\mathit{pos}))\\
&\qquad\qquad +\ \ \text{lr-p-pc-1}\,(\text{cur-expr}\,(\text{butlast}\,(\mathit{pos}),\ \mathit{body}),\\
&\qquad\qquad\qquad\qquad \text{list}\,(\text{car}\,(\text{last}\,(\mathit{pos}))))))
\end{aligned}
$$

THEOREM: lr-p-pc-1-nil
$$\text{lr-p-pc-1}\,(\mathit{body},\ \mathbf{nil}) = \mathtt{0}$$

THEOREM: lr-p-pc-1-nx-helper
$$
\begin{aligned}
&(\text{listp}\,(\mathit{expr})\\
&\ \wedge\ \ (\text{car}\,(\mathit{expr}) \neq \text{S-TEMP-FETCH})\\
&\ \wedge\ \ (\text{car}\,(\mathit{expr}) \neq \text{S-TEMP-EVAL})\\
&\ \wedge\ \ (\text{car}\,(\mathit{expr}) \neq \text{S-TEMP-TEST})\\
&\ \wedge\ \ (\text{car}\,(\mathit{expr}) \neq \texttt{'quote})\\
&\ \wedge\ \ (n \not\simeq \mathtt{0})\\
&\ \wedge\ \ (n < \text{length}\,(\mathit{expr})))\\
&\rightarrow\ \ (\text{lr-p-pc-1}\,(\mathit{expr},\ \text{list}\,(n))\\
&\qquad =\ \ \mathbf{if}\ \text{car}\,(\mathit{expr}) = \texttt{'if}\\
&\qquad\qquad \mathbf{then\ case\ on}\ n\text{:}\\
&\qquad\qquad\qquad \mathbf{case} = \mathtt{1}\\
&\qquad\qquad\qquad \mathbf{then}\ \mathtt{0}\\
&\qquad\qquad\qquad \mathbf{case} = \mathtt{2}\\
&\qquad\qquad\qquad\ \ \mathbf{then}\ \mathtt{3} + \text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(\mathit{expr}))\\
&\qquad\qquad\qquad \mathbf{otherwise}\ \text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(\mathit{expr}))\\
&\qquad\qquad\qquad\qquad\qquad +\ \ \text{lr-p-c-size}\,(\mathbf{t},\ \text{caddr}\,(\mathit{expr}))\\
&\qquad\qquad\qquad\qquad\qquad +\ \ \mathtt{4}\ \mathbf{endcase}\\
&\qquad\qquad \mathbf{else}\ \text{lr-p-c-size-list}\,(n - 1,\ \mathit{expr})\ \mathbf{endif})
\end{aligned}
$$

THEOREM: lr-p-pc-1-nx
$$
\begin{aligned}
&(\text{lr-programs-properp}\,(l,\ \mathit{table})\\
&\ \wedge\ \ \text{good-posp}\,(\texttt{'list},\ \text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l)))\\
&\ \wedge\ \ \text{listp}\,(\text{offset}\,(\text{p-pc}\,(l)))\\
&\ \wedge\ \ \text{listp}\,(\text{lr-expr-list}\,(l))\\
&\ \wedge\ \ (\text{car}\,(\text{cur-expr}\,(\text{butlast}\,(\text{offset}\,(\text{p-pc}\,(l))),\\
&\qquad\qquad\qquad \text{program-body}\,(\text{p-current-program}\,(l))))\\
&\qquad \neq\ \ \texttt{'if}))\\
&\rightarrow\ \ (\text{lr-p-pc-1}\,(\text{program-body}\,(\text{p-current-program}\,(l)),\ \text{nx}\,(\text{offset}\,(\text{p-pc}\,(l))))\\
&\qquad =\ \ (\text{lr-p-pc-1}\,(\text{program-body}\,(\text{p-current-program}\,(l)),\ \text{offset}\,(\text{p-pc}\,(l)))\\
&\qquad\qquad +\ \ \text{lr-p-c-size}\,(\mathbf{t},\ \text{lr-expr}\,(l))))
\end{aligned}
$$

EVENT: Disable lr-p-pc-1-listp-offset.

THEOREM: lr-p-pc-1-dv-1-car-lr-expr-if

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{car}\,(\text{lr-expr}\,(l)) = \texttt{'if})} \\
\land & \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
\land & \text{lr-programs-properp}\,(l,\ table) \\
\land & (body = \text{program-body}\,(\text{p-current-program}\,(l)))) \\
\rightarrow & (\text{lr-p-pc-1}\,(body,\ \text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \texttt{1})) \\
& \quad = \ \text{lr-p-pc-1}\,(body,\ \text{offset}\,(\text{p-pc}\,(l))))
\end{array}
$$

THEOREM: lr-p-pc-1-dv-2-car-lr-expr-if

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\text{lr-expr}\,(l))} \\
\land & (\text{car}\,(\text{lr-expr}\,(l)) = \texttt{'if}) \\
\land & \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
\land & \text{lr-programs-properp}\,(l,\ table)) \\
\rightarrow & (\text{lr-p-pc-1}\,(\text{program-body}\,(\text{p-current-program}\,(l)),\ \text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \texttt{2})) \\
& \quad = \ (\texttt{3} \\
& \qquad\quad + \ \text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(\text{lr-expr}\,(l))) \\
& \qquad\quad + \ \text{lr-p-pc-1}\,(\text{program-body}\,(\text{p-current-program}\,(l)), \\
& \qquad\qquad\qquad\quad \text{offset}\,(\text{p-pc}\,(l)))))
\end{array}
$$

THEOREM: lr-p-pc-1-dv-3-car-lr-expr-if

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\text{lr-expr}\,(l))} \\
\land & (\text{car}\,(\text{lr-expr}\,(l)) = \texttt{'if}) \\
\land & \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
\land & \text{lr-programs-properp}\,(l,\ table)) \\
\rightarrow & (\text{lr-p-pc-1}\,(\text{program-body}\,(\text{p-current-program}\,(l)),\ \text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \texttt{3})) \\
& \quad = \ (\text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(\text{lr-expr}\,(l))) \\
& \qquad\quad + \ \texttt{3} \\
& \qquad\quad + \ \text{lr-p-c-size}\,(\mathbf{t},\ \text{caddr}\,(\text{lr-expr}\,(l))) \\
& \qquad\quad + \ \texttt{1} \\
& \qquad\quad + \ \text{lr-p-pc-1}\,(\text{program-body}\,(\text{p-current-program}\,(l)), \\
& \qquad\qquad\qquad\quad \text{offset}\,(\text{p-pc}\,(l)))))
\end{array}
$$

THEOREM: lr-p-c-size-not-1-car-if

$$(\text{listp}\,(expr) \land (\text{car}\,(expr) = \texttt{'if})) \rightarrow (\text{lr-p-c-size}\,(\mathbf{t},\ expr) \neq \texttt{1})$$

THEOREM: lr-p-c-size-ge-plus-2-size-cadr-car-if

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(expr) \land (\text{car}\,(expr) = \texttt{'if}))} \\
\rightarrow & ((1 + (1 + \text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(expr)))) < \text{lr-p-c-size}\,(\mathbf{t},\ expr))
\end{array}
$$

THEOREM: get-comp-if-helper-helper

$$
\begin{array}{ll}
\multicolumn{2}{l}{((n \not< (x + y + \texttt{3})) \land (n \neq (x + y + \texttt{3})))} \\
\rightarrow & (\text{get}\,(n - (x + \texttt{3} + y),\ \text{cons}\,(w,\ l3)) \\
& \quad = \ \text{get}\,(n - (x + y + \texttt{4}),\ l3))
\end{array}
$$

THEOREM: get-comp-if-helper

```
get (n, append (l1, append (list (x, y, z), append (l2, cons (w, l3)))))
=   if n < length (l1) then get (n, l1)
    elseif n < (length (l1) + 3)
    then get (n − length (l1), list (x, y, z))
    elseif n < (length (l1) + length (l2) + 3)
    then get (n − (length (l1) + 3), l2)
    elseif n = (length (l1) + length (l2) + 3) then w
    else get (n − (length (l1) + length (l2) + 4), l3) endif
```

THEOREM: get-comp-if

```
get (n, comp-if (test-instrs, then-instrs, else-instrs, m))
=   if n < length (test-instrs) then get (n, test-instrs)
    elseif n < (length (test-instrs) + 3)
    then get (n − length (test-instrs),
              list (identity (list ('push-constant, LR-F-ADDR)),
                   '(eq),
                   list ('test-bool-and-jump,
                        't,
                        lr-make-label (m
                                      +   4
                                      +   length (test-instrs)
                                      +   length (then-instrs)))))
    elseif n < (length (test-instrs) + length (then-instrs) + 3)
    then get (n − (length (test-instrs) + 3), then-instrs)
    elseif n = (length (test-instrs) + length (then-instrs) + 3)
    then list ('jump,
              lr-make-label (m
                            +   4
                            +   length (test-instrs)
                            +   length (then-instrs)
                            +   length (else-instrs)))
    else get (n − (length (test-instrs)
                  +   length (then-instrs)
                  +   4),
             else-instrs) endif
```

DEFINITION:

```
p-final-pc (flag, l, n)
=   if flag = 'list
    then add-addr (lr-p-pc (l), n + lr-p-c-size ('list, lr-expr-list (l)))
    else add-addr (lr-p-pc (l), n + lr-p-c-size (flag, lr-expr (l))) endif
```

EVENT: Disable p-final-pc.

THEOREM: proper-p-statep-p-set-pc

```
(proper-p-statep (p) ∧ (area-name (p-pc (p)) = area-name (pc)))
→   (proper-p-statep (p-set-pc (p, pc)) = p-objectp-type ('pc, pc, p))
```

THEOREM: proper-p-statep-p-set-pc-equal-p-set-pc

```
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 ∧   lr-programs-properp (l, table)
 ∧   (p (lr->p (l), p-clock1 (t, l, c))
        =   p-set-pc (lr->p (lr-eval (t, l, c)),
                      tag ('pc,
                           cons (area-name (p-pc (l)),
                                 lr-p-pc-1 (program-body (p-current-program (l)),
                                            offset (p-pc (l)))
                                 +   lr-p-c-size (t, lr-expr (l))))))
 ∧   proper-p-statep (lr->p (lr-eval (t, l, c))))
→   proper-p-statep (p (lr->p (l), p-clock1 (t, l, c)))
```

THEOREM: lr-eval-p-pc-equivalence-helper-1

```
p (lr->p (l1), p-clock1 (flag1, l2, c1) + p-clock1 (flag2, l3, c2))
=   p (p (lr->p (l1), p-clock1 (flag1, l2, c1)), p-clock1 (flag2, l3, c2))
```

THEOREM: lr-eval-p-pc-equivalence-helper-1-5

```
(listp (offset (p-pc (l))) ∧ listp (lr-expr-list (l)))
→   (p-final-pc ('list,
                 lr-set-expr (lr-eval (t, l, c), l, nx (offset (p-pc (l)))),
                 0)
        =   tag ('pc,
                 cons (area-name (p-pc (l)),
                       lr-p-pc-1 (program-body (p-current-program (l)),
                                  nx (offset (p-pc (l))))
                       +   lr-p-c-size-list (length (lr-expr-list (l)) − 1,
                                             lr-expr-list (l)))))
```

THEOREM: lr-eval-p-pc-equivalence-helper-2

```
((p-psw (l) = 'run)
 ∧   listp (offset (p-pc (l)))
 ∧   (¬ listp (lr-expr-list (l))))
→   (p-set-pc (lr->p (l),
               tag ('pc,
                    cons (area-name (p-pc (l)),
                          lr-p-pc-1 (program-body (p-current-program (l)),
                                     offset (p-pc (l))))))
        =   p (lr->p (l), 0))
```

THEOREM: lr-eval-p-pc-equivalence-helper-3

```
((p-psw (l) = 'run)
```

$\wedge$ (*flag* $\neq$ `'list`)
$\wedge$ litatom (lr-expr (l))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (length (p-temp-stk (l)) $<$ p-max-temp-stk-size (l)))
$\rightarrow$ (p-set-pc (lr->p (lr-set-tstk (l,
cons (cdr (assoc (lr-expr (l),
bindings (car (p-ctrl-stk (l))))),
p-temp-stk (l)))),
p-final-pc (*flag*, l, `0`))
= p (lr->p (l), `1`))

THEOREM: lr->p-lr-set-pos-dv-1-car-lr-expr-if
((car (lr-expr (l)) = `'if`)
$\wedge$ listp (lr-expr (l))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*))
$\rightarrow$ (lr->p (lr-set-pos (l, dv (offset (p-pc (l)), `1`))) = lr->p (l))

THEOREM: lr-p-proper-statep-listp-p-temp-stk-type-car-addr
(lr-p-proper-statep (*temp-stk*, *ctrl-stk*, *data-seg*, *table*) $\wedge$ listp (*temp-stk*))
$\rightarrow$ (type (car (*temp-stk*)) = `'addr`)

THEOREM: proper-p-statep-lessp-length-p-temp-stk-max-temp-stk-size
proper-p-statep (lr->p (l))
$\rightarrow$ (p-max-temp-stk-size (l) $\not<$ length (p-temp-stk (l)))

THEOREM: length-p-temp-stk-lr-eval-lr-set-pos
(proper-p-statep (lr->p (l))
$\wedge$ good-posp1 (*pos*, program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ lr-proper-formalsp (cdr (p-prog-segment (l)))
$\wedge$ (p-psw (lr-eval (**t**, lr-set-pos (l, *pos*), c)) = `'run`))
$\rightarrow$ (length (p-temp-stk (lr-eval (**t**, lr-set-pos (l, *pos*), c)))
= (`1` + length (p-temp-stk (l))))

THEOREM: not-lessp-length-proper-p-statep-lr-eval-lr-set-pos
(good-posp1 (*pos*, program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ (length (p-temp-stk (l)) $\not<$ (p-max-temp-stk-size (l) $-$ 1))
$\wedge$ lr-proper-formalsp (cdr (p-prog-segment (l))))
$\rightarrow$ (p-psw (lr-if-ok (lr-eval (**t**, lr-set-pos (l, *pos*), c))) $\neq$ `'run`)

THEOREM: lr-pop-tstk-lr-if-ok
(p-psw (lr-if-ok (l)) = `'run`)
$\rightarrow$ (lr-pop-tstk (lr-if-ok (l)) = lr-pop-tstk (l))

THEOREM: add-addr-p-final-pc
$(\text{add-addr}\,(\text{p-final-pc}\,(\mathit{flag},\ l,\ n),\ 1 + m)$
$\ \ = \ \ \text{add-addr}\,(\text{p-final-pc}\,(\mathit{flag},\ l,\ 1 + n),\ m))$
$\wedge \ \ (\text{add-addr}\,(\text{p-final-pc}\,(\mathit{flag},\ l,\ n),\ \texttt{0}) = \text{p-final-pc}\,(\mathit{flag},\ l,\ n))$

THEOREM: lessp-3-lr-p-c-size-car-if
$(\text{listp}\,(\mathit{expr}) \wedge (\text{car}\,(\mathit{expr}) = \texttt{'if})) \rightarrow (\texttt{3} < \text{lr-p-c-size}\,(\texttt{t},\ \mathit{expr}))$

THEOREM: comp-body-1-car-expr-if
$$
\begin{array}{l}
((\text{car}\,(\mathit{expr}) = \texttt{'if}) \wedge \text{listp}\,(\mathit{expr})) \\
\rightarrow \ \ (\text{comp-body-1}\,(\texttt{t},\ \mathit{expr},\ n) \\
\qquad = \ \ \text{comp-if}\,(\text{comp-body-1}\,(\texttt{t},\ \text{cadr}\,(\mathit{expr}),\ n), \\
\qquad\qquad\qquad \text{comp-body-1}\,(\texttt{t}, \\
\qquad\qquad\qquad\qquad\qquad \text{caddr}\,(\mathit{expr}), \\
\qquad\qquad\qquad\qquad\qquad n + \texttt{3} + \text{lr-p-c-size}\,(\texttt{t},\ \text{cadr}\,(\mathit{expr}))), \\
\qquad\qquad\qquad \text{comp-body-1}\,(\texttt{t}, \\
\qquad\qquad\qquad\qquad\qquad \text{cadddr}\,(\mathit{expr}), \\
\qquad\qquad\qquad\qquad\qquad n \\
\qquad\qquad\qquad\qquad\qquad + \ \ \texttt{4} \\
\qquad\qquad\qquad\qquad\qquad + \ \ \text{lr-p-c-size}\,(\texttt{t},\ \text{cadr}\,(\mathit{expr})) \\
\qquad\qquad\qquad\qquad\qquad + \ \ \text{lr-p-c-size}\,(\texttt{t},\ \text{caddr}\,(\mathit{expr}))), \\
\qquad\qquad\qquad n))
\end{array}
$$

THEOREM: get-lr-p-c-size-lessp-restn-lr-p-pc-1-comp-body-1
$$
\begin{array}{l}
(\text{good-posp1}\,(\mathit{pos},\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
\ \wedge \ \ \text{lr-programs-properp}\,(l,\ \mathit{table}) \\
\ \wedge \ \ \text{listp}\,(\text{lr-expr}\,(l)) \\
\ \wedge \ \ (\text{car}\,(\text{lr-expr}\,(l)) = \texttt{'if}) \\
\ \wedge \ \ (m < \texttt{3}) \\
\ \wedge \ \ (\mathit{name} = \text{area-name}\,(\text{p-pc}\,(l))) \\
\ \wedge \ \ (\mathit{pos} = \text{offset}\,(\text{p-pc}\,(l)))) \\
\rightarrow \ \ (\text{unlabel}\,(\text{get}\,(\text{offset}\,(\text{p-final-pc}\,(\texttt{t},\ \text{lr-set-pos}\,(l,\ \text{dv}\,(\mathit{pos},\ \texttt{1})),\ m)), \\
\qquad\qquad\qquad \text{program-body}\,(\text{assoc}\,(\mathit{name},\ \text{comp-programs}\,(\text{p-prog-segment}\,(l)))))) \\
\qquad = \ \ \text{get}\,(m, \\
\qquad\qquad\quad \text{list}\,(\text{list}\,(\texttt{'push-constant},\ \textsc{lr-f-addr}), \\
\qquad\qquad\qquad \texttt{'(eq)}, \\
\qquad\qquad\qquad \text{list}\,(\texttt{'test-bool-and-jump}, \\
\qquad\qquad\qquad\quad \texttt{'t}, \\
\qquad\qquad\qquad\quad \text{lr-make-label}\,(\text{lr-p-pc-1}\,(\text{program-body}\,(\text{p-current-program}\,(l)), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{pos}) \\
\qquad\qquad\qquad\qquad\qquad\quad + \ \ \texttt{4} \\
\qquad\qquad\qquad\qquad\qquad\quad + \ \ \text{lr-p-c-size}\,(\texttt{t}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{cadr}\,(\text{lr-expr}\,(l))) \\
\qquad\qquad\qquad\qquad\qquad\quad + \ \ \text{lr-p-c-size}\,(\texttt{t}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{caddr}\,(\text{lr-expr}\,(l)))))))
\end{array}
$$

THEOREM: area-name-p-final-pc
$\text{area-name}\,(\text{p-final-pc}\,(\mathit{flag},\ l,\ n)) = \text{area-name}\,(\text{p-pc}\,(l))$

THEOREM: lr-eval-p-pc-equivalence-helper-4-helper-1

$$
\begin{array}{l}
\textbf{let}\ \ \mathit{test}\ \ \textbf{be}\ \ \text{lr-eval}\,(\mathbf{t},\ \text{lr-set-pos}\,(l,\ \text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ 1)),\ c) \\
\textbf{in} \\
((\text{p-psw}\,(l) = \texttt{'run}) \\
\ \wedge\ \ (c \not\simeq 0) \\
\ \wedge\ \ \text{listp}\,(\text{lr-expr}\,(l)) \\
\ \wedge\ \ (\text{car}\,(\text{lr-expr}\,(l)) = \texttt{'if}) \\
\ \wedge\ \ (\text{p-psw}\,(\text{lr-if-ok}\,(\mathit{test})) = \texttt{'run}) \\
\ \wedge\ \ (\text{car}\,(\text{p-temp-stk}\,(\mathit{test})) \neq \text{LR-F-ADDR}) \\
\ \wedge\ \ \text{proper-p-statep}\,(\text{lr->p}\,(l)) \\
\ \wedge\ \ \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
\ \wedge\ \ \text{lr-programs-properp}\,(l,\ \mathit{table}) \\
\ \wedge\ \ \text{lr-p-proper-statep}\,(\text{p-temp-stk}\,(l), \\
\qquad\qquad\qquad\qquad\qquad \text{p-ctrl-stk}\,(l), \\
\qquad\qquad\qquad\qquad\qquad \text{p-data-segment}\,(l), \\
\qquad\qquad\qquad\qquad\qquad \mathit{table}) \\
\ \wedge\ \ \text{lr-proper-formalsp}\,(\text{cdr}\,(\text{p-prog-segment}\,(l))) \\
\ \wedge\ \ (\text{p-psw}\,(\text{lr-eval}\,(\mathbf{t}, \\
\qquad\qquad\qquad\quad \text{lr-set-expr}\,(\text{lr-pop-tstk}\,(\mathit{test}), \\
\qquad\qquad\qquad\qquad\qquad\qquad l, \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ 2)), \\
\qquad\qquad\qquad\quad c)) \\
\qquad = \ \ \texttt{'run})) \\
\to\ \ (\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(\mathit{test}), \\
\qquad\qquad\qquad \text{p-final-pc}\,(\mathbf{t}, \\
\qquad\qquad\qquad\qquad\qquad \text{lr-set-pos}\,(l,\ \text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ 1)), \\
\qquad\qquad\qquad\qquad\qquad 0)), \\
\qquad\quad 3) \\
\qquad = \ \ \text{p-set-pc}\,(\text{lr->p}\,(\text{lr-pop-tstk}\,(\mathit{test})), \\
\qquad\qquad\qquad \text{tag}\,(\texttt{'pc}, \\
\qquad\qquad\qquad\qquad \text{cons}\,(\text{area-name}\,(\text{p-pc}\,(l)), \\
\qquad\qquad\qquad\qquad\qquad 3 \\
\qquad\qquad\qquad\qquad\qquad +\ \ \text{lr-p-c-size}\,(\mathbf{t}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{cadr}\,(\text{lr-expr}\,(l))) \\
\qquad\qquad\qquad\qquad\qquad +\ \ \text{lr-p-pc-1}\,(\text{program-body}\,(\text{p-current-program}\,(l)), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{offset}\,(\text{p-pc}\,(l))))))))\ \ \textbf{endlet}
\end{array}
$$

THEOREM: lessp-plus-lr-p-c-size-cadr-caddr-3-car-if

$$
\begin{array}{l}
(\text{listp}\,(\mathit{expr}) \wedge (\text{car}\,(\mathit{expr}) = \texttt{'if})) \\
\to\ \ ((\text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(\mathit{expr})) + \text{lr-p-c-size}\,(\mathbf{t},\ \text{caddr}\,(\mathit{expr}))) \\
\qquad <\ \ (((\text{lr-p-c-size}\,(\mathbf{t},\ \mathit{expr}) - 1) - 1) - 1))
\end{array}
$$

THEOREM: get-plus-lr-p-c-size-cadr-caddr-4-comp-body-cur-expr

$((\mathit{size1} = \text{length}\,(\mathit{test})) \land (\mathit{size2} = \text{length}\,(\mathit{then})))$
$\rightarrow\ (\text{get}\,(3 + \mathit{size1} + \mathit{size2},\ \text{comp-if}\,(\mathit{test},\ \mathit{then},\ \mathit{else},\ n))$
$\quad = \text{list}\,(\texttt{'jump},$
$\qquad \text{lr-make-label}\,(4 + n + \text{length}\,(\mathit{test}) + \text{length}\,(\mathit{then}) + \text{length}\,(\mathit{else}))))$

THEOREM: cur-expr-add1-opener

$\text{cur-expr}\,(\text{cons}\,(1 + n,\ \mathit{pos}),\ \mathit{body}) = \text{cur-expr}\,(\mathit{pos},\ \text{get}\,(n,\ \text{cdr}\,(\mathit{body})))$

THEOREM: lr-p-pc-1-car-expr-if-2

$(\text{listp}\,(\mathit{expr}) \land (\text{car}\,(\mathit{expr}) = \texttt{'if}))$
$\rightarrow\ (\text{lr-p-pc-1}\,(\mathit{expr},\ \texttt{'(2)}) = (3 + \text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(\mathit{expr}))))$

THEOREM: get-plus-lr-p-pc-1-lr-pc-size-cadr-assoc-comp-body-if-4

$(\text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(\mathit{l1})),\ \text{program-body}\,(\text{p-current-program}\,(\mathit{l1})))$
$\land\ \text{lr-programs-properp}\,(\mathit{l1},\ \mathit{table})$
$\land\ \text{listp}\,(\text{lr-expr}\,(\mathit{l1}))$
$\land\ (\text{car}\,(\text{lr-expr}\,(\mathit{l1})) = \texttt{'if})$
$\land\ (\mathit{pos} = \text{dv}\,(\text{offset}\,(\text{p-pc}\,(\mathit{l1})),\ 2))$
$\land\ (\text{area-name}\,(\text{p-pc}\,(\mathit{l2})) = \text{area-name}\,(\text{p-pc}\,(\mathit{l1})))$
$\land\ (\text{p-prog-segment}\,(\mathit{l2}) = \text{p-prog-segment}\,(\mathit{l1})))$
$\rightarrow\ (\text{get}\,(\text{offset}\,(\text{p-final-pc}\,(\mathbf{t},\ \text{lr-set-expr}\,(\mathit{l2},\ \mathit{l1},\ \mathit{pos}),\ 0)),$
$\qquad \text{program-body}\,(\text{assoc}\,(\text{area-name}\,(\text{p-pc}\,(\mathit{l1})),$
$\qquad\qquad \text{comp-programs}\,(\text{p-prog-segment}\,(\mathit{l1})))))$
$\quad = \text{list}\,(\texttt{'dl},$
$\qquad \text{lr-make-label}\,(\text{offset}\,(\text{p-final-pc}\,(\mathbf{t},\ \text{lr-set-expr}\,(\mathit{l2},\ \mathit{l1},\ \mathit{pos}),\ 0))),$
$\qquad \mathbf{nil},$
$\qquad \text{list}\,(\texttt{'jump},$
$\qquad\quad \text{lr-make-label}\,(4 + \text{lr-p-pc-1}\,(\text{program-body}\,(\text{p-current-program}\,(\mathit{l1})),\ \text{offset}\,(\text{p-pc}\,(\mathit{l1})))$
$\qquad\qquad + \text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(\text{lr-expr}\,(\mathit{l1})))$
$\qquad\qquad + \text{lr-p-c-size}\,(\mathbf{t},\ \text{caddr}\,(\text{lr-expr}\,(\mathit{l1})))$
$\qquad\qquad + \text{lr-p-c-size}\,(\mathbf{t},\ \text{cadddr}\,(\text{lr-expr}\,(\mathit{l1})))))))$

THEOREM: find-label-lr-make-label-label-instrs

$((m \not< n) \land (m \in \mathbf{N}) \land (m < (n + \text{length}(\mathit{instrs}))))$
$\rightarrow\quad (\text{find-label}(\text{lr-make-label}(m),\ \text{label-instrs}(\mathit{instrs},\ n)) = (m - n))$

THEOREM: find-label-past-else-lr-expr-car-if

$$
\begin{array}{ll}
& (\text{listp}(\text{lr-expr}(l)) \\
\land & (\text{car}(\text{lr-expr}(l)) = \texttt{'if}) \\
\land & (\mathit{expr} = \text{lr-expr}(l)) \\
\land & (\mathit{body} = \text{program-body}(\text{p-current-program}(l))) \\
\land & (\mathit{pos} = \text{offset}(\text{p-pc}(l))) \\
\land & \text{lr-programs-properp}(l,\ \mathit{table}) \\
\land & \text{good-posp1}(\mathit{pos},\ \mathit{body})) \\
\rightarrow & (\text{find-label}(\text{lr-make-label}(1 + (1 + (1 + (1 + (\text{lr-p-c-size}(\mathbf{t},\ \text{cadr}(\mathit{expr})) \\
& \qquad + \text{lr-p-c-size}(\mathbf{t},\ \text{caddr}(\mathit{expr})) \\
& \qquad + \text{lr-p-c-size}(\mathbf{t},\ \text{cadddr}(\mathit{expr})) \\
& \qquad + \text{lr-p-pc-1}(\mathit{body},\ \mathit{pos})))))), \\
& \qquad \text{program-body}(\text{assoc}(\text{area-name}(\text{p-pc}(l)), \\
& \qquad\qquad \text{comp-programs}(\text{p-prog-segment}(l))))) \\
& \quad = (1 + (1 + (1 + (1 + (\text{lr-p-c-size}(\mathbf{t},\ \text{cadr}(\mathit{expr})) \\
& \qquad + \text{lr-p-c-size}(\mathbf{t},\ \text{caddr}(\mathit{expr})) \\
& \qquad + \text{lr-p-c-size}(\mathbf{t},\ \text{cadddr}(\mathit{expr})) \\
& \qquad + \text{lr-p-pc-1}(\mathit{body},\ \mathit{pos})))))))
\end{array}
$$

THEOREM: lr-eval-p-pc-equivalence-helper-4-helper-2

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textbf{let } \mathit{test} \textbf{ be } \text{lr-eval}(\mathbf{t},\ \text{lr-set-pos}(l,\ \text{dv}(\text{offset}(\text{p-pc}(l)),\ \texttt{1})),\ c)} \\
\multicolumn{2}{l}{\textbf{in}} \\
\multicolumn{2}{l}{\textbf{let } \mathit{then} \textbf{ be } \text{lr-set-expr}(\text{lr-pop-tstk}(\mathit{test}),\ l,\ \text{dv}(\text{offset}(\text{p-pc}(l)),\ \texttt{2}))} \\
\multicolumn{2}{l}{\textbf{in}} \\
& ((\text{p-psw}(l) = \texttt{'run}) \\
\land & (\mathit{flag} \neq \texttt{'list}) \\
\land & (c \neq \texttt{0}) \\
\land & (c \in \mathbf{N}) \\
\land & \text{listp}(\text{lr-expr}(l)) \\
\land & (\text{car}(\text{lr-expr}(l)) = \texttt{'if}) \\
\land & (\text{p-psw}(\text{lr-if-ok}(\mathit{test})) = \texttt{'run}) \\
\land & (\text{car}(\text{p-temp-stk}(\mathit{test})) \neq \text{LR-F-ADDR}) \\
\land & \text{proper-p-statep}(\text{lr->p}(l)) \\
\land & \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \\
& \qquad \text{program-body}(\text{p-current-program}(l))) \\
\land & \text{lr-programs-properp}(l,\ \mathit{table}) \\
\land & \text{lr-p-proper-statep}(\text{p-temp-stk}(l),
\end{array}
$$

```
                              p-ctrl-stk (l),
                              p-data-segment (l),
                              table)
 ∧   lr-proper-formalsp (cdr (p-prog-segment (l)))
 ∧   (p-psw (lr-eval (t, then, c)) = 'run))
→    (p (p-set-pc (lr->p (lr-eval (t, then, c)),
                  p-final-pc (t, then, 0)),
        1)
       =   p-set-pc (lr->p (lr-eval (t, then, c)),
                     p-final-pc (flag, l, 0))) endlet endlet
```

THEOREM: lr-eval-p-pc-equivalence-helper-4

```
let test  be  lr-eval (t, lr-set-pos (l, pos), c),
    cadr-size  be  lr-p-c-size (t, cadr (lr-expr (l))),
    lr-p-pc-1  be  lr-p-pc-1 (program-body (p-current-program (l)),
                              offset (p-pc (l))),
    then  be  lr-set-expr (lr-pop-tstk (lr-eval (t, lr-set-pos (l, pos), c)),
                           l,
                           dv (offset (p-pc (l)), 2))
in
((p-psw (l) = 'run)
 ∧   (flag ≠ 'list)
 ∧   (c ≠ 0)
 ∧   (c ∈ N)
 ∧   listp (lr-expr (l))
 ∧   (car (lr-expr (l)) = 'if)
 ∧   (p-psw (lr-if-ok (test)) = 'run)
 ∧   (car (p-temp-stk (test)) ≠ LR-F-ADDR)
 ∧   (p (lr->p (l), p-clock1 (t, lr-set-pos (l, pos), c))
       =   p-set-pc (lr->p (lr-eval (t, lr-set-pos (l, pos), c)),
                     p-final-pc (t, lr-set-pos (l, pos), 0)))
 ∧   (p (p-set-pc (lr->p (lr-pop-tstk (test)),
                   tag ('pc,
                        cons (area-name (p-pc (l)),
                              3 + cadr-size + lr-p-pc-1))),
        p-clock1 (t, then, c))
       =   p-set-pc (lr->p (lr-eval (t, then, c)),
                     p-final-pc (t, then, 0)))
 ∧   proper-p-statep (lr->p (l))
 ∧   good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 ∧   lr-programs-properp (l, table)
 ∧   lr-p-proper-statep (p-temp-stk (l),
                         p-ctrl-stk (l),
                         p-data-segment (l),
```

$$
\begin{array}{ll}
 & \qquad\qquad\quad \mathit{table}) \\
\land & \text{lr-proper-formalsp}\,(\text{cdr}\,(\text{p-prog-segment}\,(l))) \\
\land & (\text{p-psw}\,(\text{lr-eval}\,(\mathbf{t},\ \mathit{then},\ c)) = \texttt{'run}) \\
\land & (\mathit{pos} = \text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \texttt{1})) \\
\rightarrow & (\text{p}\,(\text{lr->p}\,(l), \\
 & \quad \text{p-clock1}\,(\mathbf{t},\ \text{lr-set-pos}\,(l,\ \mathit{pos}),\ c) \\
 & \quad\ + \ \ \texttt{3} \\
 & \quad\ + \ \ \text{p-clock1}\,(\mathbf{t},\ \mathit{then},\ c) \\
 & \quad\ + \ \ \texttt{1}) \\
 & = \ \ \text{p-set-pc}\,(\text{lr->p}\,(\text{lr-eval}\,(\mathbf{t},\ \mathit{then},\ c)), \\
 & \qquad\qquad \text{p-final-pc}\,(\mathit{flag},\ l,\ \texttt{0}))) \ \textbf{endlet}
\end{array}
$$

THEOREM: lessp-plus-lr-p-pc-1-lr-p-c-size-3-1-lr-expr-car-if

$$
\begin{array}{ll}
\multicolumn{2}{l}{((\text{car}\,(\text{lr-expr}\,(l)) = \texttt{'if})} \\
\land & \text{listp}\,(\text{lr-expr}\,(l)) \\
\land & \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l))) \\
\land & \text{lr-programs-properp}\,(l,\ \mathit{table})) \\
\rightarrow & ((\text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(\text{lr-expr}\,(l))) \\
 & \quad + \ \ \texttt{3} \\
 & \quad + \ \ \text{lr-p-c-size}\,(\mathbf{t},\ \text{caddr}\,(\text{lr-expr}\,(l))) \\
 & \quad + \ \ \texttt{1} \\
 & \quad + \ \ \text{lr-p-pc-1}\,(\text{program-body}\,(\text{p-current-program}\,(l)),\ \text{offset}\,(\text{p-pc}\,(l)))) \\
 & < \ \ \text{length}\,(\text{program-body}\,(\text{assoc}\,(\text{area-name}\,(\text{p-pc}\,(l)), \\
 & \qquad\qquad\qquad\qquad \text{comp-programs}\,(\text{p-prog-segment}\,(l))))))
\end{array}
$$

THEOREM: find-label-else-start-lr-expr-car-if

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{listp}\,(\text{lr-expr}\,(l))} \\
\land & (\text{car}\,(\text{lr-expr}\,(l)) = \texttt{'if}) \\
\land & (\mathit{expr} = \text{lr-expr}\,(l)) \\
\land & (\mathit{body} = \text{program-body}\,(\text{p-current-program}\,(l))) \\
\land & (\mathit{pos} = \text{offset}\,(\text{p-pc}\,(l))) \\
\land & \text{lr-programs-properp}\,(l,\ \mathit{table}) \\
\land & \text{good-posp1}\,(\mathit{pos},\ \mathit{body})) \\
\rightarrow & (\text{find-label}\,(\text{lr-make-label}\,(1 + (1 + (1 + (1 + (\text{lr-p-c-size}\,(\mathbf{t}, \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{cadr}\,(\mathit{expr})) \\
 & \qquad\qquad\qquad\qquad\qquad\qquad + \ \ \text{lr-p-c-size}\,(\mathbf{t}, \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{caddr}\,(\mathit{expr})) \\
 & \qquad\qquad\qquad\qquad\qquad\qquad + \ \ \text{lr-p-pc-1}\,(\mathit{body}, \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathit{pos})))))), \\
 & \qquad\qquad \text{program-body}\,(\text{assoc}\,(\text{area-name}\,(\text{p-pc}\,(l)), \\
 & \qquad\qquad\qquad\qquad \text{comp-programs}\,(\text{p-prog-segment}\,(l))))) \\
 & = \ \ (1 + (1 + (1 + (1 + (\text{lr-p-c-size}\,(\mathbf{t},\ \text{cadr}\,(\mathit{expr})) \\
 & \qquad\qquad\qquad + \ \ \text{lr-p-c-size}\,(\mathbf{t},\ \text{caddr}\,(\mathit{expr})) \\
 & \qquad\qquad\qquad + \ \ \text{lr-p-pc-1}\,(\mathit{body},\ \mathit{pos})))))))
\end{array}
$$

THEOREM: lr-eval-p-pc-equivalence-helper-5-helper-1
let *test* **be** lr-eval (**t**, lr-set-pos (l, dv (offset (p-pc (l)), `1`)), c)
in
((p-psw (l) = `'run`)
∧ ($c \not\simeq$ `0`)
∧ listp (lr-expr (l))
∧ (car (lr-expr (l)) = `'if`)
∧ (p-psw (lr-if-ok (*test*)) = `'run`)
∧ (car (p-temp-stk (*test*)) = LR-F-ADDR)
∧ proper-p-statep (lr->p (l))
∧ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
∧ lr-programs-properp (l, *table*)
∧ lr-p-proper-statep (p-temp-stk (l),
p-ctrl-stk (l),
p-data-segment (l),
table)
∧ lr-proper-formalsp (cdr (p-prog-segment (l))))
→ (p (p-set-pc (lr->p (*test*),
p-final-pc (**t**,
lr-set-pos (l, dv (offset (p-pc (l)), `1`)),
`0`)),
`3`)
= p-set-pc (lr->p (lr-pop-tstk (*test*)),
tag (`'pc`,
cons (area-name (p-pc (l)),
lr-p-c-size (**t**, cadr (lr-expr (l)))
\+ `3`
\+ lr-p-c-size (**t**,
caddr (lr-expr (l)))
\+ `1`
\+ lr-p-pc-1 (program-body (p-current-program (l)),
offset (p-pc (l)))))))) **endlet**

THEOREM: lr-eval-p-pc-equivalence-helper-5-helper-2
let *test* **be** lr-eval (**t**, lr-set-pos (l, dv (offset (p-pc (l)), `1`)), c)
in
let *else* **be** lr-set-expr (lr-pop-tstk (*test*), l, dv (offset (p-pc (l)), `3`))
in
((*flag* ≠ `'list`)
∧ listp (lr-expr (l))
∧ (car (lr-expr (l)) = `'if`)
∧ good-posp1 (offset (p-pc (l)),
program-body (p-current-program (l)))
∧ lr-programs-properp (l, *table*))

$\rightarrow$ (p-final-pc (*flag*, *l*, 0) = p-final-pc (**t**, *else*, 0)) **endlet endlet**

THEOREM: lr-eval-p-pc-equivalence-helper-5

let *test* **be** lr-eval (**t**, lr-set-pos (*l*, *pos*), *c*),
  cadr-size **be** lr-p-c-size (**t**, cadr (lr-expr (*l*))),
  caddr-size **be** lr-p-c-size (**t**, caddr (lr-expr (*l*))),
  lr-p-pc-1 **be** lr-p-pc-1 (program-body (p-current-program (*l*)),
          offset (p-pc (*l*))),
  else **be** lr-set-expr (lr-pop-tstk (lr-eval (**t**, lr-set-pos (*l*, *pos*), *c*)),
          l,
          dv (offset (p-pc (*l*)), 3))
in
((p-psw (*l*) = `'run`)
$\land$ (*flag* $\neq$ `'list`)
$\land$ ($c \neq 0$)
$\land$ ($c \in \mathbf{N}$)
$\land$ listp (lr-expr (*l*))
$\land$ (car (lr-expr (*l*)) = `'if`)
$\land$ (p-psw (lr-if-ok (*test*)) = `'run`)
$\land$ (car (p-temp-stk (*test*)) = LR-F-ADDR)
$\land$ (p (p-set-pc (lr->p (lr-pop-tstk (*test*)),
      tag (`'pc`,
        cons (area-name (p-pc (*l*)),
          cadr-size
          + 3
          + *caddr-size*
          + 1
          + *lr-p-pc-1*))),
    p-clock1 (**t**, *else*, *c*))
  = p-set-pc (lr->p (lr-eval (**t**, *else*, *c*)),
      p-final-pc (**t**, *else*, 0)))
$\land$ (p (lr->p (*l*), p-clock1 (**t**, lr-set-pos (*l*, *pos*), *c*))
  = p-set-pc (lr->p (*test*),
      p-final-pc (**t**, lr-set-pos (*l*, *pos*), 0)))
$\land$ proper-p-statep (lr->p (*l*))
$\land$ good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
$\land$ lr-programs-properp (*l*, *table*)
$\land$ lr-p-proper-statep (p-temp-stk (*l*),
      p-ctrl-stk (*l*),
      p-data-segment (*l*),
      table)
$\land$ lr-proper-formalsp (cdr (p-prog-segment (*l*)))
$\land$ (p-psw (lr-eval (**t**, *else*, *c*)) = `'run`)
$\land$ (*pos* = dv (offset (p-pc (*l*)), 1)))

```
→    (p (lr->p (l),
          p-clock1 (t, lr-set-pos (l, pos), c)
          +   3
          +   p-clock1 (t, else, c))
       =   p-set-pc (lr->p (lr-eval (t, else, c)),
                     p-final-pc (flag, l, 0))) endlet
```

THEOREM: lr-p-pc-1-dv-1-car-lr-expr-temp-eval

```
 ((car (lr-expr (l)) = S-TEMP-EVAL)
  ∧   listp (lr-expr (l))
  ∧   good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
  ∧   lr-programs-properp (l, table)
  ∧   (body = program-body (p-current-program (l))))
 →   (lr-p-pc-1 (body, dv (offset (p-pc (l)), 1))
       =   lr-p-pc-1 (body, offset (p-pc (l))))
```

THEOREM: comp-body-1-car-expr-temp-eval

```
 (listp (expr) ∧ (car (expr) = S-TEMP-EVAL))
 →   (comp-body-1 (t, expr, n)
       =   append (comp-body-1 (t, cadr (expr), n),
                   list (list ('set-local, caddr (expr)))))
```

THEOREM: lr-eval-p-pc-equivalence-helper-5-get-lr-p-c-size

```
 (listp (lr-expr (l))
  ∧   (car (lr-expr (l)) = S-TEMP-EVAL)
  ∧   good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
  ∧   lr-programs-properp (l, table)
  ∧   (body = program-body (p-current-program (l)))
  ∧   (progs = p-prog-segment (l)))
 →   (unlabel (get (lr-p-pc-1 (body, offset (p-pc (l)))
                    +   lr-p-c-size (t, cadr (lr-expr (l))),
                    program-body (assoc (area-name (p-pc (l)),
                                         comp-programs (progs)))))
       =   list ('set-local, caddr (lr-expr (l))))
```

THEOREM: lr->p-lr-set-pos-dv-1-car-lr-expr-temp-eval

```
 (listp (lr-expr (l))
  ∧   (car (lr-expr (l)) = S-TEMP-EVAL)
  ∧   good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
  ∧   lr-programs-properp (l, table))
 →   (lr->p (lr-set-pos (l, dv (offset (p-pc (l)), 1))) = lr->p (l))
```

THEOREM: lr-eval-p-pc-equivalence-helper-6-helper

```
 let lr-eval be lr-eval (t, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c)
 in
```

((p-psw (l) = 'run)
 ∧ (*flag* ≠ 'list)
 ∧ ($c \neq 0$)
 ∧ ($c \in \mathbf{N}$)
 ∧ listp (lr-expr (l))
 ∧ (car (lr-expr (l)) = S-TEMP-EVAL)
 ∧ proper-p-statep (lr->p (l))
 ∧ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ (p-psw (*lr-eval*) = 'run))
→ (p (p-set-pc (lr->p (*lr-eval*),
                 p-final-pc (**t**,
                             lr-set-pos (l, dv (offset (p-pc (l)), 1)),
                             0)),
      1)
   = p-set-pc (lr->p (lr-set-temp (*lr-eval*,
                                   car (p-temp-stk (*lr-eval*)),
                                   caddr (lr-expr (l)))),
               p-final-pc (*flag*, l, 0))) **endlet**

THEOREM: lr-eval-p-pc-equivalence-helper-6
((p-psw (l) = 'run)
 ∧ (*flag* ≠ 'list)
 ∧ ($c \neq 0$)
 ∧ ($c \in \mathbf{N}$)
 ∧ listp (lr-expr (l))
 ∧ (car (lr-expr (l)) = S-TEMP-EVAL)
 ∧ (p (lr->p (l), p-clock1 (**t**, lr-set-pos (l, *pos*), c))
    = p-set-pc (lr->p (lr-eval (**t**, lr-set-pos (l, *pos*), c)),
                p-final-pc (**t**, lr-set-pos (l, *pos*), 0)))
 ∧ proper-p-statep (lr->p (l))
 ∧ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ (p-psw (lr-eval (**t**, lr-set-pos (l, *pos*), c)) = 'run)
 ∧ (*pos* = dv (offset (p-pc (l)), 1)))
→ (p-set-pc (lr->p (lr-set-temp (lr-eval (**t**, lr-set-pos (l, *pos*), c),
                                 car (p-temp-stk (lr-eval (**t**,
                                                           lr-set-pos (l, *pos*),
                                                           c))),
                                 caddr (lr-expr (l)))),
             p-final-pc (*flag*, l, 0))
   = p (lr->p (l), p-clock1 (**t**, lr-set-pos (l, *pos*), c) + 1))

THEOREM: get-comp-temp-test

```
(listp (expr) ∧ (car (expr) = S-TEMP-TEST))
→   (get (m, comp-body-1 (t, expr, n))
      =   if m < 4
          then get (m,
                    list (list ('push-local, caddr (expr)),
                          list ('push-constant,
                                identity (LR-UNDEF-ADDR)),
                          '(eq),
                          list ('test-bool-and-jump,
                                'f,
                                lr-make-label (n
                                               +   6
                                               +   lr-p-c-size (t,
                                                                cadr (expr))))))
          elseif m < (lr-p-c-size (t, cadr (expr)) + 4)
          then get (m − 4, comp-body-1 (t, cadr (expr), n + 4))
          else get (m − (lr-p-c-size (t, cadr (expr)) + 4),
                    list (list ('set-local, caddr (expr)),
                          list ('jump,
                                lr-make-label (n
                                               +   7
                                               +   lr-p-c-size (t,
                                                                cadr (expr)))),
                          list ('push-local, caddr (expr)))) endif)
```

THEOREM: lr-p-c-size-temp-test-opener

```
(listp (expr) ∧ (car (expr) = S-TEMP-TEST))
→   (lr-p-c-size (t, expr) = (lr-p-c-size (t, cadr (expr)) + 7))
```

THEOREM: get-+-lr-p-pc-1-lessp-3-temp-test-assoc-comp-programs

```
(listp (lr-expr (l))
 ∧   (car (lr-expr (l)) = S-TEMP-TEST)
 ∧   good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 ∧   lr-programs-properp (l, table)
 ∧   (n < 4)
 ∧   (progs = p-prog-segment (l))
 ∧   (body = program-body (p-current-program (l)))
 ∧   (pos = offset (p-pc (l))))
→   (unlabel (get (lr-p-pc-1 (body, pos) + n,
                   program-body (assoc (area-name (p-pc (l)),
                                        comp-programs (progs)))))
      =   get (n,
               list (list ('push-local, caddr (lr-expr (l))),
                     identity (list ('push-constant, LR-UNDEF-ADDR)),
```

```
                    '(eq),
                    list ('test-bool-and-jump,
                          'f,
                          lr-make-label (lr-p-pc-1 (body, pos)
                                         +  6
                                         +  lr-p-c-size (t,
                                                         cadr (lr-expr (l))))))))
```

THEOREM: car-comp-body-lr-expr-3-temp-test
(listp (*expr*) ∧ (car (*expr*) = S-TEMP-TEST))
→ (car (comp-body-1 (**t**, *expr*, *n*)) = list (**'push-local**, caddr (*expr*)))

THEOREM: get-+-lr-p-pc-1-n-2-size-temp-test-assoc-comp-programs

```
 (listp (lr-expr (l))
  ∧   (car (lr-expr (l)) = S-TEMP-TEST)
  ∧   good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
  ∧   lr-programs-properp (l, table)
  ∧   (n ≮ 4)
  ∧   (6 ≮ n)
  ∧   (progs = p-prog-segment (l))
  ∧   (body = program-body (p-current-program (l)))
  ∧   (pos = offset (p-pc (l))))
 →    (unlabel (get (lr-p-pc-1 (body, pos)
                     +  n
                     +  lr-p-c-size (t, cadr (lr-expr (l))),
                     program-body (assoc (area-name (p-pc (l)),
                                          comp-programs (progs)))))
       =  get (n - 4,
               list (list ('set-local, caddr (lr-expr (l))),
                     list ('jump,
                           lr-make-label (lr-p-pc-1 (body, pos)
                                          +  7
                                          +  lr-p-c-size (t,
                                                          cadr (lr-expr (l))))),
                     list ('push-local, caddr (lr-expr (l))))))
```

EVENT: Disable get-comp-temp-test.

EVENT: Disable lr-p-c-size-temp-test-opener.

THEOREM: definedp-caddr-lr-expr-bindings-ctrl-stk-rewrite

```
 (lr-programs-properp (l, table)
  ∧   ((car (lr-expr (l)) = S-TEMP-FETCH)
       ∨   (car (lr-expr (l)) = S-TEMP-TEST))
```

$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ proper-p-statep (lr->p (l)))
$\rightarrow$ definedp (caddr (lr-expr (l)), bindings (car (p-ctrl-stk (l))))

THEOREM: member-lr-good-pointerps-type-addr-untag-whole
(($addr \in list$)
$\wedge$ (type ($addr$) = `'addr`)
$\wedge$ (untag ($addr$) = $rest$)
$\wedge$ ($addr \neq$ list (`'addr`, $rest$)))
$\rightarrow$ ($\neg$ lr-good-pointerps ($list$, $data\text{-}seg$))

THEOREM: find-label-temp-test-end-lr-expr-car-temp-test
(listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) = S-TEMP-TEST)
$\wedge$ ($body$ = program-body (p-current-program (l)))
$\wedge$ (pos = offset (p-pc (l)))
$\wedge$ ($expr$ = lr-expr (l))
$\wedge$ lr-programs-properp (l, $table$)
$\wedge$ good-posp1 (pos, $body$)
$\wedge$ ($7 \not< n$))
$\rightarrow$ (find-label (lr-make-label (lr-p-pc-1 ($body$, pos)
$+$ n
$+$ lr-p-c-size (**t**, cadr ($expr$))),
program-body (assoc (area-name (p-pc (l)),
comp-programs (p-prog-segment (l)))))
= (lr-p-pc-1 ($body$, pos) + n + lr-p-c-size (**t**, cadr ($expr$))))

THEOREM: lr-p-proper-statep-lr-good-pointerps-strip-cdrs-binding
(lr-p-proper-statep (p-temp-stk (l), p-ctrl-stk (l), p-data-segment (l), $table$)
$\wedge$ proper-p-statep (lr->p (l)))
$\rightarrow$ lr-good-pointerps (strip-cdrs (bindings (car (p-ctrl-stk (l)))),
p-data-segment (l))

THEOREM: lr-eval-p-pc-equivalence-helper-7
(($flag \neq$ `'list`)
$\wedge$ listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) = S-TEMP-TEST)
$\wedge$ (p-max-temp-stk-size (l) $\not<$ (2 + length (p-temp-stk (l))))
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, $table$)
$\wedge$ lr-p-proper-statep (p-temp-stk (l),
p-ctrl-stk (l),
p-data-segment (l),
$table$)

$\wedge$ (p-psw (lr-do-temp-fetch (l)) = `'run`))
$\rightarrow$ (p-set-pc (lr->p (lr-do-temp-fetch (l)), p-final-pc (*flag*, l, `0`))
= p (lr->p (l), `5`))

THEOREM: lr-p-pc-dv-1-s-temp-test
(listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) = S-TEMP-TEST)
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*))
$\rightarrow$ (lr-p-pc-1 (program-body (p-current-program (l)), dv (offset (p-pc (l)), `1`))
= (lr-p-pc-1 (program-body (p-current-program (l)), offset (p-pc (l)))
+ `4`))

THEOREM: lr-eval-p-pc-equivalence-helper-8-helper-1
((p-psw (l) = `'run`)
$\wedge$ listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) = S-TEMP-TEST)
$\wedge$ (p-max-temp-stk-size (l) $\not<$ (`2` + length (p-temp-stk (l))))
$\wedge$ (local-var-value (caddr (lr-expr (l)), p-ctrl-stk (l)) = LR-UNDEF-ADDR)
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*))
$\rightarrow$ (lr->p (lr-set-pos (l, dv (offset (p-pc (l)), `1`))) = p (lr->p (l), `4`))

THEOREM: lr-eval-p-pc-equivalence-helper-8-helper-2-helper
(`4` + lr-p-c-size (**t**, *expr*) + `1`) = (`5` + lr-p-c-size (**t**, *expr*))

THEOREM: lr-eval-p-pc-equivalence-helper-8-helper-2
let *lr-eval* **be** lr-eval (**t**, lr-set-pos (l, dv (offset (p-pc (l)), `1`)), c)
in
((*flag* $\neq$ `'list`)
$\wedge$ listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) = S-TEMP-TEST)
$\wedge$ (p-max-temp-stk-size (l) $\neq$ `0`)
$\wedge$ (p-max-temp-stk-size (l) $\in \mathbf{N}$)
$\wedge$ (p-max-temp-stk-size (l) $\neq$ `1`)
$\wedge$ (((p-max-temp-stk-size (l) $-$ 1) $-$ 1) $\not<$ length (p-temp-stk (l)))
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (*lr-eval*) = `'run`))
$\rightarrow$ (p-set-pc (lr->p (lr-set-temp (*lr-eval*,
car (p-temp-stk (*lr-eval*)),
caddr (lr-expr (l)))),
p-final-pc (*flag*, l, `0`))

= p (p-set-pc (lr->p (*lr-eval*),
p-final-pc (**t**,
lr-set-pos (l,
dv (offset (p-pc (l)), 1)),
0)),
2)) **endlet**

EVENT: Disable lr-eval-p-pc-equivalence-helper-8-helper-2-helper.

THEOREM: lr-eval-p-pc-equivalence-helper-8
((p-psw (l) = 'run)
∧ (*flag* ≠ 'list)
∧ listp (lr-expr (l))
∧ (car (lr-expr (l)) = S-TEMP-TEST)
∧ (p-max-temp-stk-size (l) $\not<$ (2 + length (p-temp-stk (l))))
∧ (¬ lr-eval-temp-setp (l))
∧ (p (lr->p (lr-set-pos (l, *pos*)), p-clock1 (**t**, lr-set-pos (l, *pos*), c))
= p-set-pc (lr->p (lr-eval (**t**, lr-set-pos (l, *pos*), c)),
p-final-pc (**t**, lr-set-pos (l, *pos*), 0)))
∧ proper-p-statep (lr->p (l))
∧ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
∧ lr-programs-properp (l, *table*)
∧ lr-p-proper-statep (p-temp-stk (l),
p-ctrl-stk (l),
p-data-segment (l),
table)
∧ (p-psw (lr-eval (**t**, lr-set-pos (l, *pos*), c)) = 'run)
∧ (*pos* = dv (offset (p-pc (l)), 1)))
→ (p-set-pc (lr->p (lr-set-temp (lr-eval (**t**, lr-set-pos (l, *pos*), c),
car (p-temp-stk (lr-eval (**t**,
lr-set-pos (l, *pos*),
c))),
caddr (lr-expr (l)))),
p-final-pc (*flag*, l, 0))
= p (lr->p (l), 4 + p-clock1 (**t**, lr-set-pos (l, *pos*), c) + 2))

THEOREM: comp-body-1-car-expr-temp-fetch
(listp (*expr*) ∧ (car (*expr*) = S-TEMP-FETCH))
→ (comp-body-1 (**t**, *expr*, n) = list (list ('push-local, caddr (*expr*))))

THEOREM: get-lr-p-pc-1-comp-body-1-temp-fetch
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
∧ listp (lr-expr (l))
∧ (car (lr-expr (l)) = S-TEMP-FETCH)

$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (*prog* = assoc (area-name (p-pc (l)), p-prog-segment (l))))
$\rightarrow$ (get (lr-p-pc-1 (program-body (*prog*), offset (p-pc (l))),
comp-body-1 (`t`, program-body (*prog*), `0`))
= list (`'push-local`, caddr (lr-expr (l))))

THEOREM: lr-eval-p-pc-equivalence-helper-9
((p-psw (l) = `'run`)
$\wedge$ (*flag* $\neq$ `'list`)
$\wedge$ listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) = S-TEMP-FETCH)
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (p-psw (lr-do-temp-fetch (l)) = `'run`))
$\rightarrow$ (p-set-pc (lr->p (lr-do-temp-fetch (l)), p-final-pc (*flag*, l, `0`))
= p (lr->p (l), `1`))

THEOREM: lr-eval-p-pc-equivalence-helper-10
((p-psw (l) = `'run`)
$\wedge$ (*flag* $\neq$ `'list`)
$\wedge$ listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) = `'quote`)
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ (length (p-temp-stk (l)) < p-max-temp-stk-size (l)))
$\rightarrow$ (p-set-pc (lr->p (lr-set-tstk (l, cons (cadr (lr-expr (l)), p-temp-stk (l)))),
p-final-pc (*flag*, l, `0`))
= p (lr->p (l), `1`))

THEOREM: lr-expr-cur-expr-if-same
(car (cur-expr (offset (p-pc (l)), program-body (p-current-program (l)))) = `'if`)
= (car (lr-expr (l)) = `'if`)

THEOREM: lr-p-pc-lr-set-pos-dv-1-car-lr-expr-funcall
(listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
$\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-EVAL)
$\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-TEST)
$\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-FETCH)
$\wedge$ (car (lr-expr (l)) $\neq$ `'quote`)
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*))
$\rightarrow$ (lr-p-pc (lr-set-pos (l, dv (offset (p-pc (l)), `1`))) = lr-p-pc (l))

THEOREM: lr->p-lr-set-pos-dv-1-car-lr-expr-funcall

(listp (lr-expr (l))
 ∧ (car (lr-expr (l)) ≠ `'if`)
 ∧ (car (lr-expr (l)) ≠ S-TEMP-EVAL)
 ∧ (car (lr-expr (l)) ≠ S-TEMP-TEST)
 ∧ (car (lr-expr (l)) ≠ S-TEMP-FETCH)
 ∧ (car (lr-expr (l)) ≠ `'quote`)
 ∧ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*))
→ (lr->p (lr-set-pos (l, dv (offset (p-pc (l)), `1`))) = lr->p (l))

THEOREM: p-set-pc-lr->p-equal-p-fact
(p-set-pc (lr->p (p-state (*pc1*,
 ctrl-stk,
 temp-stk,
 prog-seg,
 data-seg,
 max-ctrl,
 max-temp,
 word-size,
 psw)),
 pc2)
 = p)
= ((p-pc (p) = *pc2*)
 ∧ (p-ctrl-stk (p) = *ctrl-stk*)
 ∧ (p-temp-stk (p) = *temp-stk*)
 ∧ (p-data-segment (p) = *data-seg*)
 ∧ (p-prog-segment (p) = comp-programs (*prog-seg*))
 ∧ (p-max-ctrl-stk-size (p) = *max-ctrl*)
 ∧ (p-max-temp-stk-size (p) = *max-temp*)
 ∧ (p-word-size (p) = *word-size*)
 ∧ (p-psw (p) = *psw*))

EVENT: Disable p-set-pc-lr->p-equal-p-fact.

THEOREM: lr->p-p-run-subr-p-run-subr-clock
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 ∧ lr-programs-properp (l, *table*)
 ∧ listp (lr-expr (l))
 ∧ (car (lr-expr (l)) ≠ `'if`)
 ∧ subrp (car (lr-expr (l))))
→ (p-run-subr (car (lr-expr (l)),
 p-set-pc (lr->p (*new-l*),
 add-addr (lr-p-pc (l),
 lr-p-c-size-list (arity (car (lr-expr (l))),

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{lr-expr}\,(l)))))\\
\quad = \quad \text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(\textit{new-l}),\\
\qquad\qquad\qquad\qquad \text{add-addr}\,(\text{lr-p-pc}\,(l),\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{lr-p-c-size-list}\,(\text{arity}\,(\text{car}\,(\text{lr-expr}\,(l))),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{lr-expr}\,(l)))),\\
\qquad\qquad \text{p-run-subr-clock}\,(l,\ \textit{new-l})))
\end{array}
$$

THEOREM: p-pc-run-car

$$
\begin{array}{l}
(\text{proper-p-statep}\,(\text{lr->p}\,(l))\\
\ \wedge\ \ (\text{p-psw}\,(l) = \texttt{'run})\\
\ \wedge\ \ (\text{p-psw}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-car-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
\qquad = \ \texttt{'run})\\
\ \wedge\ \ \text{lr-programs-properp}\,(l,\ \textit{table})\\
\ \wedge\ \ (\text{unlabel}\,(\text{get}\,(\text{offset}\,(pc),\\
\qquad\qquad\qquad \text{program-body}\,(\text{assoc}\,(\text{area-name}\,(pc),\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{p-prog-segment}\,(\text{lr->p}\,(l))))))\\
\qquad = \ \texttt{'(call car)}))\\
\to\ \ (\text{p-pc}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-car-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
\qquad = \ \text{add-addr}\,(pc,\ \texttt{1}))
\end{array}
$$

THEOREM: p-pc-run-cdr

$$
\begin{array}{l}
(\text{proper-p-statep}\,(\text{lr->p}\,(l))\\
\ \wedge\ \ (\text{p-psw}\,(l) = \texttt{'run})\\
\ \wedge\ \ (\text{p-psw}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-cdr-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
\qquad = \ \texttt{'run})\\
\ \wedge\ \ \text{lr-programs-properp}\,(l,\ \textit{table})\\
\ \wedge\ \ (\text{unlabel}\,(\text{get}\,(\text{offset}\,(pc),\\
\qquad\qquad\qquad \text{program-body}\,(\text{assoc}\,(\text{area-name}\,(pc),\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{p-prog-segment}\,(\text{lr->p}\,(l))))))\\
\qquad = \ \texttt{'(call cdr)}))\\
\to\ \ (\text{p-pc}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-cdr-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
\qquad = \ \text{add-addr}\,(pc,\ \texttt{1}))
\end{array}
$$

THEOREM: p-pc-run-cons

$$
\begin{array}{l}
(\text{proper-p-statep}\,(\text{lr->p}\,(l))\\
\ \wedge\ \ (\text{p-psw}\,(l) = \texttt{'run})\\
\ \wedge\ \ (\text{p-psw}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-cons-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
\qquad = \ \texttt{'run})\\
\ \wedge\ \ \text{lr-programs-properp}\,(l,\ \textit{table})\\
\ \wedge\ \ (\text{unlabel}\,(\text{get}\,(\text{offset}\,(pc),\\
\qquad\qquad\qquad \text{program-body}\,(\text{assoc}\,(\text{area-name}\,(pc),\\
\qquad\qquad\qquad\qquad\qquad\qquad \text{p-prog-segment}\,(\text{lr->p}\,(l))))))\\
\qquad = \ \texttt{'(call cons)}))\\
\to\ \ (\text{p-pc}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-cons-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
\qquad = \ \text{add-addr}\,(pc,\ \texttt{1}))
\end{array}
$$

THEOREM: p-pc-run-false
$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{proper-p-statep}\,(\text{lr->p}\,(l))}\\
\land & (\text{p-psw}\,(l) = \texttt{'run})\\
\land & (\text{p-psw}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-false-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
 & \quad = \quad \texttt{'run})\\
\land & \text{lr-programs-properp}\,(l,\ \mathit{table})\\
\land & (\text{unlabel}\,(\text{get}\,(\text{offset}\,(pc),\\
 & \qquad\qquad \text{program-body}\,(\text{assoc}\,(\text{area-name}\,(pc),\\
 & \qquad\qquad\qquad\qquad \text{p-prog-segment}\,(\text{lr->p}\,(l))))))\\
 & \quad = \quad \texttt{'(call false)}))\\
\to & (\text{p-pc}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-false-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
 & \quad = \quad \text{add-addr}\,(pc,\ \texttt{1}))
\end{array}
$$

THEOREM: p-pc-run-falsep
$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{proper-p-statep}\,(\text{lr->p}\,(l))}\\
\land & (\text{p-psw}\,(l) = \texttt{'run})\\
\land & (\text{p-psw}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-falsep-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
 & \quad = \quad \texttt{'run})\\
\land & \text{lr-programs-properp}\,(l,\ \mathit{table})\\
\land & (\text{unlabel}\,(\text{get}\,(\text{offset}\,(pc),\\
 & \qquad\qquad \text{program-body}\,(\text{assoc}\,(\text{area-name}\,(pc),\\
 & \qquad\qquad\qquad\qquad \text{p-prog-segment}\,(\text{lr->p}\,(l))))))\\
 & \quad = \quad \texttt{'(call falsep)}))\\
\to & (\text{p-pc}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-falsep-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
 & \quad = \quad \text{add-addr}\,(pc,\ \texttt{1}))
\end{array}
$$

THEOREM: p-pc-run-listp
$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{proper-p-statep}\,(\text{lr->p}\,(l))}\\
\land & (\text{p-psw}\,(l) = \texttt{'run})\\
\land & (\text{p-psw}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-listp-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
 & \quad = \quad \texttt{'run})\\
\land & \text{lr-programs-properp}\,(l,\ \mathit{table})\\
\land & (\text{unlabel}\,(\text{get}\,(\text{offset}\,(pc),\\
 & \qquad\qquad \text{program-body}\,(\text{assoc}\,(\text{area-name}\,(pc),\\
 & \qquad\qquad\qquad\qquad \text{p-prog-segment}\,(\text{lr->p}\,(l))))))\\
 & \quad = \quad \texttt{'(call listp)}))\\
\to & (\text{p-pc}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-listp-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
 & \quad = \quad \text{add-addr}\,(pc,\ \texttt{1}))
\end{array}
$$

THEOREM: p-pc-run-nlistp
$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{proper-p-statep}\,(\text{lr->p}\,(l))}\\
\land & (\text{p-psw}\,(l) = \texttt{'run})\\
\land & (\text{p-psw}\,(\text{p}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc),\ \text{p-nlistp-clock}\,(\text{p-set-pc}\,(\text{lr->p}\,(l),\ pc))))\\
 & \quad = \quad \texttt{'run})\\
\land & \text{lr-programs-properp}\,(l,\ \mathit{table})
\end{array}
$$

$\wedge$ (unlabel (get (offset (pc),
program-body (assoc (area-name (pc),
p-prog-segment (lr->p (l))))))
= `'(call nlistp)`))
$\rightarrow$ (p-pc (p (p-set-pc (lr->p (l), pc), p-nlistp-clock (p-set-pc (lr->p (l), pc))))
= add-addr (pc, `1`))

THEOREM: p-pc-run-true
(proper-p-statep (lr->p (l))
$\wedge$ (p-psw (l) = `'run`)
$\wedge$ (p-psw (p (p-set-pc (lr->p (l), pc), p-true-clock (p-set-pc (lr->p (l), pc))))
= `'run`)
$\wedge$ lr-programs-properp (l, $table$)
$\wedge$ (unlabel (get (offset (pc),
program-body (assoc (area-name (pc),
p-prog-segment (lr->p (l))))))
= `'(call true)`))
$\rightarrow$ (p-pc (p (p-set-pc (lr->p (l), pc), p-true-clock (p-set-pc (lr->p (l), pc))))
= add-addr (pc, `1`))

THEOREM: p-pc-run-truep
(proper-p-statep (lr->p (l))
$\wedge$ (p-psw (l) = `'run`)
$\wedge$ (p-psw (p (p-set-pc (lr->p (l), pc), p-truep-clock (p-set-pc (lr->p (l), pc))))
= `'run`)
$\wedge$ lr-programs-properp (l, $table$)
$\wedge$ (unlabel (get (offset (pc),
program-body (assoc (area-name (pc),
p-prog-segment (lr->p (l))))))
= `'(call truep)`))
$\rightarrow$ (p-pc (p (p-set-pc (lr->p (l), pc), p-truep-clock (p-set-pc (lr->p (l), pc))))
= add-addr (pc, `1`))

THEOREM: p-run-subr-p-pc-add-addr-lr-p-pc-lr-p-c-size
let pos **be** dv (offset (p-pc (l)), `1`)
in
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, $table$)
$\wedge$ listp (lr-expr (l))
$\wedge$ subrp (car (lr-expr (l)))
$\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
$\wedge$ (car (lr-expr (l)) $\neq$ `'quote`)
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ (p-psw (lr-eval (`'list`, lr-set-pos (l, pos), c)) = `'run`)
$\wedge$ (p-psw (p-run-subr (car (lr-expr (l)),

$$
\begin{array}{l}
\qquad\qquad\qquad\text{p-set-pc}\,(\text{lr->p}\,(\text{lr-eval}\,(\texttt{'list},\\
\qquad\qquad\qquad\qquad\qquad\qquad\text{lr-set-pos}\,(l,\ pos),\\
\qquad\qquad\qquad\qquad\qquad\qquad c)),\\
\qquad\qquad\qquad\qquad\text{lr-return-pc}\,(l))))\\
\quad = \quad \texttt{'run}))\\
\to \quad (\text{p-pc}\,(\text{p-run-subr}\,(\text{car}\,(\text{lr-expr}\,(l)),\\
\qquad\qquad\qquad\text{p-set-pc}\,(\text{lr->p}\,(\text{lr-eval}\,(\texttt{'list},\\
\qquad\qquad\qquad\qquad\qquad\qquad\text{lr-set-pos}\,(l,\ pos),\\
\qquad\qquad\qquad\qquad\qquad\qquad c)),\\
\qquad\qquad\qquad\qquad\text{lr-return-pc}\,(l))))\\
\quad = \quad \text{add-addr}\,(\text{lr-return-pc}\,(l),\ \texttt{1}))\ \textbf{endlet}
\end{array}
$$

THEOREM: lr-eval-p-pc-equivalence-helper-11

$$
\begin{array}{l}
((\text{p-psw}\,(l) = \texttt{'run})\\
\wedge \quad (\mathit{flag} \neq \texttt{'list})\\
\wedge \quad (c \neq \texttt{0})\\
\wedge \quad (c \in \mathbf{N})\\
\wedge \quad \text{listp}\,(\text{lr-expr}\,(l))\\
\wedge \quad (\text{car}\,(\text{lr-expr}\,(l)) \neq \texttt{'if})\\
\wedge \quad (\text{car}\,(\text{lr-expr}\,(l)) \neq \text{S-TEMP-EVAL})\\
\wedge \quad (\text{car}\,(\text{lr-expr}\,(l)) \neq \text{S-TEMP-TEST})\\
\wedge \quad (\text{car}\,(\text{lr-expr}\,(l)) \neq \text{S-TEMP-FETCH})\\
\wedge \quad (\text{car}\,(\text{lr-expr}\,(l)) \neq \texttt{'quote})\\
\wedge \quad \text{subrp}\,(\text{car}\,(\text{lr-expr}\,(l)))\\
\wedge \quad (\text{p-psw}\,(\text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ pos),\ c)) = \texttt{'run})\\
\wedge \quad (\text{p}\,(\text{lr->p}\,(l),\ \text{p-clock1}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ pos),\ c))\\
\qquad = \quad \text{p-set-pc}\,(\text{lr->p}\,(\text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ pos),\ c)),\\
\qquad\qquad\qquad \text{p-final-pc}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ pos),\ \texttt{0})))\\
\wedge \quad \text{proper-p-statep}\,(\text{lr->p}\,(l))\\
\wedge \quad \text{good-posp1}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \text{program-body}\,(\text{p-current-program}\,(l)))\\
\wedge \quad \text{lr-programs-properp}\,(l,\ \mathit{table})\\
\wedge \quad (\text{p-psw}\,(\text{lr-apply-subr}\,(l,\ \text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ pos),\ c)))\\
\qquad = \quad \texttt{'run})\\
\wedge \quad (pos = \text{dv}\,(\text{offset}\,(\text{p-pc}\,(l)),\ \texttt{1})))\\
\to \quad (\text{p-set-pc}\,(\text{lr->p}\,(\text{lr-apply-subr}\,(l,\ \text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ pos),\ c))),\\
\qquad\qquad \text{p-final-pc}\,(\mathit{flag},\ l,\ \texttt{0}))\\
\qquad = \quad \text{p}\,(\text{lr->p}\,(l),\\
\qquad\qquad \text{p-clock1}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ pos),\ c)\\
\qquad\qquad + \quad \text{p-run-subr-clock}\,(l,\\
\qquad\qquad\qquad\qquad \text{lr-eval}\,(\texttt{'list},\ \text{lr-set-pos}\,(l,\ pos),\ c))))
\end{array}
$$

THEOREM: p-set-pc-twice

$$\text{p-set-pc}\,(\text{p-set-pc}\,(p,\ \mathit{pc1}),\ \mathit{pc2}) = \text{p-set-pc}\,(p,\ \mathit{pc2})$$

THEOREM: lr-eval-p-pc-equivalence-helper-12-helper-1

(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ listp (lr-expr (l))
$\land$ (car (lr-expr (l)) $\neq$ `'if`)
$\land$ (car (lr-expr (l)) $\neq$ `'quote`)
$\land$ ($\neg$ subrp (car (lr-expr (l))))
$\land$ litatom (car (lr-expr (l)))
$\land$ (p-psw (lr-eval (**t**, lr-funcall (l, *new-l*), $c - 1$)) $=$ `'run`)
$\land$ (p-psw (*new-l*) $=$ `'run`)
$\land$ (p-prog-segment (l) $=$ p-prog-segment (*new-l*)))
$\rightarrow$ (p (p-set-pc (lr->p (*new-l*),
add-addr (lr-p-pc (l),
lr-p-c-size-list (arity (car (lr-expr (l))),
lr-expr (l)))),
`1`)
$=$ lr->p (lr-funcall (l, *new-l*)))

THEOREM: lr-expr-funcall
((p-psw (lr-funcall (l, *new-l*)) $=$ `'run`)
$\land$ (p-prog-segment (l) $=$ p-prog-segment (*new-l*)))
$\rightarrow$ (lr-expr (lr-funcall (l, *new-l*))
$=$ program-body (assoc (user-fname (car (lr-expr (l))),
p-prog-segment (l))))

THEOREM: unlabel-car-last-comp-body
unlabel (car (last (comp-body (*body*)))) $=$ `'(ret)`

THEOREM: unlabel-get-last-funcall-body-assoc-comp-programs
(listp (lr-expr (l))
$\land$ ($\neg$ subrp (car (lr-expr (l))))
$\land$ (car (lr-expr (l)) $\neq$ `'quote`)
$\land$ litatom (car (lr-expr (l)))
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ (*prog-seg* $=$ p-prog-segment (l)))
$\rightarrow$ (unlabel (get (lr-p-c-size (**t**,
program-body (assoc (user-fname (car (lr-expr (l))),
prog-seg))),
program-body (assoc (user-fname (car (lr-expr (l))),
comp-programs (*prog-seg*)))))
$=$ `'(ret)`)

THEOREM: lr-eval-preserves-cdr-p-ctrl-stk-lr-funcall
(listp (lr-expr (l))
$\land$ ($\neg$ subrp (car (lr-expr (l))))

$\land$ (car (lr-expr (l)) $\neq$ `'if`)
$\land$ (car (lr-expr (l)) $\neq$ `'quote`)
$\land$ litatom (car (lr-expr (l)))
$\land$ proper-p-statep (lr->p (l))
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ (p-psw (lr-eval (**t**, lr-funcall (l, *new-l*), $c - 1$)) = `'run`)
$\land$ (p-psw (*new-l*) = `'run`)
$\land$ (*new-l* = lr-eval (`'list`, lr-set-pos (l, dv (offset (p-pc (l)), `1`)), c)))
$\rightarrow$ (cdr (p-ctrl-stk (lr-eval (**t**, lr-funcall (l, *new-l*), $c - 1$)))
= p-ctrl-stk (*new-l*))

THEOREM: lr-eval-preserves-ret-pc-car-p-ctrl-stk
(proper-p-statep (lr->p (l))
$\land$ good-posp (*flag*, offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ (p-psw (lr-eval (*flag*, l, c)) = `'run`))
$\rightarrow$ (ret-pc (car (p-ctrl-stk (lr-eval (*flag*, l, c))))
= ret-pc (car (p-ctrl-stk (l))))

THEOREM: lr-eval-preserves-ret-pc-car-p-ctrl-stk-lr-funcall
(listp (lr-expr (l))
$\land$ ($\neg$ subrp (car (lr-expr (l))))
$\land$ (car (lr-expr (l)) $\neq$ `'if`)
$\land$ (car (lr-expr (l)) $\neq$ `'quote`)
$\land$ litatom (car (lr-expr (l)))
$\land$ proper-p-statep (lr->p (l))
$\land$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ (p-psw (lr-eval (**t**, lr-funcall (l, *new-l*), $c - 1$)) = `'run`)
$\land$ (p-psw (*new-l*) = `'run`)
$\land$ (*new-l* = lr-eval (`'list`, lr-set-pos (l, dv (offset (p-pc (l)), `1`)), c)))
$\rightarrow$ (ret-pc (car (p-ctrl-stk (lr-eval (**t**, lr-funcall (l, *new-l*), $c - 1$))))
= add-addr (lr-return-pc (l), `1`))

THEOREM: lr-eval-p-pc-equivalence-helper-12-helper-2
let *new-l* **be** lr-eval (`'list`, lr-set-pos (l, dv (offset (p-pc (l)), `1`)), c)
in
(good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\land$ lr-programs-properp (l, *table*)
$\land$ listp (lr-expr (l))
$\land$ (car (lr-expr (l)) $\neq$ `'if`)
$\land$ (car (lr-expr (l)) $\neq$ `'quote`)
$\land$ ($\neg$ subrp (car (lr-expr (l))))
$\land$ proper-p-statep (lr->p (l))

$\wedge$ litatom (car (lr-expr (l)))
$\wedge$ lr-proper-formalsp (cdr (p-prog-segment (l)))
$\wedge$ (p-psw (lr-eval (`t`, lr-funcall (l, *new-l*), $c - 1$)) = `'run`)
$\wedge$ (p-psw (*new-l*) = `'run`))
$\rightarrow$ (p (p-set-pc (lr->p (lr-eval (`t`, lr-funcall (l, *new-l*), $c - 1$)),
add-addr (lr-p-pc (lr-funcall (l, *new-l*)),
lr-p-c-size (`t`,
lr-expr (lr-funcall (l, *new-l*))))),
`1`)
= p-set-pc (lr->p (lr-pop-cstk (lr-eval (`t`,
lr-funcall (l, *new-l*),
$c - 1$))),
add-addr (lr-p-pc (l),
lr-p-c-size-list (arity (car (lr-expr (l))),
lr-expr (l))
+ `1`))) **endlet**

THEOREM: lr-eval-p-pc-equivalence-helper-12
let *fs* **be** lr-funcall (l, lr-eval (`'list`, lr-set-pos (l, *pos*), c))
in
((p-psw (l) = `'run`)
$\wedge$ (*flag* $\neq$ `'list`)
$\wedge$ ($c \neq$ `0`)
$\wedge$ ($c \in \mathbf{N}$)
$\wedge$ listp (lr-expr (l))
$\wedge$ (car (lr-expr (l)) $\neq$ `'if`)
$\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-EVAL)
$\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-TEST)
$\wedge$ (car (lr-expr (l)) $\neq$ S-TEMP-FETCH)
$\wedge$ (car (lr-expr (l)) $\neq$ `'quote`)
$\wedge$ ($\neg$ subrp (car (lr-expr (l))))
$\wedge$ litatom (car (lr-expr (l)))
$\wedge$ (p-psw (lr-eval (`'list`, lr-set-pos (l, *pos*), c)) = `'run`)
$\wedge$ (p (lr->p (*fs*), p-clock1 (`t`, *fs*, $c - 1$))
= p-set-pc (lr->p (lr-eval (`t`, *fs*, $c - 1$)),
p-final-pc (`t`, *fs*, `0`)))
$\wedge$ (p (lr->p (l), p-clock1 (`'list`, lr-set-pos (l, *pos*), c))
= p-set-pc (lr->p (lr-eval (`'list`, lr-set-pos (l, *pos*), c)),
p-final-pc (`'list`, lr-set-pos (l, *pos*), `0`)))
$\wedge$ proper-p-statep (lr->p (l))
$\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
$\wedge$ lr-programs-properp (l, *table*)
$\wedge$ lr-proper-formalsp (cdr (p-prog-segment (l)))
$\wedge$ (p-psw (lr-eval (`t`, *fs*, $c - 1$)) = `'run`)

```
  ∧   (pos = dv (offset (p-pc (l)), 1)))
→   (p-set-pc (lr->p (lr-pop-cstk (lr-eval (t, fs, c − 1))),
                  p-final-pc (flag, l, 0))
       =   p (lr->p (l),
               p-clock1 ('list, lr-set-pos (l, pos), c)
               +   1
               +   p-clock1 (t, fs, c − 1)
               +   1)) endlet
```

THEOREM: lr-eval-p-pc-equivalence

```
 (proper-p-statep (lr->p (l))
  ∧   good-posp (flag, offset (p-pc (l)), program-body (p-current-program (l)))
  ∧   ((flag ≠ 'list)
        ∨   (car (cur-expr (butlast (offset (p-pc (l))),
                               program-body (p-current-program (l))))
             ≠   'if))
  ∧   lr-programs-properp (l, table)
  ∧   lr-p-proper-statep (p-temp-stk (l),
                          p-ctrl-stk (l),
                          p-data-segment (l),
                          table)
  ∧   lr-proper-formalsp (cdr (p-prog-segment (l)))
  ∧   (p-psw (lr-eval (flag, l, c)) = 'run))
→   (p (lr->p (l), p-clock1 (flag, l, c))
       =   p-set-pc (lr->p (lr-eval (flag, l, c)), p-final-pc (flag, l, 0)))
```

EVENT: Disable lr-expr-cur-expr-if-same.

DEFINITION:

```
logic->p-clock (expr,
                alist,
                program-names,
                heap-size,
                max-temp-stk-size,
                max-ctrl-stk-size,
                word-size)
=   (p-clock1 (t,
               s->lr (logic->s (expr, alist, program-names),
                      heap-size,
                      max-temp-stk-size,
                      max-ctrl-stk-size,
                      word-size),
               cdr (v&c$ (t, expr, alist)) + 1)
     +   2)
```

DEFINITION:
total-heap-reqs (*expr*, *alist*, *program-names*, *heap-size*)
= lr-total-heap-reqs (*expr*,
alist,
program-names,
heap-size,
cdr (v&c$ (**t**, *expr*, *alist*)) + 1)

DEFINITION:
max-ctrl-reqs (*expr*, *alist*, *program-names*)
= lr-max-ctrl-reqs (*expr*,
alist,
program-names,
cdr (v&c$ (**t**, *expr*, *alist*)) + 1)

DEFINITION:
max-temp-reqs (*expr*, *alist*, *program-names*)
= lr-max-temp-reqs (*expr*,
alist,
program-names,
cdr (v&c$ (**t**, *expr*, *alist*)) + 1)

DEFINITION:
max-word-size-reqs (*expr*, *alist*, *program-names*, *heap-size*)
= lr-max-word-size-reqs (*expr*,
alist,
program-names,
heap-size,
cdr (v&c$ (**t**, *expr*, *alist*)) + 1)

THEOREM: s-formals-s-prog-logic->s
s-formals (s-prog (logic->s (*expr*, *alist*, *pnames*))) = strip-cars (*alist*)

THEOREM: deposit-answer-addr-preserves-lr-valp
(adpp (untag (LR-ANSWER-ADDR), *data-seg*) ∧ lr-valp (*value*, *addr*, *data-seg*))
→ lr-valp (*value*,
addr,
deposit (*anything*, identity (LR-ANSWER-ADDR), *data-seg*))

THEOREM: p-last-2-instrs-main-program
(adpp (untag (LR-ANSWER-ADDR), p-data-segment (*l*))
∧ (p-psw (*l*) = `'run`)
∧ listp (p-temp-stk (*l*))
∧ lr-valp (*value*, car (p-temp-stk (*l*)), p-data-segment (*l*)))
→ lr-valp (*value*,

```
            fetch (identity (LR-ANSWER-ADDR),
                   p-data-segment (p (p-set-pc (lr->p (l),
                                                tag ('pc,
                                                     cons (name (car (p-prog-segment (l))),
                                                           lr-p-c-size (t,
                                                                        program-body (car (p-prog-segment (l))))
                                     2))),
            p-data-segment (p (p-set-pc (lr->p (l),
                                         tag ('pc,
                                              cons (name (car (p-prog-segment (l))),
                                                    lr-p-c-size (t,
                                                                 program-body (car (p-prog-segment (l))))))),
                              2)))
```

THEOREM: lr-programs-properp-s->lr-logic->s

```
 (l-proper-exprp (t, expr, pnames, strip-cars (alist))
  ∧  l-proper-programsp (pnames)
  ∧  all-litatoms (strip-cars (alist))
  ∧  l-data-seg-body-restrictedp (t, expr)
  ∧  l-restrict-subrps-progs (pnames)
  ∧  l-restrict-subrps (t, expr)
  ∧  l-restrictedp (pnames, alist)
  ∧  (heap-size ≮ s-total-heap-reqs (s-progs (logic->s (expr, alist, pnames)),
                                     alist,
                                     heap-size)))
 →   lr-programs-properp (s->lr (logic->s (expr, alist, pnames),
                                 heap-size,
                                 max-ctrl,
                                 max-temp,
                                 word-size),
                          cdr (lr-data-seg-table (s-progs (logic->s (expr,
                                                                     alist,
                                                                     pnames)),
                                                  alist,
                                                  heap-size)))
```

THEOREM: logic->p-ok-really-helper-1

```
 (proper-p-statep (lr->p (l))
  ∧  good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
  ∧  lr-programs-properp (l, table)
  ∧  lr-p-proper-statep (p-temp-stk (l),
                         p-ctrl-stk (l),
                         p-data-segment (l),
                         table)
```

∧ lr-proper-formalsp (cdr (p-prog-segment (*l*)))
∧ (p-psw (lr-eval (**t**, *l*, *c*)) = 'run)
∧ adpp (untag (LR-ANSWER-ADDR), p-data-segment (*l*))
∧ (area-name (p-pc (*l*)) = name (car (p-prog-segment (*l*))))
∧ (offset (p-pc (*l*)) = **nil**)
∧ lr-valp (*value*,
car (p-temp-stk (lr-eval (**t**, *l*, *c*))),
p-data-segment (lr-eval (**t**, *l*, *c*))))
→ lr-valp (*value*,
fetch (LR-ANSWER-ADDR,
p-data-segment (p (p (lr->p (*l*), p-clock1 (**t**, *l*, *c*)), 2))),
p-data-segment (p (p (lr->p (*l*), p-clock1 (**t**, *l*, *c*)), 2)))

THEOREM: name-car-p-prog-segment-s->lr
name (car (p-prog-segment (s->lr (*s*, *heap-size*, *max-ctrl*, *max-temp*, *word-size*))))
= name (car (s-progs (*s*)))

THEOREM: p-pc-s->lr
p-pc (s->lr (*s*, *heap-size*, *max-temp*, *max-ctrl*, *word-size*))
= tag ('pc, cons (s-pname (*s*), s-pos (*s*)))

THEOREM: adpp-untag-lr-answer-addr-s->lr
adpp (identity (untag (LR-ANSWER-ADDR)),
p-data-segment (s->lr (*s*, *heap-size*, *max-ctrl*, *max-temp*, *word-size*)))

THEOREM: lr-s-similar-statesp-s->lr-logic->s-lr-data-seg-table
((*heap-size* $\not<$ s-total-heap-reqs (s-progs (logic->s (*expr*, *alist*, *pnames*)),
alist,
heap-size))
∧ l-restrictedp (*pnames*, *alist*)
∧ l-data-seg-body-restrictedp (**t**, *expr*))
→ lr-s-similar-statesp (*alist*,
nil,
s->lr (logic->s (*expr*, *alist*, *pnames*),
heap-size,
max-ctrl,
max-temp,
word-size),
cdr (lr-data-seg-table (s-progs (logic->s (*expr*,
alist,
pnames)),
alist,
heap-size)))

THEOREM: s-eval-flag-run-v&c$-not-f-flag-t

(v&c$ (**t**, *expr*, *alist*)
$\wedge$ (cdr (v&c$ (**t**, *expr*, *alist*)) $<$ *c*)
$\wedge$ l-proper-programsp (*pnames*)
$\wedge$ l-proper-exprp (**t**, *expr*, *pnames*, strip-cars (*alist*))
$\wedge$ all-litatoms (strip-cars (*alist*)))
$\rightarrow$ (s-err-flag (s-eval (**t**, logic->s (*expr*, *alist*, *pnames*), *c*)) = `'run`)

THEOREM: lr-proper-formalsp-cdr-p-prog-segment-s->lr-logic->s
(l-proper-programsp (*pnames*)
$\wedge$ l-proper-exprp (**t**, *expr*, *pnames*, strip-cars (*alist*))
$\wedge$ all-litatoms (strip-cars (*alist*)))
$\rightarrow$ lr-proper-formalsp (cdr (p-prog-segment (s->lr (logic->s (*expr*, *alist*, *pnames*), *heap-size*, *max-ctrl*, *max-temp*, *word-size*))))

THEOREM: lr-s-similar-params-lr-good-pointerps-strip-cdrs
lr-s-similar-params (*s-params*, *lr-params*, *data-seg*)
$\rightarrow$ lr-good-pointerps (strip-cdrs (*lr-params*), *data-seg*)

THEOREM: lr-p-proper-statep-s->lr
((*heap-size* $\not<$ s-total-heap-reqs (s-progs (*s*), s-params (*s*), *heap-size*))
$\wedge$ s-restrictedp (s-progs (*s*), s-params (*s*))
$\wedge$ (*params* = s-params (*s*)))
$\rightarrow$ lr-p-proper-statep (p-temp-stk (s->lr (*s*, *heap-size*, *max-ctrl*, *max-temp*, *word-size*)),
p-ctrl-stk (s->lr (*s*, *heap-size*, *max-ctrl*, *max-temp*, *word-size*)),
p-data-segment (s->lr (*s*, *heap-size*, *max-ctrl*, *max-temp*, *word-size*)),
cdr (lr-data-seg-table (s-progs (*s*), *params*, *heap-size*)))

THEOREM: proper-p-statep-lr->p-s->lr-logic->s

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{l-proper-exprp}\,(\mathbf{t},\ \mathit{expr},\ \mathit{pnames},\ \text{strip-cars}\,(\mathit{alist}))} \\
\land & \text{l-proper-programsp}\,(\mathit{pnames}) \\
\land & \text{all-litatoms}\,(\text{strip-cars}\,(\mathit{alist})) \\
\land & \text{l-data-seg-body-restrictedp}\,(\mathbf{t},\ \mathit{expr}) \\
\land & \text{l-restrictedp}\,(\mathit{pnames},\ \mathit{alist}) \\
\land & \text{l-restrict-subrps}\,(\mathbf{t},\ \mathit{expr}) \\
\land & \text{l-restrict-subrps-progs}\,(\mathit{pnames}) \\
\land & (\mathit{heap\text{-}size} \not< (\text{s-total-heap-reqs}\,(\text{s-progs}\,(\text{logic->s}\,(\mathit{expr},\ \mathit{alist},\ \mathit{pnames})), \\
& \qquad\qquad\qquad\qquad\qquad\qquad \mathit{alist}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad \mathit{heap\text{-}size}) \\
& \qquad\qquad\quad +\ \ \text{s-eval-heap-r}\,(\mathbf{t},\ \text{logic->s}\,(\mathit{expr},\ \mathit{alist},\ \mathit{pnames}),\ c))) \\
\land & (\mathit{max\text{-}ctrl} \not< \text{lr-max-ctrl-reqs}\,(\mathit{expr},\ \mathit{alist},\ \mathit{pnames},\ c)) \\
\land & (\mathit{max\text{-}ctrl} < \exp\,(2,\ \mathit{word\text{-}size})) \\
\land & (\mathit{max\text{-}ctrl} \in \mathbf{N}) \\
\land & (\mathit{max\text{-}temp} \not< \text{lr-max-temp-reqs}\,(\mathit{expr},\ \mathit{alist},\ \mathit{pnames},\ c)) \\
\land & (\mathit{max\text{-}temp} < \exp\,(2,\ \mathit{word\text{-}size})) \\
\land & (\mathit{max\text{-}temp} \in \mathbf{N}) \\
\land & (\mathit{word\text{-}size} \not< \text{lr-max-word-size-reqs}\,(\mathit{expr},\ \mathit{alist},\ \mathit{pnames},\ \mathit{heap\text{-}size},\ c)) \\
\land & (\mathit{word\text{-}size} \in \mathbf{N})) \\
\rightarrow & \text{proper-p-statep}\,(\text{lr->p}\,(\text{s->lr}\,(\text{logic->s}\,(\mathit{expr},\ \mathit{alist},\ \mathit{pnames}), \\
& \qquad\qquad\qquad\qquad\qquad \mathit{heap\text{-}size}, \\
& \qquad\qquad\qquad\qquad\qquad \mathit{max\text{-}ctrl}, \\
& \qquad\qquad\qquad\qquad\qquad \mathit{max\text{-}temp}, \\
& \qquad\qquad\qquad\qquad\qquad \mathit{word\text{-}size})))
\end{array}
$$

THEOREM: logic->p-ok-really

$$
\begin{array}{ll}
\multicolumn{2}{l}{(\text{l-proper-exprp}\,(\mathbf{t},\ \mathit{expr},\ \mathit{pnames},\ \text{strip-cars}\,(\mathit{alist}))} \\
\land & \text{l-proper-programsp}\,(\mathit{pnames}) \\
\land & \text{all-litatoms}\,(\text{strip-cars}\,(\mathit{alist})) \\
\land & \text{l-data-seg-body-restrictedp}\,(\mathbf{t},\ \mathit{expr}) \\
\land & \text{l-restrictedp}\,(\mathit{pnames},\ \mathit{alist}) \\
\land & \text{v\&c\$}\,(\mathbf{t},\ \mathit{expr},\ \mathit{alist}) \\
\land & \text{l-restrict-subrps}\,(\mathbf{t},\ \mathit{expr}) \\
\land & \text{l-restrict-subrps-progs}\,(\mathit{pnames}) \\
\land & (\mathit{heap\text{-}size} \not< \text{total-heap-reqs}\,(\mathit{expr},\ \mathit{alist},\ \mathit{pnames},\ \mathit{heap\text{-}size})) \\
\land & (\mathit{max\text{-}ctrl} \not< \text{max-ctrl-reqs}\,(\mathit{expr},\ \mathit{alist},\ \mathit{pnames})) \\
\land & (\mathit{max\text{-}ctrl} < \exp\,(2,\ \mathit{word\text{-}size})) \\
\land & (\mathit{max\text{-}ctrl} \in \mathbf{N}) \\
\land & (\mathit{max\text{-}temp} \not< \text{max-temp-reqs}\,(\mathit{expr},\ \mathit{alist},\ \mathit{pnames})) \\
\land & (\mathit{max\text{-}temp} < \exp\,(2,\ \mathit{word\text{-}size})) \\
\land & (\mathit{max\text{-}temp} \in \mathbf{N}) \\
\land & (\mathit{word\text{-}size} \not< \text{max-word-size-reqs}\,(\mathit{expr},\ \mathit{alist},\ \mathit{pnames},\ \mathit{heap\text{-}size})) \\
\land & (\mathit{word\text{-}size} \in \mathbf{N})) \\
\rightarrow & \text{lr-valp}\,(\text{car}\,(\text{v\&c\$}\,(\mathbf{t},\ \mathit{expr},\ \mathit{alist})),
\end{array}
$$

```
fetch (LR-ANSWER-ADDR,
       p-data-segment (p (logic->p (expr,
                                    alist,
                                    pnames,
                                    heap-size,
                                    max-ctrl,
                                    max-temp,
                                    word-size),
                          logic->p-clock (expr,
                                          alist,
                                          pnames,
                                          heap-size,
                                          max-ctrl,
                                          max-temp,
                                          word-size)))),
p-data-segment (p (logic->p (expr,
                             alist,
                             pnames,
                             heap-size,
                             max-ctrl,
                             max-temp,
                             word-size),
                   logic->p-clock (expr,
                                   alist,
                                   pnames,
                                   heap-size,
                                   max-ctrl,
                                   max-temp,
                                   word-size))))
```

Index

add-addr, 6, 7, 10, 11, 14, 15, 37, 44, 48, 53–55, 75–81, 91, 96–99, 102, 106, 109, 110, 112–115, 117–121, 126, 128, 153, 154, 160, 162, 181, 185, 186, 196, 199, 200, 218, 219, 225, 226, 230–232, 234, 240, 242, 250, 252, 255–258, 264–266, 268–273, 289, 316, 341, 351, 354, 371–377
add-addr-add-addr, 53
add-addr-of-non-number, 53
add-addr-p-final-pc, 354
add1-add1-lr-boundary-nodep, 241
add1-addr, 14, 38, 47
add1-lr-boundary-nodep, 241
adp-name, 40, 44, 53, 101, 115
adp-name-cons, 101
adp-name-untag-add-addr, 53
adp-name-untag-sub-addr, 44
adp-offset, 40, 44
adp-offset-cons, 44
adp-offset-untag-add-addr, 44
adp-offset-untag-sub-addr, 44
adpp, 7, 47, 48, 52, 55, 75, 96, 99, 102–104, 106, 107, 109, 112, 114–118, 120–127, 130, 153–156, 185, 196, 199, 200, 218–221, 225, 231–233, 235, 240–242, 249, 250, 254–258, 264–268, 289, 308–311, 318, 339, 342, 379, 381
adpp-add-addr-0, 48
adpp-add-addr-fact-2, 106
adpp-adpp-sub-addr, 103
adpp-area-name-offset-same, 106
adpp-cons-heap-name-node-size-lr-init-data-seg, 263
adpp-cons-pack-definedp-area-name, 48
adpp-cons-pack-opener, 218
adpp-deposit-a-list, 120
adpp-deposit-anything-at-all, 47
adpp-deposit-other-area, 115
adpp-fetch-lr-fp-addr-car-lr-compile-quote, 241
adpp-lessp-offset-deposit, 47
adpp-lr-compile-quote, 220
adpp-same-signature, 48
adpp-same-signature-car-lr-compile-quote, 221
adpp-same-signature-lr-apply-subr, 156
adpp-untag-add-addr-lr-nodep-not-max-addr, 199
adpp-untag-add-addr-offset-car, 196
adpp-untag-add-addr-offset-cdr, 196
adpp-untag-add-addr-offset-on-free-listp, 200
adpp-untag-answer-addr-car-lr-data-seg-table, 318
adpp-untag-definedp-area-name, 47
adpp-untag-definedp-area-name-free-ptr, 220
adpp-untag-lessp-offset, 48
adpp-untag-listp, 48
adpp-untag-lr-answer-addr-s->lr, 381
adpp-untag-lr-fp-addr-lr-init-data-seg, 225
adpp-untag-numberp-offset, 48
all-definedp, 327, 328
all-definedp-strip-cdrs-lr-init-data-seg-table, 328
all-litatoms, 35, 220, 273, 274, 280, 284, 305, 306, 308, 309, 318, 329, 330, 332, 380, 382, 383
all-litatoms-all-user-fnamesp-plistp, 309
all-litatoms-not-plist, 331, 332
all-litatoms-not-plist-lr-proper-programsp, 332

all-litatoms-s-formals-member-l
r-programs-properp, 306
all-litatoms-s-formals-member-s
-programs-properp, 305
all-litatoms-strip-cdrs-lr-make
-temp-name-alist, 280
-temp-name-alist-1, 280
all-numberps, 344
all-numberps-strip-cadrs-numberp
-cdr-assoc, 344
all-numberps-strip-cadrs-subr-a
rity-alist, 344
all-p-objectps, 49, 52, 73, 74, 310, 315, 316
all-p-objectps-append, 73
all-p-objectps-bad-type, 52
all-p-objectps-first-n, 74
all-p-objectps-get, 310
all-p-objectps-lr->p-similar-
states, 49
all-p-objectps-lr-init-heap-conte
nts, 316
nts-helper, 315
nts-helper-helper, 315
all-p-objectps-put, 310
all-p-objectps-reverse, 73
all-p-objects-lookup, 220, 234–239, 263
all-p-objects-lookup-cons-table, 234
all-p-objects-lookup-deposit, 236
all-p-objects-lookup-deposit-a-
list, 235
all-p-objects-lookup-lr-compile
-quote, 237
all-p-objects-lookup-lr-data-se
g-table-body, 238
g-table-list, 239
all-p-objects-lookup-strip-cdrs
-lr-init-data-seg-table, 263
all-undef-addr-strip-cdrs-lr-ma
ke-initial-temps, 324
all-undef-addrs, 35, 284, 292, 324, 344
all-undef-addrs-strip-cadrs-lr-
make-temp-var-dcls, 292
all-undef-addrs-strip-cadrs-temp
-vars-programs-properp, 344
-vars-programs-properp-1, 344
all-user-fnamesp, 35, 51, 291, 292, 307–309, 332
all-user-fnamesp-strip-cars-s-c
onstruct-programs, 332
append-butlast-list-car-last, 348
append-first-n-restn, 167
area-name, 7, 8, 15, 21, 23, 24, 33, 35, 38–40, 45, 47, 48, 52–55, 57, 59, 63, 72, 75–94, 103–106, 108, 110, 113, 115–118, 120, 121, 126, 127, 133, 134, 153, 154, 157, 163, 172, 187, 198–204, 218, 220, 226–233, 240, 242, 249, 255, 256, 264–266, 268, 269, 289, 308–311, 315, 316, 339, 343, 347, 352, 354–362, 364–366, 369, 371–373, 381
area-name-add-addr, 53
area-name-lr-p-pc, 45
area-name-lr-return-pc, 53
area-name-p-final-pc, 355
area-name-p-pc-lr-eval, 47
area-name-p-pc-lr-funcall, 91
area-name-sub-addr, 54
area-name-tag, 40
arity, 34, 57, 153, 163, 168–170, 180, 198–205, 207, 208, 211, 342–345, 370, 371, 375, 377
arity-formals-not-quote, 169
ascii-0, 12, 31, 283
ascii-1, 12
ascii-9, 12
ascii-dash, 12, 31
assoc-append-1, 50
assoc-cdr, 288, 289
assoc-definedp-table-lr-compile
-quote, 240
assoc-definedp-table-lr-data-se
g-table-body, 327

g-table-list, 327
assoc-definedp-table-lr-init-dat
a-seg-table, 240
assoc-put-assoc-3, 47
axiom-53, 1

bindings, 23, 61, 62, 74, 91, 92, 94, 95, 122, 123, 130, 134, 135, 141, 142, 144, 145, 150, 151, 183, 191, 336, 337, 353, 366
bindings-make-p-call-frame, 74
butlast, 7, 56, 166, 282, 295, 296, 304, 348, 349, 378

caar-lr-compile-programs, 211
cadddr-max-r, 190
car-append, 57
car-car-lr-compile-programs-pro
gs, 132
car-comp-body-1-litatom, 347
car-comp-body-lr-expr-3-temp-te
st, 365
car-last-first-n-add1-get, 166
car-lr-compile-body, 129
car-lr-expr-s->lr1, 137
car-reverse-last, 166
car-untag-lr-p-pc, 45
car-untag-lr-return-pc, 54
car-untag-p-pc-lr-eval, 86
car-untag-p-pc-lr-funcall, 91
cddr-add-addr, 53
cddr-lr-return-pc, 54
cddr-nil-lr-p-pc, 45
cddr-nil-make-p-call-frame, 74
cddr-sub-addr, 54
cdr-assoc-member-strip-cdrs, 107
cdr-compile-quote-list-t0-lr-init
-data-seg-cons-table, 271
cdr-p-temp-stk-lr-apply-subr, 160
cdr-p-temp-stk-p-run-subr, 159
cdr-untag-lr-p-pc-lr-funcall, 90
change-elements, 1
comp-body, 33, 58, 59, 72, 93, 284, 375
comp-body-1, 32, 33, 57–59, 66–72, 84, 285, 286, 288, 292, 308, 309, 347, 354, 362, 364, 365, 368, 369
comp-body-1-car-expr-if, 354
comp-body-1-car-expr-temp-eval, 362
comp-body-1-car-expr-temp-fetch, 368
comp-body-1-list-not-listp, 285
comp-if, 31, 32, 57, 66–68, 286, 351, 354, 356
comp-programs, 33, 36, 44, 45, 51, 59–61, 63, 73, 85, 86, 89, 91–93, 187, 210, 285–292, 307, 308, 310, 347, 354, 356, 357, 359, 362, 364–366, 370, 375
comp-programs-1, 33, 45, 59–61, 210, 281, 308
comp-programs-assoc-cons-opener, 210
comp-temp-test, 31, 33, 66, 68, 288
count-codelist, 13
count-codelist-make-symbol, 13
count-codelist1, 12, 13, 146, 147
count-codelist1-append-non-listp, 147
count-codelist1-cdr-gensym, 147
count-codelist1-cons, 146
count-list, 15, 16
cur-expr, 7, 56, 62, 63, 65, 67–72, 85, 129, 158, 282, 295–304, 348, 349, 356, 369, 378
cur-expr-add1-opener, 356
cur-expr-lr-compile-body-t, 129
cur-expr-nlistp-pos, 85

definedp, 16, 33–35, 42, 45, 47, 48, 50, 51, 53, 57, 59–61, 63, 74, 84, 85, 91, 92, 94, 101–104, 107, 112, 116, 120, 132–135, 141–145, 147, 149–151, 156, 163, 169, 187, 199, 206, 208, 210, 212, 216–224, 234, 237–243, 245, 249–256, 259–264, 267–279, 286–292, 294, 295, 301, 302, 306–308, 310,

311, 313–315, 317, 320–322,
324–328, 331, 344, 366
definedp-0, 210
definedp-append, 45
definedp-area-name-member-p-cur
rent-program, 59
definedp-caddr-lr-expr-bindings
-ctrl-stk, 63
-ctrl-stk-rewrite, 365
definedp-cadr-cur-expr-quote-lr
-data-seg-table, 302
-data-seg-table-body, 301
-data-seg-table-list, 302
definedp-car-lr-compile-quote, 219
definedp-cdr-assoc-lr-good-pointe
rps, 336
definedp-cdr-lr-compile-quote-t, 264
definedp-comp-programs-1-define
dp-orig, 45
definedp-comp-programs-definedp
-lr-programs-properp, 91
-orig, 45
definedp-deposit, 47
definedp-deposit-a-list, 120
definedp-listp-cdr-assoc-lr-prope
r-p-areasp, 102
definedp-litatom-lr-proper-p-are
as, 101
definedp-lr-compile-programs, 112
definedp-lr-data-seg-body-list-
n-not-lessp, 301
definedp-lr-funcall-prog-segment, 91
definedp-lr-heap-name-lr-init-d
ata-seg, 234
definedp-lr-make-temp-name-alist, 151
-1, 151
definedp-name-p-objectp-tag-0-l
r-proper-p-areasp, 237
definedp-object-cdr-lr-compile-q
uote-list, 254
definedp-pairlist, 151
definedp-s-temps-s-eval, 147
definedp-strip-cars-append-membe
r-x, 134
r-x-2, 149
definedp-table-definedp-car-lr-
data-seg-table-body, 221
data-seg-table-list, 221
definedp-table-definedp-car-lr-i
nit-data-seg-table, 234
definedp-table-definedp-cdr-lr-
compile-quote, 219
data-seg-table-body, 220
data-seg-table-body-n, 301
data-seg-table-list, 220
definedp-table-definedp-cdr-lr-i
nit-data-seg-table, 221
definedp-table-lr-compile-quote
-self, 240
definedp-user-fname-s-construct
-programs, 331
definition, 37
definitions-subrps-lr-programs-p
roperp, 51
delete, 96, 107, 117, 181, 228, 229,
249, 250, 252
delete-all, 331
delete-append, 228
deposit, 14, 15, 47, 53, 78, 81, 102,
107, 110, 114–121, 125, 185,
186, 218, 231, 232, 236, 237,
241, 255, 256, 268, 310, 338,
339, 379
deposit-a-list, 14, 15, 47, 78, 81, 120,
121, 126–128, 162, 186, 218,
220, 221, 231, 232, 235, 236,
242, 249, 254, 257–261, 311–
313, 316, 339, 341
deposit-a-list-cons-opener, 47
deposit-a-list-nil, 47
deposit-answer-addr-preserves-l
r-valp, 379
deposit-cons, 218
deposit-deposit, 109
deposit-free-ptr-preserves-lr-v
alp, 107
deposit-good-node-preserves-lr-p
roper-free-listp, 121

deposit-ref-count-move-inward-2, 119
deposit-ref-count-move-outward, 110
difference-decreases, 35
disjointp, 141–145
disjointp-commutative, 145
disjointp-cons-arg2, 143
disjointp-lr-make-temp-name-ali
st-1, 143
st-2, 145
disjointp-nlistp-arg2, 143
disjointp-plist-arg-2, 144
dl, 31
dv, 40–42, 44, 62, 73, 110–112, 136–140, 144, 145, 148, 152, 156, 158–160, 164, 165, 168, 171, 173, 179, 180, 188–190, 192–195, 205–215, 282, 295–299, 302, 334, 335, 340, 342, 343, 345, 346, 350, 353–363, 367–370, 373, 374, 376, 378

equal-add-addr-fact, 226
equal-append-final-0, 146
equal-append-initial, 146
equal-append-same-length-fact, 134
equal-ascii-0-lr-convert-digit-t
o-ascii, 283
equal-cddr-p-frame-nil, 3
equal-lengths-same-signature-ca
r-lr-compile-quote, 221
equal-p-psw-lr-eval-run-lr-eval
-lr-set-error, 137
equal-plus-lessp-fact, 205
equal-plus-remainder-0-fact, 55
exp, 88, 195, 196, 318, 329, 332, 383
exp-log-2-lessp-add1-fact-1, 196
exp-log-lessp-fact-1, 196

fall-off-proofp, 306
fall-off-proofp-append-cons-ret, 306
fetch, 6, 7, 10, 11, 15, 17, 27, 52, 53, 55, 75–84, 96–98, 102, 104, 106, 109, 112–114, 121, 126, 128, 153, 154, 160–163, 181, 182, 186–189, 191, 196, 198–207, 217–219, 221, 225, 230, 232, 235, 240–242, 249–266, 268–272, 275–280, 289, 294, 295, 310–315, 320–328, 341, 380, 381, 384
fetch-0-addr-compile-quote-list
-init-data-seg, 271
fetch-add-addr-deposit-a-list-n
ode, 126
fetch-add-addr-ref-count-f-addr
-lr-init-data-seg, 269
fetch-add-addr-ref-count-lr-0-a
ddr-lr-init-data-seg, 270
fetch-add-addr-ref-count-lr-t-a
ddr-lr-init-data-seg, 270
fetch-add-addr-ref-count-offset
-lr-init-data-seg-help-1, 226
fetch-cons, 217
fetch-deposit, 53
fetch-deposit-a-list, 218
fetch-deposit-a-list-node, 126
fetch-f-addr-compile-quote-list
-init-data-seg, 271
fetch-fp-addr-compile-quote-list
-t0-lr-init-data-seg-cons-table, 272
fetch-init-init-data-seg-genera
lized, 229
fetch-init-init-data-seg-sub-ad
dr, 232
fetch-lr-f-addr-lr-init-data-se
g, 269
fetch-lr-fp-addr-compile-quote-
0, 268
fetch-lr-fp-addr-compile-quote-t, 264
fetch-lr-fp-addr-lr-init-data-se
g, 225
fetch-lr-nodep-add-addr, 106
fetch-offset-lr-t-addr-ref-count
-offset-compile-quote-t, 264
fetch-ref-count-0-addr-compile-q
uote-list-init-data-seg, 271
fetch-ref-count-f-addr-compile-q
uote-list-init-data-seg, 272

fetch-ref-count-lr-init-data-se
g-free-list, 266
fetch-ref-count-t-addr-compile-q
uote-list-init-data-seg, 270
fetch-t-addr-compile-quote-list
-init-data-seg, 270
fetch-unbox-nat-0-addr-compile-q
uote-list-init-data-seg, 271
find-label, 357, 359, 366
find-label-else-start-lr-expr-c
ar-if, 359
find-label-lr-make-label-label-i
nstrs, 357
find-label-past-else-lr-expr-ca
r-if, 357
find-label-temp-test-end-lr-exp
r-car-temp-test, 366
find-labelp, 64, 283, 284
find-labelp-lr-make-label-comp-
body, 284
find-labelp-lr-make-label-label
-instrs, 284
find-non-proper-instr, 8, 9
find-non-proper-programs, 8, 9
first-n, 73, 74, 166, 167, 344
firstn, 8, 9, 13, 23, 67, 68, 70, 71,
108, 134, 135, 140, 141, 144
firstn-lr-p-c-size-restn-lr-p-p
c-1-comp-body-1, 71
c-1-comp-body-1-helper-1, 67
c-1-comp-body-1-helper-4, 68
c-1-comp-body-1-helper-5, 68
c-1-comp-body-1-helper-6, 69
c-1-comp-body-1-helper-7, 70
c-1-comp-body-1-helper-8, 70
firstn-put-assoc, 140
firstn-restn-plus-comp-if-1, 67
firstn-restn-plus-comp-if-2, 67
firstn-restn-small-enough-cdr-c
omp-body-1-list, 70
firstn-unlabel-instrs-comp-body
-1-lr-p-pc-1-helper-2, 67
-1-lr-p-pc-1-helper-3, 68
fix-data-segment, 8, 9
fix-program-segment, 8
formal-vars, 9, 23, 33, 35, 38, 56,
57, 59–61, 63, 91, 132, 134,
141, 165, 168, 169, 212, 287–
289, 291, 292, 306, 309
formal-vars-assoc-comp-programs, 60
-1, 60
-lr-programs-properp, 91
formal-vars-lr-compile-programs, 132
formal-vars-p-current-program-s
->lr1, 132

gensym, 13, 14, 147, 280
gensym-is-new, 14
get, 2, 21, 40, 52, 53, 56, 60, 65,
66, 68–72, 75–85, 126, 129,
153, 163, 166, 187, 198, 200–
204, 217, 218, 226, 296, 300–
303, 310, 347, 348, 350, 351,
354, 356, 362, 364, 365, 369,
371–373, 375
get-+-lr-p-pc-1-lessp-3-temp-te
st-assoc-comp-programs, 364
get-+-lr-p-pc-1-n-2-size-temp-te
st-assoc-comp-programs, 365
get-append, 72
get-cdr-lr-init-heap-contents, 226
get-comp-body-lr-compile-progra
ms, 187
get-comp-if, 351
get-comp-if-helper, 351
get-comp-if-helper-helper, 350
get-comp-temp-test, 364
get-firstn-different-lists, 71
get-label-instrs, 71
get-last-funcall-cur-expr, 84
get-lr-compile-body, 129
get-lr-compile-body-list, 129
get-lr-p-c-size-lessp-lr-p-c-si
ze-comp-body-1, 72
get-lr-p-c-size-lessp-restn-lr-p
-pc-1-comp-body-1, 354
get-lr-p-pc-1-comp-body-1-cur-e
xpr-comp-body, 72

get-lr-p-pc-1-comp-body-1-quote, 72
get-lr-p-pc-1-comp-body-1-temp-fetch, 368
get-offset-return-pc-program-body-assoc-comp-programs, 84
get-plus, 71
get-plus-lr-p-c-size-cadr-caddr-4-comp-body-cur-expr, 356
get-plus-lr-p-pc-1-lr-pc-size-cadr-assoc-comp-body-if-4, 356
get-sub1-length-car-last, 166
good-alistp, 307
good-alistp-lr-programs-properp, 307
good-posp, 4, 56, 58, 73, 95, 122–125, 130–132, 141, 148, 155, 170–172, 175, 176, 182–184, 190–192, 215, 216, 302, 304, 337, 346, 349, 376, 378
good-posp-cons-lessp-4-if-lr-programs-properp, 62
good-posp-dv-1-funcall-lr-expr, 73
good-posp-dv-1-funcall-opened, 302
good-posp-dv-1-temps-lr-expr, 62
good-posp-list-nx-offset-program-body, 58
good-posp-list-nx-t-simple, 56
good-posp-list-t-offset-program-body, 58
good-posp-lr-compile-body, 132
good-posp1, 56–58, 61–65, 67–73, 84–95, 114, 121–124, 129, 134–141, 144–146, 148, 150–160, 163–165, 168–171, 173–176, 183, 185, 187, 188, 190, 192–195, 205–212, 214, 215, 296–299, 301–304, 336–339, 341–343, 345–350, 352–370, 373–377, 380
good-posp1-cons-lessp-4-if-lr-proper-exprp, 62
good-posp1-cons-lessp-4-if-s-proper-exprp, 296
good-posp1-dv-1-temp-eval-test, 298
good-posp1-dv-1-temps-lr-expr, 144
good-posp1-expand-list-temps, 129
good-posp1-lr-compile-body, 129
good-posp1-lr-proper-exprp-get-cadddr, 68

ihint-2, 110–112
increment-num-list-count-code-list1, 13
increment-numlist, 12, 13
induct-hint-1, 166
induct-hint-10, 69
induct-hint-11, 134
induct-hint-13, 140, 141
induct-hint-14, 142
induct-hint-15, 155
induct-hint-16, 254
induct-hint-17, 225
induct-hint-18, 195, 196
induct-hint-19, 226
induct-hint-2, 269
induct-hint-20, 300
induct-hint-22, 283
induct-hint-23, 283, 284
induct-hint-3, 295, 296
induct-hint-4, 112
induct-hint-6, 59
induct-hint-7, 65, 66
induct-hint-8, 153
induct-hint-9, 71

l-data-seg-body-restrictedp, 330–332, 380, 381, 383
l-data-seg-body-restrictedp-delete-all, 331
l-data-seg-body-restrictedp-s-data-seg-body-restrictedp, 331
l-data-seg-list-restrictedp, 330, 331
l-data-seg-list-restrictedp-s-data-seg-list-restrictedp, 331
l-eval, 4
l-proper-exprp, 330–332, 380, 382, 383
l-proper-programsp, 329–332, 380, 382, 383

l-proper-programsp-1, 331
l-proper-programsp-s-progs-logi
c->s, 330
l-restrict-subrps, 216, 331, 332, 380, 383
l-restrict-subrps-progs, 216, 331, 332, 380, 383
l-restrict-subrps-progs-delete-all, 331
l-restrict-subrps-progs-s-restri
ct-subrps-progs, 331
ct-subrps-progs-logic->s, 331
l-restrict-subrps-s-restrict-su
brps, 331
l-restrictedp, 330, 332, 380, 381, 383
label-instrs, 31, 33, 58, 59, 71, 93, 284, 285, 309, 357
label-instrs-append, 284
label-instrs-proper-labeled-p-i
nstructionsp, 285
labelledp, 285
last, 7, 56, 166, 167, 205, 282, 295, 296, 298, 304, 348, 349, 375
lastcdr, 130
legal-labelp, 8, 71
legal-labelp-label-make-label, 71
length, 8, 10, 13, 15, 21, 23, 24, 31, 32, 34, 38, 41, 47, 48, 52, 56–60, 63–72, 74, 75, 84, 86, 89, 93, 96, 99, 102–104, 107, 111, 112, 116, 120, 121, 123, 129, 130, 134, 135, 139, 149, 153–156, 158–160, 163, 164, 166–170, 173, 180, 182–184, 192, 198–205, 207–213, 215, 217, 218, 220, 221, 224, 226, 242, 248–250, 252, 253, 264, 267, 268, 270, 273, 281–284, 286, 296, 298, 300–303, 310–313, 315, 318, 319, 324, 328, 329, 341–345, 349, 351–353, 356, 357, 359, 366–369
length-3-cdr-cddr-not-nil, 296
length-add1-add1-cddr-fact, 167
length-butlast, 166
length-car-lr-compile-quote, 224
length-cdr-assoc-lr-heap-name-l
r-init-data-seg, 226
length-cdr-comp-if-comp-body, 57
length-cdr-lr-expr-funcall, 153
length-cdr-lr-expr-funcall-s->
lr1, 155
length-comp-body-1-lr-p-c-size, 58
length-comp-body-lr-p-c-size, 58
length-comp-if-alt, 66
length-comp-temp-test, 66
length-delete-member, 96
length-deposit, 116
length-deposit-a-list, 220
length-formal-vars-lr-proper-fo
rmalsp-arity, 169
length-label-instrs, 58
length-last, 205
length-last-fact, 298
length-lr-all-nodes, 273
length-lr-compile-body-list, 129
length-lr-compile-body-t, 129
length-lr-convert-num-to-ascii, 283
length-lr-do-temp-fetch, 63
length-lr-init-heap-contents, 217
length-lr-make-initial-temps, 281
length-lr-make-temp-name-alist, 209
length-lr-make-temp-name-alist-1, 208
length-lr-make-temp-var-dcls, 208
length-lr-push-tstk, 64
length-make-temps-entries, 209
length-p-temp-stk-lr-apply-subr, 168
length-p-temp-stk-lr-eval, 170
length-p-temp-stk-lr-eval-flag-
list, 170
list-alt, 345
not-list, 184
length-p-temp-stk-lr-eval-lr-fu
ncall, 211
length-p-temp-stk-lr-eval-lr-set
-pos, 353
-pos-flag-t, 215

length-p-temp-stk-lr-funcall, 169
length-p-temp-stk-lr-pop-tstk-l
r-eval-flag-t, 192
length-p-temp-stk-p-run-subr, 168
length-p-temp-stk-p-run-subr-he
lper-1, 167
length-pair-formals-with-addres
ses, 281
length-pair-temps-with-initial-v
alues, 75
length-pairlist, 75
length-popn, 74
length-popn-lessp-fact, 93
length-s-eval-list, 155
length-strip-cars, 112
length-strip-cdrs, 281
lessp-1-not-zerop-exp, 195
lessp-1-not-zerop-log, 195
lessp-3-lr-p-c-size-car-if, 354
lessp-4-not-zerop-not-1-not-2-3, 64
-get-car-pos, 65
lessp-cdr-untag-lr-return-pc-lr
-p-c-size, 86
lessp-count-list-cdr-count-list
-whole, 16
lessp-count-not-list-car-count-
list-whole, 16
lessp-difference-fact-1, 105
lessp-difference-lr-boundary-of
fsetp-fact-1, 105
lessp-difference-node-size-sub-
addr, 105
addr-2, 228
addr-3, 230
lessp-index-lessp-lr-p-c-size-li
st, 65
lessp-length-deposit, 102
lessp-lr-boundary-offsetp-3, 265
lessp-lr-boundary-offsetp-nodep
-plus-node-size-fact-1, 250
-plus-node-size-fact-2, 102
lessp-lr-node-on-boundaryp-node
-size, 105
lessp-lr-p-c-size-list-lessp-su
b1-length, 59
lessp-lr-p-pc-1-lr-p-c-size, 60
lessp-lr-p-pc-1-lr-p-c-size-helpe
r-1, 60
lessp-max-arg2, 211
lessp-minimum-heap-size-not-0-f
-t-must-be-undef-alt-1, 265
-t-must-be-undef-alt-1-help, 265
lessp-number-cons-cur-expr-dv-1
-listp, 282
lessp-number-cons-cur-expr-dv-2
-listp, 282
lessp-number-cons-cur-expr-dv-3
-listp, 282
lessp-number-cons-restn-cdr, 282
lessp-offset-lr-init-data-seg-a
dpp-untag-lessp-offset, 266
lessp-offset-lr-return-pc-lr-p-
c-size-good-posp, 93
lessp-plus-lr-p-c-size-cadr-cad
dr-3-car-if, 355
lessp-plus-lr-p-c-size-lr-p-pc-
1-helper, 65
1-temps, 69
1, 65
lessp-plus-lr-p-pc-1-lr-p-c-size
-3-1-lr-expr-car-if, 359
lessp-plus-remainder-0-fact, 199
lessp-sub1-lessp-fact, 227
lessp-times-difference-fact, 225
lessp-times-difference-node-on-
boundaryp-fact, 225
lessp-times-plus-fact, 227
lessp-x-sub1-facts, 311
list-ascii-0, 12, 14
list-ascii-1, 12
listp-cdr-make-p-call-frame, 74
listp-cdr-p-frame, 3
listp-comp-body, 93
listp-comp-body-1, 57
listp-label-instrs, 93
listp-lr-all-nodes, 273
listp-lr-compile-body, 129
listp-lr-compile-programs, 211

listp-lr-expr-list-s->lr1, 132
listp-lr-expr-s->lr1, 136
listp-p-ctrl-stk-lr-funcall, 90
listp-p-temp-stk-lr-do-temp-fet
ch, 64
listp-p-temp-stk-lr-push-tstk, 63
listp-p-temp-stk-proper-ctrl-st
k-lr-apply-subr, 85
k-p-run-subr, 85
listp-pairlist, 165
listp-plist-car, 348
listp-put-assoc, 119
listp-untag-add-addr, 53
listp-untag-lr-p-pc, 45
listp-untag-lr-return-pc, 54
listp-untag-sub-addr, 54
litatom-car-gensym, 280
litatom-lr-compile-body, 131
litatom-lr-expr-s->lr1, 134
litatom-lr-expr-s->lr1-s-expr, 136
local-var-value, 25, 41, 134, 151, 367
log, 180, 181, 195, 196, 245, 293, 294, 312, 313
logic->lr-ok-really, 332
logic->p, 333, 384
logic->p-clock, 378, 384
logic->p-ok-really, 383
logic->p-ok-really-helper-1, 380
logic->s, 319, 330–333, 378–383
logic-fname, 168, 169
lr->p, 36, 37, 44, 45, 60–64, 73, 75–90, 92–95, 107, 109, 114, 121–125, 130, 131, 134, 135, 137, 141, 144–146, 148, 150, 151, 154–161, 163, 164, 168, 170–176, 182–185, 187, 188, 190–195, 198–207, 210–212, 214–216, 274, 281, 282, 292–294, 308–310, 319, 323, 333, 334, 336–338, 340–343, 345–347, 352, 353, 355, 357–363, 366–378, 380, 381, 383
lr->p-lr-set-pos-dv-1-car-lr-e
xpr-funcall, 370
xpr-if, 353
xpr-temp-eval, 362
lr->p-p-run-subr-p-run-subr-c
lock, 370
lr-0-addr, 6, 27, 75, 76, 79, 80, 99, 102, 161, 218, 219, 265, 270–273, 340
lr-abs, 10, 11
lr-add-to-data-seg, 15, 17
lr-add1-tag, 5, 9, 10, 17, 98, 219, 245, 256, 257, 260, 271, 312
lr-all-nodes, 226, 228, 229, 231–234, 266, 267, 273
lr-all-nodes-lessp-max-addr-ope
ner, 229
lr-all-nodes-nil, 228
lr-all-nodes-not-lessp-min-offset
-max-addr, 232
lr-all-nodes-offset-max-addr-ope
ner-helper, 229
lr-all-nodes-offset-same-max, 229
lr-answer-addr, 6, 15, 33, 59, 230, 231, 233, 309, 310, 316, 318, 379–381, 384
lr-apply-subr, 37, 42, 46, 85, 87–90, 122, 124, 153, 155–158, 160, 164, 165, 168, 189, 208, 214, 343, 374
lr-apply-subr-preserves-lr-p-pr
oper-statep, 343
lr-apply-subr-preserves-lr-prope
r-free-listp, 122
r-heapp, 156
r-heapp2, 154
lr-apply-subr-preserves-lr-valp, 124
lr-boundary-nodep, 7, 54, 55, 102–107, 109, 112–114, 117, 121, 122, 124–127, 130, 153–155, 162, 199, 221–233, 235, 237–242, 249–256, 259, 264–270, 273–279, 289, 315, 317, 320–322, 339, 342
lr-boundary-nodep-add-addr-lr-n
ode-size, 54

lr-boundary-nodep-equal-plus-fa
ct, 113
ct-zero, 113
lr-boundary-nodep-lessp-lr-node
-size-0, 225
-size-1, 225
-size-2, 225
lr-boundary-nodep-lessp-plus-fa
ct, 199
lr-boundary-nodep-not-lessp-fact
-helper, 225
lr-boundary-nodep-sub-addr, 54
lr-boundary-nodep-tag-cons-time
s-lr-node-size, 230
lr-boundary-offsetp, 7, 55, 102, 103, 105, 115, 125, 227–229, 241, 250, 265, 266, 269, 273
lr-boundary-offsetp-difference-
lr-node-size, 269
not-equal-lessp-fact-1, 227
not-equal-lessp-fact-2, 103
lr-boundary-offsetp-equal-plus-
fact, 55
fact-zero, 125
lr-boundary-offsetp-plus, 241
lr-boundary-offsetp-sub1-length
-heap-name, 102
lr-boundary-offsetp-times-lr-no
de-size-anything, 102
lr-boundaryp-nodep-difference-n
ode-size, 227
lr-car-offset, 6, 10, 27, 28, 75, 79, 97, 98, 106, 160, 162, 181, 196, 341
lr-cdr-offset, 6, 10, 27, 76, 80, 97, 98, 106, 160, 162, 181, 196, 341
lr-check-f-addrp, 97, 125, 126
lr-check-f-addrp-deposit-a-list, 126
lr-check-f-addrp-deposit-anythi
ng-anywhere, 125
lr-check-f-addrp-lr-undef-addr-
lr-init-data-seg, 264
lr-check-free-nodes, 96, 115, 118, 119, 229, 231, 232, 234
lr-check-free-nodes-delete-depo
sit, 118
lr-check-free-nodes-deposit-a-li
st-lr-nodep, 231
lr-check-free-nodes-deposit-free
-ptr, 118
lr-check-free-nodes-deposit-lr-
nodep, 119
lr-check-free-nodes-deposit-non
-ref-count, 115
lr-check-free-nodes-lr-free-list
-nodes-init-data-seg, 230
lr-check-free-nodes-plist-node-
list, 229
lr-check-listp-addrp, 97, 98, 125, 127
lr-check-listp-addrp-deposit-a-
list-cons, 127
list-other-place, 127
lr-check-listp-addrp-deposit-free
-ptr-0, 125
lr-check-numberp-addrp, 97, 98, 125, 127, 257
lr-check-numberp-addrp-deposit-
a-list-cons, 126
a-list-cons-same-addr, 256
free-ptr-0, 125
lr-check-resourcesp, 182, 191–195, 207, 211, 214–216
lr-check-resourcesp-funcall, 195
lr-check-resourcesp-list-set-exp
r-nx, 191
lr-check-resourcesp-listp-s-exp
r-list, 182
lr-check-resourcesp-lr-funcall-
s-fun-call-state, 213
lr-check-resourcesp-lr-funcall-p
-psw-run, 210
lr-check-resourcesp-lr-pop-tstk
-lr-eval-1, 192
-lr-eval-2, 193
lr-check-resourcesp-lr-push-tst
k-flag-run, 191

lr-check-resourcesp-s-set-pos-i
f-cadr, 191
lr-check-resourcesp-s-temp-eval, 194
lr-check-resourcesp-s-temp-test, 194
lr-check-result, 99, 131, 136, 137, 148, 151, 152, 154–156, 164, 165, 171, 174–176, 190, 198, 200–204
lr-check-result-f-not-lr-f-addr, 137
lr-check-result-flag-list-cons-v
alue, 131
lr-check-result-lr-apply-subr, 164
lr-check-result-lr-do-temp-fetc
h, 151
lr-check-result-lr-funcall, 173
lr-check-result-lr-proper-heapp, 175
lr-check-result-lr-push-tstk, 135
lr-check-result-lr-push-tstk-qu
ote, 152
lr-check-result-nil, 131
lr-check-result-not-f-lr-f-addr, 137
lr-check-result-t-chain, 137
lr-check-result1, 99, 100, 112, 130, 153, 154, 159, 160, 163, 164, 166, 167, 325
lr-check-result1-append, 112
lr-check-result1-append-2, 166
lr-check-result1-butlast, 166
lr-check-result1-first-n-temp-st
k, 167
lr-check-result1-lr-good-pointe
rp-get-n-lessp-cadr, 154
rp-get-n-lessp-car, 153
lr-check-result1-lr-valp-get-n-
lessp-length, 153
lr-check-result1-reverse-length
-1-opener, 160
-2-opener, 160
lr-check-result1-singleton-list
-opener, 130
lr-check-undef-addrp, 97, 125, 126, 264
lr-check-undef-addrp-deposit-a-
list, 126
lr-check-undef-addrp-deposit-an
ything-anywhere, 125
lr-compile-body, 19, 20, 22, 128, 129, 131, 132, 155, 216, 296–300, 303–305, 318
lr-compile-programs, 20, 22, 101, 112, 128, 132, 169, 171, 187, 208, 211, 305, 318
lr-compile-quote, 16, 18, 219–224, 237, 238, 240–245, 247, 251–255, 259, 261, 262, 264, 265, 267–273, 275, 280, 311, 313, 317, 325, 326
lr-compile-quote-flag-list-cons
-opener, 252
lr-compile-quote-flag-list-nil-
opener, 252
lr-compile-quote-lr-good-pointe
rp-tablep, 253
rp-tablep-help-1, 251
rp-tablep-help-2, 252
rp-tablep-help-3, 253
lr-compile-quote-preserves-lr-p
roper-heapp, 261
roper-heapp2, 259
lr-compile-quote-preserves-lr-v
alp, 260
lr-compile-quote-preserves-prope
r-p-data-segmentp, 313
lr-cons-tag, 5, 9, 10, 14, 27–29, 75–77, 79–83, 98, 106, 127, 128, 160–162, 180, 196, 245, 260, 312, 341
lr-convert-digit-to-ascii, 31, 283, 284
lr-convert-digit-to-ascii-equal, 283
lr-convert-num-to-ascii, 31, 283, 284
lr-convert-num-to-ascii-equal-a
rg1, 283
rg2, 284
rg2-helper-1, 284
rg2-lengths, 284
rg2-lengths-helper-1, 284
lr-count-free-nodes, 181, 182, 185–189, 191, 198–207, 248–253,

255, 259, 261–263, 267, 273, 275–280, 294, 295, 311, 313–315, 320–323, 325–328
lr-count-free-nodes-append-lr-all-nodes-fact, 273
lr-count-free-nodes-at-most, 248
lr-count-free-nodes-delete-deposit, 186
lr-count-free-nodes-deposit-a-list-lr-nodep, 249
lr-count-free-nodes-deposit-free-ptr, 185
lr-count-free-nodes-deposit-lr-nodep, 186
lr-count-free-nodes-deposit-non-ref-count, 185
lr-count-free-nodes-lr-all-nodes, 266
lr-count-free-nodes-lr-compile-q
  uote-s-heap-reqs, 250
  uote-s-heap-reqs-flag-t, 274
  uote-s-heap-reqs-help1, 250
lr-count-free-nodes-lr-data-seg
  -table-body-s-heap-reqs, 275
  -table-list-s-heap-reqs, 321
  -table-list-s-heap-reqs-1, 322
  -table-list-s-heap-reqs-help, 320
lr-count-free-nodes-lr-init-dat
  a-seg-table-s-heap-reqs-1, 322
  a-seg-table-s-init-heap-reqs, 279
lr-count-free-nodes-max-addr-lr-free-list-nodes, 186
lr-count-free-nodes-s-init-heap-reqs, 262
lr-count-lr-free-list-nodes-lr-apply-subr, 188
lr-count-lr-free-list-nodes-p-r
  un-cons, 186
  un-subr, 187
lr-data-seg-body-list-n, 300
lr-data-seg-table, 18, 22, 273, 274, 280, 281, 295, 302, 304–306, 318–320, 323, 324, 328, 380–382
lr-data-seg-table-body, 17, 18, 220, 221, 223, 238, 243, 244, 246, 274–278, 294, 300, 301, 314, 321, 326, 327
lr-data-seg-table-body-add1-opener, 301
lr-data-seg-table-body-flag-t-flag-t, 301
lr-data-seg-table-body-lr-good-p
  ointerp-tablep, 276
  ointerp-tablep-help1, 276
lr-data-seg-table-body-n, 300, 301
lr-data-seg-table-body-preserve
  s-lr-valp, 326
  s-proper-p-data-segmentp, 314
lr-data-seg-table-list, 18, 19, 220, 221, 224, 234, 239, 279, 295, 302, 314, 317, 321, 322, 327, 328
lr-data-seg-table-list-lr-good-p
  ointerp-tablep-helper-1, 277
lr-data-seg-table-list-preserve
  s-lr-s-similar-params, 327
  s-lr-valp, 326
  s-proper-p-data-segmentp, 314
lr-do-temp-fetch, 25, 41–44, 63, 64, 108, 139, 140, 149, 151, 194, 367, 369
lr-do-temp-fetch-lr-check-resourcesp-temp-test, 194
lr-do-temp-fetch-run-lr-eval-temp-setp, 151
lr-eval, 40–42, 44, 46, 47, 86, 87, 90, 92–95, 110, 112, 122–125, 128, 130, 131, 133, 134, 136–142, 144–146, 148, 152–154, 156–160, 163, 164, 167–176, 183–185, 187, 188, 190–193, 205–208, 210, 211, 214–216, 320, 324, 329, 333–335, 337, 340, 341, 343, 345–347, 352, 353, 355, 357–363, 367, 368, 373–378, 381
lr-eval-if-p-psw-1, 44
lr-eval-leaves-listp-p-ctrl-stk

-lr->p-lr-set-pos, 337
lr-eval-leaves-listp-p-temp-stk, 137
-lr-set-pos, 176
lr-eval-litatom-opener, 134
lr-eval-p-pc-equivalence, 378
lr-eval-p-pc-equivalence-helper
-1, 352
-1-5, 352
-10, 369
-11, 374
-12, 377
-12-helper-1, 375
-12-helper-2, 376
-2, 352
-3, 352
-4, 358
-4-helper-1, 355
-4-helper-2, 357
-5, 361
-5-get-lr-p-c-size, 362
-5-helper-1, 360
-5-helper-2, 360
-6, 363
-6-helper, 362
-7, 366
-8, 368
-8-helper-1, 367
-8-helper-2, 367
-8-helper-2-helper, 367
-9, 369
lr-eval-preserves-adpp, 123
lr-eval-preserves-adpp-lr-set-p
os, 123
lr-eval-preserves-cdr-p-ctrl-st
k, 122
k-lr-funcall, 375
k-lr-set-pos, 123
lr-eval-preserves-definedp-first
n-bindings-car-p-ctrl-stk, 141
lr-eval-preserves-definedp-fn-bi
ndings-car-ctrl-stk-set-pos, 141
lr-eval-preserves-length-assoc-
data-segment, 123
lr-eval-preserves-length-bindin
gs-car-p-ctrl-stk, 183
lr-eval-preserves-lr-max-node, 122
lr-eval-preserves-lr-max-node-l
r-set-pos, 124
lr-eval-preserves-lr-p-proper-st
atep, 346
lr-eval-preserves-lr-proper-free
-listp, 124
-listp-lr-set-pos, 124
lr-eval-preserves-lr-proper-heapp, 182
-lr-set-pos, 184
lr-eval-preserves-lr-s-similar-
const-table, 171
statesp, 183
statesp-lr-set-pos, 184
lr-eval-preserves-lr-s-similar-p
arams, 172
lr-eval-preserves-lr-s-similar-te
mps, 172
lr-eval-preserves-lr-valp, 125
lr-eval-preserves-lr-valp-lr-set
-expr, 130
lr-eval-preserves-proper-p-statep
-lr->p, 95
-lr->p-lr-set-expr, 130
-lr->p-lr-set-pos, 123
-lr->p-rewrite, 122
lr-eval-preserves-ret-pc-car-p-
ctrl-stk, 376
ctrl-stk-lr-funcall, 376
lr-eval-preserves-strip-cars-bi
ndings-car-p-ctrl-stk, 122
ndings-car-p-ctrl-stk-lr-set-pos,
123
lr-eval-preserves-strip-cars-lr
-temps-car-p-ctrl-stk, 148
lr-eval-s->lr1-flag-list-opene
r-1, 175
r-2, 175
lr-eval-s->lr1-if-opener-1, 137
lr-eval-s->lr1-if-opener-2, 138
lr-eval-s->lr1-if-opener-3, 138
lr-eval-s->lr1-preserves-p-ct
rl-stk-size, 184

rl-stk-size-lr-set-pos, 192
lr-eval-s->lr1-quote-opener, 140
lr-eval-s->lr1-temp-eval-opener, 139
lr-eval-s->lr1-temp-fetch-opener, 140
lr-eval-s->lr1-temp-test-opener, 139
lr-eval-s-eval-equivalence, 176
lr-eval-s-eval-equivalence-lr-check-result-flag-list, 190
lr-eval-s-eval-equivalence-s->lr, 319
lr-eval-s-eval-flag-run, 216
lr-eval-s-eval-flag-run-s->lr, 323
lr-eval-s-eval-flag-t-s-ans-f-lr-set-pos, 185
lr-eval-s-eval-flag-t-s-ans-non-f-lr-set-pos, 185
lr-eval-s-eval-heap-r-lr-count-lr-free-list-nodes, 190
lr-eval-subrp-user-funcall-opener, 152
lr-eval-t-lr-funcall-p-psw-run, 92
lr-eval-temp-setp, 25, 41, 111, 139, 150, 151, 335, 368
lr-eval-zerop-clock, 133
lr-expr, 7, 25, 37, 40–42, 44, 57, 61–64, 72, 73, 84–95, 114, 121, 122, 124, 134, 136, 137, 139, 140, 142, 144–146, 148, 150–160, 164, 165, 168, 169, 195, 206, 212, 333–338, 340–343, 345–347, 349–371, 373–377
lr-expr-cur-expr-if-same, 369
lr-expr-flag-list-car, 40
lr-expr-funcall, 375
lr-expr-list, 7, 40, 41, 58, 94, 131, 132, 170, 334, 349, 351, 352
lr-expr-list-lr-set-pos-dv-1, 40
lr-expr-lr-set-expr, 40
lr-expr-lr-set-expr-nx, 40
lr-expr-lr-set-pos-t, 40
lr-f-addr, 6, 15, 18, 26, 28, 29, 31, 41, 76–79, 81–84, 97, 99, 102, 111, 137, 138, 161, 185, 219, 225, 234, 244, 245, 247, 256, 265, 266, 269, 271, 272, 280, 316, 334, 340, 351, 354, 355, 357, 358, 360, 361
lr-false-tag, 5, 9, 10, 15, 30, 97, 98, 219, 245, 269, 272, 293, 316
lr-fetch-fp, 6, 8–10, 96
lr-fix-data-segment, 9, 10
lr-fp-addr, 6, 15, 17, 77, 78, 80, 81, 96, 104, 107, 109, 118, 121, 125, 162, 182, 185–191, 198–207, 220, 221, 225, 230, 231, 233, 235, 240–242, 249–255, 259–265, 268, 269, 272, 275–280, 294, 295, 311–316, 320–328, 338, 339, 341
lr-free-list-nodes, 96, 102, 103, 115–119, 182, 186–189, 191, 198–207, 231, 233, 249–253, 255, 259, 261–263, 268, 273, 275–280, 294, 295, 311, 313–315, 320–323, 325–328
lr-free-list-nodes-deposit-0, 267
lr-free-list-nodes-deposit-a-list-lr-nodep, 231
lr-free-list-nodes-deposit-free-ptr, 118
lr-free-list-nodes-deposit-lr-nodep, 119
lr-free-list-nodes-deposit-lr-ref-count-offset, 116
lr-free-list-nodes-deposit-non-ref-count, 115
lr-free-list-nodes-deposit-t, 268
lr-free-list-nodes-lr-compile-quote-t0, 272
lr-free-list-nodes-lr-init-data-seg, 233
lr-free-list-nodes-lr-init-heap-contents, 233
-contents-generalized, 232
lr-free-list-nodes-member-greate

r-offset, 116
lr-funcall, 37, 42, 43, 90–95, 112, 153, 165, 169–175, 211, 212, 214, 335, 345, 346, 375–377
lr-good-pointerp, 7, 55, 97, 98, 106, 112, 127, 128, 137, 153, 160–162, 196, 241, 255–258, 336–339
lr-good-pointerp-cdr-assoc-car-lr-compile-quote-list, 254
lr-good-pointerp-deposit-a-list-node, 339
lr-good-pointerp-deposit-non-add-addr-not-good-pointerp, 256
lr-good-pointerp-deposit-non-ref-not-good-pointerp, 255
lr-good-pointerp-deposit-ref-count-not-good-pointerp, 255
lr-good-pointerp-lessp-offset-max-heap-node, 106
lr-good-pointerp-lr-undef-addr, 258
lr-good-pointerp-opener, 55
lr-good-pointerp-table-cons, 242
lr-good-pointerp-tablep, 241, 242, 253–255, 259–261, 269, 277–279, 325–327, 336, 338
lr-good-pointerp-tablep-definedp-table, 241
lr-good-pointerp-tablep-deposit
-a-list, 242
-free-ptr, 241
lr-good-pointerp-tablep-f-lr-f-addr-lr-init-data-seg, 269
lr-good-pointerp-type-tag-nat, 161
lr-good-pointerps, 336–345, 366, 382
lr-good-pointerps-all-undef-addrs, 344
lr-good-pointerps-append, 344
lr-good-pointerps-cdr, 340
lr-good-pointerps-cons-fetch-ca
r-temp-stk-cdr-car, 341
r-temp-stk-cdr-cdr, 341
lr-good-pointerps-cons-fetch-fp-addr-deposit-a-list-cons, 341
lr-good-pointerps-cons-lr-0-addr-lr-proper-heapp, 340
lr-good-pointerps-cons-lr-f-addr-lr-proper-heapp, 340
lr-good-pointerps-cons-lr-t-addr-lr-proper-heapp, 340
lr-good-pointerps-deposit-a-list-node, 339
lr-good-pointerps-deposit-free-ptr, 338
lr-good-pointerps-first-n, 344
lr-good-pointerps-popn, 345
lr-good-pointerps-put-assoc, 337
lr-good-pointerps-reverse, 344
lr-heap-name, 6–10, 15, 55, 96, 102–104, 107, 121, 126, 127, 130, 153, 154, 199, 221–226, 228–234, 237–242, 248–256, 259–261, 264, 266–270, 272–279, 289, 311–313, 315–317, 320–322
lr-if-ok, 24, 41, 43, 95, 110, 136, 138, 139, 175, 176, 192, 193, 215, 334, 353, 355, 357, 358, 360, 361
lr-init-data-seg, 15, 18, 225, 226, 233, 234, 244, 245, 247, 263–267, 269–273, 280, 316, 317, 320
lr-init-data-seg-table, 18, 221, 224, 234, 240, 245, 247, 263, 267, 278–280, 295, 315, 317, 323, 325, 326, 328
lr-init-data-seg-table-lr-good-pointerp-tablep, 278
lr-init-data-seg-table-preserve
s-lr-valp, 325
s-proper-p-data-segmentp, 314
lr-init-heap-contents, 14, 15, 217, 218, 226, 230–233, 315, 316
lr-init-heap-contents-add1-opener, 218
lr-init-tag, 5, 14, 218
lr-initial-cstk, 19, 22
lr-make-initial-temps, 19, 259, 274,

281, 324
lr-make-label, 31, 32, 71, 72, 85, 187, 284, 309, 347, 351, 354, 356, 357, 359, 364–366
lr-make-label-equal, 284
lr-make-label-not-numberp, 71
lr-make-program, 11, 20, 33
lr-make-temp-name-alist, 14, 20, 22, 128, 132, 133, 145–148, 150, 151, 209, 280, 281, 305, 318
lr-make-temp-name-alist-1, 14, 143, 145, 147, 148, 151, 171, 208, 280, 281
lr-make-temp-name-alist-1-plist
-arg-1, 281
lr-make-temp-name-alist-plist-a
rg-1, 281
lr-make-temp-var-dcls, 20, 132, 133, 135, 171, 208, 292
lr-max-ctrl-reqs, 319, 332, 379, 383
lr-max-node, 96, 99, 102, 104, 107, 109, 112, 114, 116, 121, 122, 124, 125, 130, 154, 155, 162, 182, 186–189, 191, 198–205, 207, 220–225, 230, 235, 237–242, 249–255, 257–264, 267, 268, 274–280, 294, 295, 311, 313–315, 317, 320–323, 325–328, 342
lr-max-node-adpp-definedp-lr-he
ap-name, 104
lr-max-node-car-lr-compile-quote, 224
lr-max-node-car-lr-data-seg-tab
le, 320
le-body, 223
le-list, 322
lr-max-node-car-lr-init-data-se
g-table, 267
lr-max-node-deposit, 116
lr-max-node-deposit-a-list, 220
lr-max-node-lr-init-data-seg, 225
lr-max-node-lr-nodep-opener-fact
s, 104
lr-max-node-same-signature, 116
lr-max-temp-reqs, 319, 332, 379, 383
lr-max-word-size-reqs, 319, 332, 379, 383
lr-minimum-heap-size, 6, 103, 264, 265, 273
lr-minimum-heapp, 99, 102, 103, 107, 108, 161, 162, 242, 248, 251–253, 255, 258–260, 263, 274–279, 315, 320, 321
lr-minimum-heapp-lr-compile-quote, 242
lr-minimum-heapp-lr-data-seg-ta
ble-body, 274
lr-minimum-heapp-lr-init-data-se
g, 263
lr-minimum-heapp-not-equal-lengt
h-1, 248
lr-minimum-heapp-opener-2, 103
lr-minimum-heapp-opener-3, 103
lr-minimum-heapp-opener-adpp-lr
-0-addr, 102
-f-addr, 102
-t-addr, 102
-undef-addr, 102
lr-minimum-heapp-same-signature, 108
lr-minus-tag, 6
lr-negative-guts-offset, 7, 11
lr-new-cons, 14, 17
lr-new-node, 6, 14, 15, 17, 218
lr-node-listp, 103, 107, 115, 118, 119, 185, 186, 232, 233, 249
lr-node-listp-delete, 107
lr-node-listp-deposit-anything-
at-all, 107
lr-node-listp-lr-all-nodes, 233
lr-node-listp-lr-free-list-node
s, 103
lr-node-size, 6, 7, 9, 14, 15, 54, 55, 96–99, 102, 103, 105, 113–115, 119, 125, 126, 185, 199, 200, 217, 218, 225–230, 232–234, 250, 253, 255, 256, 264–266, 268–270, 272, 273, 312, 313, 315, 316, 320

lr-nodep, 7, 55, 97, 99, 103, 105, 106, 112–115, 118, 119, 128, 154, 155, 161, 162, 185, 186, 254, 257–260, 342
lr-nodep-car-lr-compile-quote, 254
lr-nodep-deposit-a-list, 254
lr-nodep-lr-proper-heapp-nodep, 105
lr-nodep-member-lr-node-listp, 103
lr-nodep-member-lr-node-listp-a
dpp-untag-listp, 115
dpp-untag-numberp-offset, 115
lr-nodep-member-lr-node-listp-l
r-boundaryp-offsetp, 115
lr-nodep-opener, 55
lr-nodify, 9, 10
lr-nodify-tag, 9
lr-p-c-size, 20, 21, 32, 35, 37, 57, 58, 60, 65–72, 286, 288, 292, 348–352, 354–362, 364–367, 375, 377, 380
lr-p-c-size-flag-list, 58
lr-p-c-size-flag-not-list-not-0, 35
lr-p-c-size-ge-plus-2-size-cadr
-car-if, 350
lr-p-c-size-list, 21, 57–60, 65, 66, 69–71, 84, 348, 349, 352, 371, 375, 377
lr-p-c-size-list-0, 59
lr-p-c-size-list-0-opener, 57
lr-p-c-size-list-add1-opener, 57
lr-p-c-size-list-car-opener, 69
lr-p-c-size-list-funcall-not-le
ssp-fact, 66
lr-p-c-size-nlistp-body, 67
lr-p-c-size-not-1-car-if, 350
lr-p-c-size-s-temp-test-eval-ca
dr-not-lessp-fact, 66
lr-p-c-size-temp-test-opener, 364
lr-p-pc, 21, 36, 37, 44, 45, 61–64, 90, 347, 351, 369–371, 375, 377
lr-p-pc-1, 21, 22, 60, 65, 67–72, 347–350, 352, 354–362, 364–367, 369
lr-p-pc-1-append, 348
lr-p-pc-1-append-helper-1, 348
lr-p-pc-1-append-helper-2, 348
lr-p-pc-1-append-helper-3, 348
lr-p-pc-1-append-helper-4, 348
lr-p-pc-1-body-0, 60
lr-p-pc-1-car-expr-if-2, 356
lr-p-pc-1-dv-1-car-lr-expr-if, 350
lr-p-pc-1-dv-1-car-lr-expr-temp
-eval, 362
lr-p-pc-1-dv-2-car-lr-expr-if, 350
lr-p-pc-1-dv-3-car-lr-expr-if, 350
lr-p-pc-1-listp-offset, 349
lr-p-pc-1-nil, 349
lr-p-pc-1-nx, 349
lr-p-pc-1-nx-helper, 349
lr-p-pc-1-plist, 348
lr-p-pc-dv-1-s-temp-test, 367
lr-p-pc-lr-do-temp-fetch, 64
lr-p-pc-lr-pop-tstk, 62
lr-p-pc-lr-push-tstk, 61
lr-p-pc-lr-set-pos-dv-1-car-lr-e
xpr-funcall, 369
lr-p-pc-lr-set-temp, 63
lr-p-proper-statep, 336–338, 343, 345–347, 353, 355, 358–361, 366, 368, 378, 380, 382
lr-p-proper-statep-cdr-assoc-ca
ddr-lr-expr-bindings, 337
lr-p-proper-statep-cdr-lr-ctrl-
stk, 346
lr-p-proper-statep-cdr-temp-stk, 337
lr-p-proper-statep-cons-p-frame
-put-assoc, 337
lr-p-proper-statep-listp-p-temp
-stk-type-car-addr, 353
lr-p-proper-statep-lr-funcall, 345
lr-p-proper-statep-lr-good-pointe
rps-strip-cdrs-binding, 366
lr-p-proper-statep-lr-push-tstk
-cdr-assoc-lr-expr, 336
lr-p-proper-statep-p-temps-stk-
lr-push-tstk-quote, 338
lr-p-proper-statep-s->lr, 382

lr-pack-tag, 6, 9, 10
lr-params, 23, 95, 100, 108, 131, 133,
135, 136, 140, 142, 149, 157,
165, 172
lr-params-lr-apply-subr, 157
lr-params-lr-do-temp-fetch, 149
lr-params-lr-eval, 95
lr-params-lr-funcall, 165
lr-params-lr-pop-tstk, 108
lr-params-lr-push-tstk, 131
lr-params-lr-set-error, 133
lr-params-lr-set-expr, 108
lr-params-lr-set-expr-lr-pop-cst
k, 172
lr-params-lr-set-pos, 136
lr-params-lr-set-temp, 140
lr-params-p-frame-not-definedp-p
ut-assoc-anything, 141
lr-pop-cstk, 25, 42, 44, 94, 153, 172,
173, 377, 378
lr-pop-tstk, 24, 39, 41, 62, 94, 108,
111, 136, 138, 175, 192–194,
334, 335, 353, 355, 357, 358,
360, 361
lr-pop-tstk-lr-if-ok, 353
lr-programs-properp, 35, 47, 51, 56,
57, 61–64, 72, 73, 75–95,
114, 121–125, 130, 131, 135,
137, 141, 144, 145, 148, 150,
152–160, 163, 164, 168–176,
182–185, 187, 188, 190–195,
198, 200–208, 210–212, 214–
216, 282, 292–294, 306, 309,
310, 319, 320, 323, 336–339,
341–347, 349, 350, 352–378,
380
lr-programs-properp-1, 35, 56, 57,
63, 305–308, 344
lr-programs-properp-1-all-user-
fnamesp-not-user-fnamep, 51
lr-programs-properp-1-lr-compile
-programs, 305
lr-programs-properp-1-lr-proper
-exprp, 56
lr-programs-properp-all-user-fn
amesp-strip-cars-cdr, 309
lr-programs-properp-caar-main, 309
lr-programs-properp-definedp-ca
r-untag-p-pc, 61
lr-programs-properp-definedp-su
brp-runtime-support, 206
lr-programs-properp-expr-quote-t
ype-addr, 64
lr-programs-properp-funcall-not
-caar-prog-seg, 169
lr-programs-properp-lr->p-s->
lr1-definedp-s-pname, 145
lr-programs-properp-lr-eval, 47
lr-programs-properp-lr-funcall, 94
lr-programs-properp-lr-if-ok, 95
lr-programs-properp-lr-pop-tstk, 94
lr-programs-properp-lr-programs
-properp-1, 56
lr-programs-properp-lr-proper-e
xprp-lr-expr, 57
lr-programs-properp-lr-set-erro
r, 175
lr-programs-properp-lr-set-expr, 94
lr-programs-properp-lr-set-pos, 90
lr-programs-properp-member-lr-e
xpr-temps, 150
lr-programs-properp-not-definedp
-subrp-runtime-support, 163
lr-programs-properp-s->lr, 319
lr-programs-properp-s->lr-logi
c->s, 380
lr-programs-properp-s->lr-ope
ned, 306
lr-programs-properp-s->lr1-de
finedp-cdr-s-progs, 208
finedp-s-progs, 212
lr-proper-ctrl-stkp, 336, 339, 340
lr-proper-ctrl-stkp-deposit-a-li
st-node, 339
lr-proper-ctrl-stkp-deposit-free
-ptr, 339
lr-proper-exprp, 34, 35, 56–58, 62,

65, 67–72, 152, 287–292, 296–300, 303–305, 307, 348, 349
lr-proper-exprp-cadr-temps, 68
lr-proper-exprp-car-if-cadddr, 58
lr-proper-exprp-car-if-caddr, 58
lr-proper-exprp-car-if-cadr, 58
lr-proper-exprp-flag-list-cdr-f
uncall, 290
lr-proper-exprp-flag-list-cons, 296
lr-proper-exprp-flag-list-nil, 296
lr-proper-exprp-flag-not-list-c
ons-funcall, 303
ons-if, 297
ons-if-helper, 296
ons-temp-eval, 298
ons-temp-fetch, 298
ons-temp-test, 299
lr-proper-exprp-flag-not-list-n
ot-listp, 304
lr-proper-exprp-length-cur-expr, 57
lr-proper-exprp-list-lr-proper-
get-t, 56
lr-proper-exprp-list-quote-opene
r, 152
lr-proper-exprp-lr-proper-exprp
-cur-expr, 56
lr-proper-exprp-p-lr-compile-pr
ograms, 304
ograms-flag-t, 305
ograms-helper-1, 296
ograms-helper-2, 302
lr-proper-exprp-program-body-not
-listp, 307
lr-proper-exprp-t-lr-proper-get
-t, 56
lr-proper-formalsp, 168–170, 343, 345, 347, 353, 355, 358–361, 377, 378, 381, 382
lr-proper-formalsp-cdr-p-prog-se
gment-s->lr-logic->s, 382
lr-proper-formalsp-lr-compile-p
rograms, 169
lr-proper-free-listp, 96, 99, 104, 107, 109, 114, 121, 122, 124, 125, 130, 154, 155, 162, 221–224, 234, 235, 237–239, 241, 242, 249–254, 259, 260, 264, 267, 268, 274–279, 317, 320–322, 342
lr-proper-free-listp-car-lr-compi
le-quote, 222
lr-proper-free-listp-car-lr-init
-data-seg-table, 224
lr-proper-free-listp-length-sub
1-not-lessp, 242
lr-proper-free-listp-lr-count-f
ree-nodes-max-addr, 249
ree-nodes-max-addr-alt, 250
lr-proper-free-listp-lr-init-dat
a-seg, 234
a-seg-helper, 234
lr-proper-free-listp-member-free
-addr-lr-free-list-nodes, 249
lr-proper-free-listp-opener-1, 104
lr-proper-free-listp-opener-2, 104
lr-proper-free-listp-opener-2-a
dpp-untag-listp, 104
dpp-untag-numberp-offset, 104
rea-name-alt, 241
lr-proper-free-listp-opener-2-l
r-nodep, 254
lr-proper-free-listp-type-fetch
-free-ptr, 109
lr-proper-heapp, 99, 100, 107, 112, 131, 135, 137, 143, 149–152, 156–158, 161–164, 171–173, 175, 176, 182–191, 193–196, 200, 205–207, 211, 214, 216, 259–263, 271, 274, 278–282, 284, 286, 288, 289, 292–295, 307–315, 322, 324–328, 336, 338, 340, 341, 343, 344
lr-proper-heapp-car-lr-data-seg
-table, 280
-table-body, 278
-table-helper-1, 280
-table-list, 278
lr-proper-heapp-car-lr-init-dat

a-seg-table, 279
lr-proper-heapp-lr-compile-quote
-ft-lr-init-data-seg, 271
lr-proper-heapp-lr-good-pointerp
-lr-proper-heapp-nodep, 137
lr-proper-heapp-lr-valp-0, 161
lr-proper-heapp-lr-valp-f, 161
lr-proper-heapp-lr-valp-f-helpe
r, 161
lr-proper-heapp-lr-valp-lr-f-ad
dr, 161
lr-proper-heapp-lr-valp-lr-t-ad
dr, 161
lr-proper-heapp-nodep, 97, 98, 105, 106, 125, 128, 137, 161, 257, 258, 269
lr-proper-heapp-nodep-deposit-a
-list-cons, 128
-list-numberp, 257
-list-truep, 258
lr-proper-heapp-nodep-deposit-f
ree-ptr-0, 125
lr-proper-heapp-nodep-lr-init-d
ata-seg-helper, 269
lr-proper-heapp-nodep-lr-undef-
addr-lr-init-data-seg, 269
lr-proper-heapp-nodep-tag-cons, 106
lr-proper-heapp-opener-1, 107
lr-proper-heapp-opener-3, 107
lr-proper-heapp-opener-4, 112
lr-proper-heapp1, 98, 99
lr-proper-heapp2, 98, 99, 105, 107, 125, 128, 154, 155, 257–259, 261, 270, 342
lr-proper-heapp2-deposit-a-list
-cons, 128
-numberp, 257
-truep, 258
lr-proper-heapp2-deposit-free-pt
r-0, 125
lr-proper-heapp2-lr-init-data-se
g, 270
g-helper, 270
lr-proper-p-areasp, 42, 101, 102, 104, 107, 109, 116, 120, 121, 137, 143, 149, 150, 161, 162, 196, 200, 219, 221–224, 234, 237–242, 249–254, 256–265, 267, 268, 274–279, 281, 284, 286, 288, 289, 292–295, 307–318, 320–322, 324–328, 340, 341, 344
lr-proper-p-areasp-car-lr-compi
le-quote, 219
lr-proper-p-areasp-car-lr-data-
seg-table, 281
seg-table-body, 223
seg-table-list, 234
lr-proper-p-areasp-car-lr-init-
data-seg-table, 234
lr-proper-p-areasp-deposit-a-li
st, 120
lr-proper-p-areasp-deposit-anyt
hing-anywhere, 107
lr-proper-p-areasp-lr-heap-name
-lr-init-data-seg, 234
lr-push-tstk, 24, 25, 41–43, 60–64, 73, 108, 131, 134–136, 140, 152, 167, 191, 195, 336, 338
lr-push-tstk-length, 167
lr-push-tstk-lr-check-resourcesp
-quote, 195
lr-ref-count-offset, 6, 7, 15, 28, 55, 77, 81, 96, 97, 102, 106, 109, 110, 112–115, 117–119, 121, 128, 153, 154, 181, 185, 186, 219, 230–232, 240, 242, 250, 252, 255–258, 264–266, 268–272, 289
lr-return-pc, 37, 53, 54, 85, 86, 91, 93, 114, 121, 154, 159, 168, 187, 206, 333, 334, 338, 340, 342, 343, 374, 376
lr-s-similar-const-table, 100, 101, 151, 158, 171, 218, 240, 241, 259–263, 272, 289, 292, 294, 295, 307–311, 313, 314, 324–326,

328
lr-s-similar-const-table-cdr-ca
r-lr-data-seg-table, 295
lr-s-similar-const-table-compile
-quote-t0-init-data-seg, 272
lr-s-similar-const-table-cons, 218
lr-s-similar-const-table-deposit
-cons, 259
-lr-fp-addr, 240
lr-s-similar-const-table-implie
s-lr-good-pointerp-tablep, 259
lr-s-similar-const-table-lr-app
ly-subr, 157
lr-s-similar-const-table-lr-compi
le-quote, 261
lr-s-similar-const-table-lr-dat
a-seg-table-body, 294
a-seg-table-list, 294
lr-s-similar-const-table-lr-goo
d-pointerp-opener, 240
lr-s-similar-const-table-lr-init
-data-seg-table, 295
lr-s-similar-const-table-lr-valp
-assoc, 151
-member-strip-cdrs, 289
lr-s-similar-const-table-nil, 218
lr-s-similar-const-table-p-obje
ctp-definedp, 311
lr-s-similar-const-table-type-a
ddr-member-strip-cdrs, 289
lr-s-similar-params, 100, 101, 134, 135, 157, 166, 167, 172, 326–328, 382
lr-s-similar-params-assoc-define
dp, 134
lr-s-similar-params-lr-apply-su
br, 156
lr-s-similar-params-lr-funcall, 167
lr-s-similar-params-lr-funcall-
helper-1, 167
lr-s-similar-params-lr-good-poi
nterps-strip-cdrs, 382
lr-s-similar-params-lr-valp-get, 166
lr-s-similar-params-pair-formal
s-lr-init-data-seg, 326
s-with-addresses, 328
lr-s-similar-statesp, 100, 133–136, 144, 146, 148–152, 158, 172, 173, 176, 183–185, 188–191, 193, 194, 205–207, 214, 216, 320, 324, 328, 381
lr-s-similar-statesp-lr-apply-s
ubr, 158
lr-s-similar-statesp-lr-funcall, 171
lr-s-similar-statesp-lr-if-ok, 176
lr-s-similar-statesp-lr-pop-tst
k, 136
lr-s-similar-statesp-lr-push-tst
k-litatom, 134
lr-s-similar-statesp-lr-s-set-p
os, 136
lr-s-similar-statesp-lr-s-simil
ar-params-opener, 135
ar-temps-opener, 135
lr-s-similar-statesp-lr-set-err
or, 133
lr-s-similar-statesp-lr-set-exp
r, 133
r-lr-pop-cstk, 173
lr-s-similar-statesp-s->lr-lo
gic->s-lr-data-seg-table, 381
lr-s-similar-statesp-s->lr-lr
-data-seg-table, 328
lr-s-similar-statesp-s->lr1-l
r-similar-temps, 146
lr-s-similar-statesp-s-change-te
mp, 148
mp-helper-1, 146
mp-helper-2, 144
lr-s-similar-temps, 100, 101, 135, 143, 144, 146, 149, 150, 157, 171, 173, 324
lr-s-similar-temps-lr-apply-sub
r, 157
lr-s-similar-temps-lr-funcall, 171
lr-s-similar-temps-make-temps-e
ntries-initial-temps, 324
lr-s-similar-temps-make-temps-p

air-temps, 170
lr-s-similar-temps-put-assoc-put
-assoc-helper, 143
-assoc-helper-1, 143
lr-s-simlar-statesp-lr-do-temp-fetch, 149
lr-set-error, 23–25, 39, 41, 42, 107, 133, 136, 137, 140, 153, 175
lr-set-error-lr->p, 107
lr-set-error-lr-set-error, 133
lr-set-expr, 23, 38, 40–42, 94, 95, 108, 130, 131, 133, 136, 138, 153, 172, 173, 175, 334, 335, 347, 352, 355–358, 360, 361
lr-set-expr-s->lr1-s-set-expr, 133
-lr-pop-tstk, 136
lr-set-pos, 24, 39–42, 44, 62, 90, 93–95, 122–124, 136–142, 144, 145, 148, 152, 154, 156–160, 163, 164, 167, 168, 170, 172, 173, 175, 176, 184, 185, 187, 188, 190, 192, 193, 205–208, 210, 211, 214, 215, 334, 335, 337, 340, 341, 343, 345, 346, 353–355, 357–363, 367–370, 373, 374, 376–378
lr-set-temp, 25, 41, 43, 63, 108, 139, 140, 144, 148, 363, 367, 368
lr-set-tstk, 24, 38, 353, 369
lr-t-addr, 6, 26, 29, 76–79, 82–84, 98, 99, 102, 161, 219, 265, 270, 271, 340
lr-temps, 23, 95, 101, 108, 131, 133, 135, 136, 140, 144–146, 148–150, 157, 165, 172
lr-temps-lr-apply-subr, 157
lr-temps-lr-do-temp-fetch, 149
lr-temps-lr-eval, 95
lr-temps-lr-funcall, 165
lr-temps-lr-pop-tstk, 108
lr-temps-lr-push-tstk, 131
lr-temps-lr-set-error, 133
lr-temps-lr-set-expr, 108
lr-temps-lr-set-expr-lr-pop-cst
k, 172
lr-temps-lr-set-pos, 136
lr-temps-lr-set-temp, 140
lr-temps-p-frame-put-assoc, 145
lr-total-heap-reqs, 319, 332, 379
lr-true-tag, 5, 9, 10, 17, 30, 77, 84, 98, 163, 181, 219, 245, 258, 261, 270, 294, 313
lr-type-contents-p, 26
lr-unbox-nat-offset, 7, 10, 97, 98, 219, 271
lr-undef-addr, 6, 15, 17, 19, 20, 25, 31, 35, 97, 99, 100, 102, 142, 143, 149, 150, 258, 260, 261, 264–266, 269, 312, 313, 316, 364, 367
lr-undefined-tag, 5, 15, 97, 265, 316
lr-unpack-offset, 6, 10
lr-valp, 98–100, 107, 114, 124, 125, 130, 134, 135, 143, 150, 151, 153, 160–164, 166, 167, 218, 219, 260, 261, 289, 320, 325–327, 329, 333, 379–381, 384
lr-valp-0-lr-0-addr-opener, 218
lr-valp-addr-0, 135
lr-valp-apply-subr-lr-apply-sub
r, 163
lr-valp-car-p-temp-stk-p-run-su
br, 163
br-cons-helper, 162
lr-valp-cdr-assoc-firstn-cdr-as
soc, 135
lr-valp-cons, 162
lr-valp-deposit-a-list-cons, 162
lr-valp-deposit-a-list-cons-con
s, 260
lr-valp-deposit-a-list-cons-num
berp, 260
lr-valp-deposit-a-list-cons-truep, 261
lr-valp-deposit-fetch-free-pointe
r, 114
r-offset, 113
r-offset-helper-1, 112
lr-valp-equal-value-fact, 161

lr-valp-f-lr-f-addr-opener, 219
lr-valp-fetch-tag-cons-lr-valp-
car-cdr, 160
lr-valp-fetch-tag-not-cons-lr-v
alp-car-cdr-0, 161
alp-listp, 161
lr-valp-fetch-tag-not-true-lr-v
alp-listp, 163
lr-valp-lr-compile-quote, 324
lr-valp-lr-compile-quote-flag-t, 325
lr-valp-lr-good-pointerp, 153
lr-valp-lr-s-eval-lr-s-similar-te
mps, 150
lr-valp-not-tag-cons-not-listp, 161
lr-valp-not-tag-true-not-listp, 163
lr-valp-t-lr-t-addr-opener, 219
lrps, 10

make-p-call-frame, 38, 74, 210
make-symbol, 12, 13, 147
make-temps-entries, 170–172, 209, 324, 328, 329
mark-instr, 8, 9
max-count-codelist, 13
max-ctrl-reqs, 379, 383
max-r, 178–182, 190
max-temp-reqs, 379, 383
max-word-size-reqs, 379, 383
member-area-name-offset-same, 117
member-assoc-area-name-cdr-lr-p
rograms-properp, 347
member-cdr-assoc-strip-cdrs-defi
nedp, 151
member-definedp-car, 306
member-disjointp-cons-arg2, 145
member-disjointp-non-member-1, 141
member-f-definedp-0, 307
member-lr-all-nodes, 228
member-lr-all-nodes-helper, 227
member-lr-free-list-nodes-type-
addr, 102
member-lr-good-pointerps-type-a
ddr-untag-whole, 366
member-make-symbol-max-count-co
de-list, 13
member-no-duplicatesp-assoc-equ
al, 307
member-strip-cdrs-lr-good-pointe
rp-tablep, 338
my-get-put, 2

name, 8, 9, 33, 59, 132, 168, 306, 309, 318–320, 324, 328, 329, 331, 380, 381
name-car-lr-compile-programs-pr
ogs, 132
name-car-p-prog-segment-s->lr, 381
name-car-s-progs-logic->s, 331
name-formal-vars-temp-var-dcls-p
rogram-body-cons, 59
no-duplicatesp, 117, 143, 147, 149, 150, 307, 308, 310, 318, 329, 332
no-duplicatesp-lr-free-list-node
s, 117
no-duplicatesp-occurences-1, 117
no-duplicatesp-remove-duplicate
s, 310
no-duplicatesp-strip-cars-s-con
struct-programs, 332
no-duplicatesp-strip-cars-s-pro
gs-logic->s, 332
no-duplicatesp-strip-cdrs-lr-ma
ke-temp-name-alist, 147
ke-temp-name-alist-1, 147
not-adpp-untag-add-addr-adpp-unt
ag, 120
not-adpp-untag-node-not-definedp
-lr-heap-name, 116
not-definedp-not-listp, 291
not-definedp-user-fname-p-runti
me-support-programs, 210
not-disjointp-member-arg1-cons-
arg2, 144
not-equal-0-count-list, 16
not-equal-lr-s-eval-temp-setp-n
ot-lr-s-similar-temps, 149
not-equal-make-symbol-car-gensy

m, 147
not-equal-x-add1-add1-x, 114
not-equal-x-add1-x, 114
not-iff-lr-s-temp-setp-not-lr-s
-similar-statesp, 150
-similar-statesp-helper, 150
not-labelledp-instrs, 285
not-labelledp-instrs-append, 285
not-labelledp-instrs-comp-body-
1, 285
not-lessp-difference-lr-boundar
y-offsetp-fact, 265
not-lessp-help-fact, 207
not-lessp-length-p-temp-stk-lr-
apply-subr, 89
not-lessp-length-proper-p-statep
-lr-eval-lr-set-pos, 353
not-lessp-lr-count-free-nodes-l
r-data-seg-table-heap-r, 323
r-eval-list-lr-set-pos, 205
not-lessp-lr-p-c-size-flag-t-1, 71
not-lessp-max-r-cadddr, 182
not-lessp-max-r-caddr, 182
not-lessp-max-r-cadr, 182
not-lessp-max-r-car, 182
not-lessp-p-ctrl-stk-size-make-p
-call-frame, 210
not-lessp-p-max-ctrl-stk-size-l
r-funcall, 92
not-lessp-p-max-temp-stk-size-l
r-funcall, 93
r-push-tstk, 60
not-lessp-plus-arity-length-for
mals, 208
mals-alt, 211
not-lessp-x-x, 71
not-listp-p-prog-segment-lr-exp
r, 84
not-listp-p-temp-stk-not-lr-che
ck-result1, 159
not-listp-prog-segment-not-lr-p
rograms-properp, 347
not-listp-s-progs-not-s-good-st
atep, 217
not-lr-check-resourcesp-temp-te
st-bad-max-temp-stk-size, 215
not-lr-valp-lr-undef-addr, 143
not-member-car-gensym-lr-make-te
mp-name-alist-1-cdr, 147
not-member-lr-all-nodes-too-sma
ll-addr, 228
not-member-make-symbol-lr-make-te
mp-name-alist-1-incr, 147
not-member-no-duplicates-cdr-as
soc, 149
soc-helper, 149
not-member-occurences-0, 117
not-p-max-node-fetch-fp-addr-not
-errorp-p-run-cons, 187
not-proper-p-statep-not-listp-p
-ctrl-stk, 75
not-psw-run-lr-eval, 128
not-same-signature-deposit-a-li
st-too-large-addr, 120
not-same-signature-deposit-too-
large-addr, 120
number-cons, 40, 282
number-cons-lr-expr-t-list, 40
numberp-arity, 344
numberp-car-cadr-caddr-cadddr-s
-apply-subr-r, 181
numberp-cdr-lr-p-pc, 45
numberp-cdr-untag-return-pc, 54
numberp-lessp-2-not-1-must-be-0, 265
numberp-lessp-4-not-3-not-2-not
-1-must-be-0, 265
numberp-max-r, 181
numberp-offset-return-pc, 54
numberp-offset-sub-addr, 115
numberp-s-eval-temp-ctrl-ws-heap
-r, 181
-r-opened, 195
nx, 40, 41, 56, 58, 110, 131, 133, 155, 175, 179, 191, 295, 334, 349, 352

object-addrs, 324, 325
occurrences, 117

offset, 6–10, 15, 21, 40–42, 44, 48, 53, 54, 57, 58, 61–64, 72, 73, 75–98, 101, 104–107, 109, 110, 113–118, 120–127, 131, 137, 141, 148, 150, 153, 154, 156–160, 163, 168–172, 183, 184, 187, 198–204, 206, 218, 220, 221, 225–234, 241, 242, 249, 250, 253, 256, 264–269, 273, 310–313, 315, 334–343, 345–347, 349, 350, 352–378, 380, 381
offset-add-addr, 53
offset-lr-max-node, 104
offset-p-pc-lr-funcall, 92
offset-sub-addr, 54
offset-tag, 40
offset-tag-cons, 101

p, 36, 37, 46, 75–84, 187, 198–205, 347, 352, 353, 355, 358–363, 367–369, 371–375, 377, 378, 380, 381, 384
p-accessor-clock, 27
p-accessor-code, 26, 27
p-accessors-lr->p, 44
p-accessors-lr-apply-subr, 46
p-accessors-lr-do-temp-fetch, 43
p-accessors-lr-funcall, 42
p-accessors-lr-if-ok, 43
p-accessors-lr-pop-cstk, 44
p-accessors-lr-pop-tstk, 39
p-accessors-lr-push-tstk, 43
p-accessors-lr-set-error, 39
p-accessors-lr-set-expr, 38
p-accessors-lr-set-pos, 39
p-accessors-lr-set-temp, 43
p-accessors-lr-set-tstk, 38
p-accessors-p-halt, 45
p-accessors-p-run-subr, 46
p-accessors-p-set-pc, 46
p-accessors-s->lr1, 101
p-call-okp, 37
p-car-clock, 27, 36, 75, 79, 80, 198, 333, 371
p-car-code, 27, 30, 51, 292
p-cdr-clock, 27, 36, 76, 80, 199, 333, 371
p-cdr-code, 27, 30, 51, 293
p-clock1, 334, 335, 352, 358, 359, 361–363, 368, 374, 377, 378, 381
p-cons-clock, 28, 36, 77, 80, 81, 187, 200, 333, 371
p-cons-code, 27, 30, 51, 293
p-ctrl-stk, 9, 10, 22–25, 36–39, 41, 43–46, 50, 61–63, 75–85, 87, 88, 90–92, 94, 95, 100, 101, 122, 123, 130, 134, 135, 141, 142, 144–146, 148, 150, 151, 165, 182–184, 191, 192, 198–204, 207, 210, 212, 324, 328, 336–338, 340, 343, 345–347, 353, 355, 358, 360, 361, 366–368, 370, 376, 378, 380, 382
p-ctrl-stk-size, 87, 92, 182, 184, 192, 198–204, 207, 209, 210, 212, 324, 328
p-ctrl-stk-size-0, 209
p-ctrl-stk-size-p-ctrl-stk-lr-funcall, 212
p-ctrl-stk-size-p-ctrl-stk-s->lr, 328
p-current-instruction, 53
p-current-instruction-opener, 52
p-current-program, 7, 9, 21, 23, 40, 52, 57–59, 61–64, 72, 73, 85–95, 108, 114, 121–125, 130–133, 137, 141, 142, 148, 150, 153–160, 168–172, 175, 176, 183, 184, 192, 206, 336–339, 341–343, 345–347, 349, 350, 352–370, 373–378, 380
p-current-program-lr-apply-subr, 87
p-current-program-lr-do-temp-fetch, 108
p-current-program-lr-eval, 87

p-current-program-lr-pop-tstk, 108
p-current-program-lr-push-tstk, 108
p-current-program-lr-set-error, 133
p-current-program-lr-set-expr, 40
p-current-program-lr-set-pos, 40
p-current-program-lr-set-temp, 108
p-current-program-p-state, 52
p-data-segment, 9–11, 22–25, 27, 36–39, 42–46, 48–50, 63, 75–84, 86, 87, 90, 94, 95, 101, 109, 114, 121–125, 130, 131, 135, 136, 144, 146, 148, 150–152, 154–165, 171–176, 182–185, 187–191, 193–195, 198–207, 211, 214, 216, 235, 281, 282, 284, 292–294, 308–318, 320, 329, 333, 336–338, 340–343, 345–347, 355, 358, 360, 361, 366, 368, 370, 378–382, 384
p-false-clock, 28, 36, 79, 81, 201, 333, 372
p-false-code, 28, 30, 51, 293
p-falsep-clock, 29, 36, 78, 82, 202, 333, 372
p-falsep-code, 28, 30, 51, 293
p-final-pc, 351–356, 358–363, 367–369, 374, 377, 378
p-frame, 3, 19, 142, 144, 145, 337
p-good-resultp, 50–52, 75–79
p-good-resultp-p-halt-errorp-opener, 52
p-good-resultp-p-state-opener, 51
p-good-resultp-run-car, 75
p-good-resultp-run-cdr, 75
p-good-resultp-run-cons, 77
p-good-resultp-run-false, 78
p-good-resultp-run-falsep, 78
p-good-resultp-run-listp, 76
p-good-resultp-run-nlistp, 76
p-good-resultp-run-true, 79
p-good-resultp-run-truep, 77
p-halt, 37, 38, 45, 46, 52
p-last-2-instrs-main-program, 379
p-listp-clock, 29, 37, 76, 82, 203, 333, 372
p-listp-code, 29, 30, 51, 293
p-max-ctrl-stk-size, 10, 22–25, 36–39, 43–46, 87, 88, 92, 95, 101, 131, 182, 198–204, 207, 210, 370
p-max-ctrl-stk-size-lr-eval, 46
p-max-temp-stk-size, 10, 22–25, 36–39, 41, 43–46, 60, 63, 64, 88, 89, 93, 95, 101, 111, 131, 139, 182, 198, 200–204, 207, 215, 353, 366–370
p-max-temp-stk-size-lr-eval, 46
p-nlistp-clock, 29, 37, 76, 83, 203, 333, 372, 373
p-nlistp-code, 29, 30, 51, 293
p-objectp, 48, 60–63, 78, 220, 222, 223, 234–237, 239, 240, 262, 310, 311
p-objectp-bad-type, 78
p-objectp-car-lr-compile-quote, 222
p-objectp-cdr-assoc-bindings-proper-p-alistp, 63
p-objectp-cdr-assoc-car-lr-compile-quote, 262
p-objectp-cdr-assoc-litatom-proper-p-alistp, 61
p-objectp-lookup-deposit, 236
p-objectp-lookup-deposit-a-list, 235
p-objectp-lookup-lr-init-data-seg-table, 239
p-objectp-opener-alt-lr-proper-free-listp, 235
p-objectp-similar-p-states, 48
p-objectp-type, 352
p-pc, 7, 9, 10, 21, 23–25, 35, 37–47, 50, 53, 54, 57–59, 61–64, 72, 73, 84–95, 101, 108, 114, 121–125, 131, 133, 134, 137, 141, 148, 150, 153, 154, 156–160, 163, 168–172, 183, 184, 206, 308, 334–343, 345–347, 349, 350, 352–378, 380,

381
p-pc-run-car, 371
p-pc-run-cdr, 371
p-pc-run-cons, 371
p-pc-run-false, 372
p-pc-run-falsep, 372
p-pc-run-listp, 372
p-pc-run-nlistp, 372
p-pc-run-true, 373
p-pc-run-truep, 373
p-pc-s->lr, 381
p-plus, 347
p-prog-segment, 10, 23–25, 35–39, 42–46, 48–51, 56, 57, 59, 61, 63, 72, 73, 75–95, 101, 108, 114, 121, 131, 133, 134, 154, 157, 163, 165, 169, 170, 172, 187, 198, 200–204, 212, 283, 308–310, 338, 342–345, 347, 353–362, 364–366, 369–373, 375, 377, 378, 380–382
p-prog-segment-lr-eval, 46
p-psw, 9, 10, 23–26, 36–46, 50, 60, 62–64, 73, 75–95, 101, 111, 114, 121–125, 128, 130, 131, 133, 135, 137, 138, 141, 142, 144, 145, 148, 151–160, 163–165, 167–176, 183–185, 187–195, 198–208, 210–212, 214–216, 320, 324, 334–338, 340, 342, 343, 345, 346, 352, 353, 355, 357–361, 363, 367–379, 381
p-psw-lr-eval-flag-list-flag-t, 94
p-psw-lr-pop-tstk-lr-eval-flag-t, 175
p-psw-not-run, 46
p-psw-p-halt-x-y-error-msg, 46
p-psw-run-lr-apply-subr-lr-chec k-resourcesp, 207
p-psw-run-lr-if-ok-p-psw-run, 176
p-psw-run-p-psw-lr-if-ok-not-ru n-check-resourcesp, 215
p-psw-run-p-run-subr-lr-check-re sourcesp, 206
p-psw-run-run-car-lr-check-reso urcesp, 197
p-psw-run-run-cdr-lr-check-reso urcesp, 198
p-psw-run-run-cons-lr-check-res ourcesp, 200
p-psw-run-run-false-lr-check-re sourcesp, 200
p-psw-run-run-falsep-lr-check-re sourcesp, 201
p-psw-run-run-listp-lr-check-re sourcesp, 202
p-psw-run-run-nlistp-lr-check-re sourcesp, 203
p-psw-run-run-true-lr-check-res ourcesp, 203
p-psw-run-run-truep-lr-check-re sourcesp, 204
p-recognizer-clock, 26, 29, 30
p-recognizer-code, 26, 29, 30
p-run-subr, 36, 37, 46, 85, 86, 89, 90, 114, 121, 154, 159, 163, 168, 188, 207, 338, 340, 342, 343, 371, 374
p-run-subr-clock, 333, 335, 371, 374
p-run-subr-p-pc-add-addr-lr-p-p c-lr-p-c-size, 373
p-run-subr-preserves-lr-good-poi
  nterp-tablep, 338
  nterps, 341
p-run-subr-preserves-lr-proper-
  ctrl-stkp, 339
  free-listp, 121
  heapp, 342
  heapp2, 154
  heapp2-alt, 342
p-run-subr-preserves-lr-valp, 114
p-runtime-support-programs, 30, 33, 34, 84, 85, 163, 187, 206, 210, 216, 217, 290, 291, 309
p-set-pc, 36, 37, 46, 75–86, 114, 121, 154, 159, 163, 168, 187, 198–205, 207, 333, 334, 338, 340, 342, 343, 347, 352, 353, 355,

358–363, 367–375, 377, 378, 380
p-set-pc-lr->p-equal-p-fact, 370
p-set-pc-lr->p-lr-set-expr, 347
p-set-pc-twice, 374
p-state, 10, 22–26, 36–38, 51–53, 196, 197, 222, 223, 235–240, 262, 263, 274, 285–292, 306–308, 370
p-statep, 11
p-temp-stk, 9–11, 22–25, 27, 36–39, 41, 43–46, 50, 60, 61, 63, 64, 73, 75–85, 89, 92, 93, 95, 101, 111, 130, 131, 136–140, 148, 151, 152, 154–156, 159–165, 167–171, 174–176, 182, 184, 185, 190, 192, 198–205, 207, 212, 215, 320, 329, 333, 334, 336, 338, 342, 343, 345–347, 353, 355, 357, 358, 360, 361, 363, 366–370, 378–382
p-temp-stk-lr-do-temp-fetch-p-p
sw-run, 151
p-temp-stk-lr-pop-tstk, 39
p-temp-stk-p-ctrl-stk-p-data-se
gment-run-car, 79
gment-run-cdr, 80
gment-run-cons, 80
gment-run-false, 81
gment-run-falsep, 81
gment-run-listp, 82
gment-run-nlistp, 82
gment-run-true, 83
gment-run-truep, 83
p-test-bool-and-jump-okp, 196, 197
p-test-bool-and-jump-okp-f-cons
-bool-f, 197
-bool-t, 197
p-test-bool-and-jump-okp-t-cons
-bool-f, 197
-bool-t, 196
p-true-clock, 29, 37, 79, 83, 204, 333, 373
p-true-code, 29, 30, 51, 294
p-truep-clock, 30, 37, 77, 84, 205, 334, 373
p-truep-code, 30, 51, 294
p-word-size, 10, 22–26, 36–39, 43–50, 63, 86, 88, 95, 101, 131, 182, 198–205, 207, 216, 292–294, 309, 310, 312–318, 370
p-word-size-lr-eval, 47
pair-formal-vars-with-actuals, 74, 165
pair-formals-with-addresses, 19, 220, 259, 273, 281, 326, 328
pair-formals-with-addresses-lr-
data-seg-table-list, 327
pair-temps-with-initial-values, 74, 75, 165, 171, 344
pairlist-plist-1, 170
plist, 13, 74, 134, 144, 146, 148, 170, 217, 229, 259, 281, 348
plist-delete, 229
plist-listp-x-append-x-not-0, 146
plist-lr-convert-num-to-ascii, 283
plist-strip-cdrs, 281
plistp, 34, 64, 66, 73, 130, 146, 155, 216, 233, 259, 281–283, 285, 296, 309
plistp-comp-body-1, 66
plistp-comp-if, 66
plistp-comp-programs-1, 281
plistp-comp-temp-test, 66
plistp-first-n, 73
plistp-label-instrs, 285
plistp-lastcdr-nil, 130
plistp-lr-all-nodes, 233
plistp-lr-compile-body, 155
plistp-lr-compile-body-1, 216
plistp-lr-expr-s->lr1, 155
plistp-pair-formals-with-addres
ses, 259
plistp-pairlist, 73
plistp-strip-cars, 309
plus-constant-fact-helper-1, 68
plus-times-fact-1, 105
pop, 24, 25, 39, 44, 99, 336

pop-p-ctrl-stk-lr-funcall, 92
popn, 38, 74, 93, 173, 345
popn-nlistp, 93
popn-restn, 173
pps, 9
program-body, 7–9, 21, 33, 35, 53, 56–59, 61–64, 72–95, 114, 121–125, 128, 130–132, 137, 141, 148, 150, 153–160, 163, 168–172, 175, 176, 183, 184, 187, 192, 198, 200–204, 206, 283, 286, 288, 292, 307–309, 318, 336–339, 341–343, 345–347, 349, 350, 352–378, 380
program-body-assoc-cdr-lr-compi le-programs, 318
program-body-assoc-comp-program
  s, 59
  s-1, 59
program-body-assoc-lr-compile-p rograms, 128
program-body-p-current-program-s->lr1, 132
proper-labeled-p-instructionsp, 49, 64, 282, 285–292, 306, 308
proper-labeled-p-instructionsp-
  append, 282
  comp-body-1, 292
  comp-body-1-helper-1, 285
  comp-body-1-helper-2, 286
  comp-body-1-helper-3, 287
  comp-body-1-helper-4, 287
  comp-body-1-helper-5, 289
  comp-body-1-helper-6-1, 290
  comp-body-1-helper-6-2, 290
  comp-body-1-helper-7, 291
  find-labelp-non-litatom, 64
  label-ret, 306
  lr->p-similar-states, 49
  nil, 285
proper-p-alistp, 50, 60–63, 92, 220, 274
proper-p-alistp-all-litatoms-al l-p-objectps-lookup, 220
proper-p-alistp-lr->p-similar -states, 50
proper-p-alistp-lr-funcall, 92
proper-p-alistp-lr-make-initial -temps, 274
proper-p-alistp-p-objectp, 60
proper-p-alistp-pair-formal-wit h-addresses, 273
proper-p-alistp-put-assoc, 62
proper-p-ctrl-stkp, 50, 88, 92
proper-p-ctrl-stkp-lr->p-simi lar-states, 50
proper-p-ctrl-stkp-lr-apply-sub r, 88
proper-p-ctrl-stkp-lr-funcall, 92
proper-p-data-segmentp, 49, 52, 90, 109, 310–318
proper-p-data-segmentp-bad-type, 52
proper-p-data-segmentp-deposit, 310
proper-p-data-segmentp-deposit-
  a-list-cons, 311
  a-list-cons-cons, 311
  a-list-cons-numberp, 312
  a-list-cons-truep, 312
  helper, 310
proper-p-data-segmentp-fetch, 310
proper-p-data-segmentp-implies-lr-proper-p-areasp, 109
proper-p-data-segmentp-lr->p-similar-states, 49
proper-p-data-segmentp-lr-apply -subr, 90
proper-p-data-segmentp-lr-data-seg-table, 318
proper-p-data-segmentp-lr-init-
  data-seg, 316
  data-seg-compile-t0, 317
  data-seg-helper, 316
proper-p-framep, 63, 86, 87, 92
proper-p-framep-lr->p-similar -states, 86
proper-p-framep-lr-apply-subr, 87
proper-p-framep-top-p-ctrl-stk-lr-funcall, 91

proper-p-instructionp, 8, 49, 64, 281–283, 308
proper-p-instructionp-add-addr, 281
proper-p-instructionp-deposit, 281
proper-p-instructionp-eq, 281
proper-p-instructionp-fetch, 281
proper-p-instructionp-jump-labe
l, 283
proper-p-instructionp-pop-globa
l-free-ptr, 281
proper-p-instructionp-push-const
ant-opener, 64
proper-p-instructionp-push-glob
al-free-ptr, 282
proper-p-instructionp-push-loca
l-temp-car, 282
l-temp-cdr, 282
l-temp-cons, 282
proper-p-instructionp-ret, 281
proper-p-instructionp-set-globa
l, 308
proper-p-instructionp-set-local
-temp-cons, 282
proper-p-instructionp-similar-p
-states, 49
proper-p-instructionp-test-bool
-and-jump-label, 283
proper-p-prog-segmentp, 49, 64, 73, 74, 89, 92, 308–310
proper-p-prog-segmentp-append, 64
proper-p-prog-segmentp-comp-pro
grams, 310
grams-1, 308
grams-1-helper, 307
proper-p-prog-segmentp-length-p
rogram-body, 74
proper-p-prog-segmentp-lr->p-
similar-states, 49
proper-p-prog-segmentp-lr-apply
-subr, 88
proper-p-prog-segmentp-p-runtime
-support-programs, 309
proper-p-programp, 8, 292–294, 309
proper-p-programp-append-car-pr
og-segment, 308
proper-p-programp-p-car-code, 292
proper-p-programp-p-cdr-code, 292
proper-p-programp-p-cons-code, 293
proper-p-programp-p-false-code, 293
proper-p-programp-p-falsep-code, 293
proper-p-programp-p-listp-code, 293
proper-p-programp-p-nlistp-code, 293
proper-p-programp-p-true-code, 293
proper-p-programp-p-truep-code, 294
proper-p-push-constant-instructi
onp, 64
proper-p-state-p-p-run-subr-ope
ner-1, 89
ner-2, 89
ner-3, 90
proper-p-statep, 61–64, 73, 75–83, 85–90, 93–95, 109, 114, 121–125, 130, 131, 133–135, 137, 141, 144–146, 148, 150, 151, 154–161, 163, 164, 168, 170–176, 182–185, 187, 188, 190–195, 198, 200–207, 210–212, 214–216, 319, 323, 336–338, 340–343, 345, 346, 352, 353, 355, 357, 358, 360, 361, 363, 366–368, 371–374, 376–378, 380, 383
proper-p-statep-bad-type-1, 75
proper-p-statep-bad-type-2, 78
proper-p-statep-lessp-length-p-te
mp-stk-max-temp-stk-size, 353
proper-p-statep-lr->p-equal-w
ord-size-0, 88
proper-p-statep-lr->p-implies
-lr-proper-p-areasp, 109
proper-p-statep-lr->p-lessp-ct
rl-stk-size, 87
proper-p-statep-lr->p-lessp-m
ax-ctrl-stk-size, 88
ax-temp-stk-size, 88
proper-p-statep-lr->p-lr-eval
-list, 158
-list-helper, 158

proper-p-statep-lr->p-lr-pop-t
stk, 62
proper-p-statep-lr->p-lr-push
-tstk, 61
proper-p-statep-lr->p-lr-set-e
xpr, 95
proper-p-statep-lr->p-lr-set-p
os, 62
proper-p-statep-lr->p-member-
formals-definedp-bindings, 135
proper-p-statep-lr->p-not-0-p
-temp-stk, 130
proper-p-statep-lr->p-numberp
-max-ctrl-stk-size, 87
-max-temp-stk-size, 88
-word-size, 88
proper-p-statep-lr->p-plistp-p
-temp-stk, 130
proper-p-statep-lr->p-s->lr, 318
-logic->s, 383
1-strip-cars-bindings-ctrl-stk, 145
proper-p-statep-lr->p-strip-c
ars-bindings-ctrl-stk, 134
proper-p-statep-lr-apply-subr, 90
proper-p-statep-lr-apply-subr-st
ate, 86
proper-p-statep-lr-do-temp-fetc
h, 64
proper-p-statep-lr-funcall, 93
proper-p-statep-lr-if-ok, 95
proper-p-statep-lr-push-tstk-qu
ote, 73
proper-p-statep-lr-set-error, 133
proper-p-statep-lr-set-expr-lr-p
op-cstk, 94
proper-p-statep-lr-set-temp, 63
proper-p-statep-p-run-subr, 85
proper-p-statep-p-set-pc, 352
proper-p-statep-p-set-pc-equal-p
-set-pc, 352
proper-p-temp-stkp, 49, 50, 60, 61,
63, 73, 74, 89, 92, 93, 130
proper-p-temp-stkp-all-p-objectp
s, 74
proper-p-temp-stkp-lr->p-lr-p
ush-tstk, 60
proper-p-temp-stkp-lr->p-simi
lar-states, 49
proper-p-temp-stkp-lr-apply-sub
r, 89
proper-p-temp-stkp-lr-funcall, 93
proper-p-temp-stkp-lr-push-tstk
-assoc-bindings, 61
proper-p-temp-stkp-p-temp-stk-l
r-do-temp-fetch, 63
r-push-tstk-quote, 73
proper-p-temp-stkp-plistp-p-temp
-stk, 130
proper-p-temp-stkp-popn, 74
proper-p-temp-var-dclsp, 49, 284, 307
proper-p-temp-var-dclsp-all-lit
atoms-all-undef-addrs, 284
proper-p-temp-var-dclsp-lr->p
-similar-states, 49
properp-p-temp-var-dclps-member
-lr-programs-properp, 307
push, 24, 38, 151
put, 2, 109, 218, 310
put-assoc, 47, 62, 109, 119, 140, 142–
146, 337
put-assoc-opener-1, 142
put-assoc-opener-2, 142
put-assoc-put-assoc-1, 109
put-assoc-put-assoc-2, 109
put-assoc-restn, 144
put-not-listp, 109
put-put, 109
put-value, 8, 10
put-zero, 109

remainder-difference-not-equal-
lessp-fact, 227
remove-duplicates, 310, 331
restn, 7, 13, 23, 67–71, 99, 108, 112,
144, 149, 159, 160, 167, 173,
282, 295, 304
restn-add1-opener-alt, 159
restn-comp-body-1-list-fact, 69

ret-pc, 74, 88, 91, 92, 144, 376
ret-pc-make-p-call-frame, 74
reverse, 73, 100, 160, 163, 164, 166, 167, 170, 344
reverse-butlast, 166
reverse-reverse-alt, 170

s->lr, 22, 319, 323, 328, 329, 333, 378, 380–383
s->lr-ok, 329
s->lr1, 22, 23, 101, 110–112, 132–140, 144–146, 148, 150–153, 155, 156, 163–165, 170–176, 182–185, 187–195, 205–208, 210–212, 214–216, 274, 306
s->lr1-lr-funcall-s-fun-call-state, 165
s->lr1-s-set-pos-lr-set-pos, 136
s-add-temp-r, 178–180
s-all-progs-temps-setp, 4, 183, 192, 216, 324, 329, 330
s-all-temps-setp, 4, 183, 192, 216, 324, 329
s-ans, 148, 155, 156, 164, 165, 172, 174, 176, 179, 185, 190, 193, 197–205, 320, 329
s-apply-car-r, 177
s-apply-cdr-r, 177, 178
s-apply-cons-r, 177, 178
s-apply-false-r, 177, 178
s-apply-falsep-r, 177, 178
s-apply-listp-r, 177, 178
s-apply-nlistp-r, 177, 178
s-apply-subr-r, 177, 180, 181, 188, 189, 198–206, 214
s-apply-true-r, 177, 178
s-apply-truep-r, 177, 178
s-body, 4, 18, 20, 22, 128, 132, 134–140, 144–146, 148, 150–152, 155, 156, 163–165, 170, 171, 173–176, 182–185, 187, 188, 190–195, 205–212, 214–217, 223, 244, 246, 248, 277, 295, 296, 302–305, 318, 324, 329, 330
s-body-car-s-progs-logic->s, 330
s-change-temp, 144, 148
s-check-temps-setp, 4, 183, 192, 216, 324, 329
s-collect-all-temps, 183, 192
s-construct-programs, 331, 332
s-data-seg-body-restrictedp, 248, 277, 278, 294, 314, 326, 331
s-data-seg-list-restrictedp, 248, 278, 294, 314, 326, 327, 331
s-err-flag, 4, 22, 101, 110, 111, 133–136, 147, 148, 155, 156, 165, 171, 174–176, 178–180, 182, 183, 190–193, 195, 205–207, 209–216, 324, 329, 382
s-eval, 4, 101, 110–112, 133, 136, 147, 148, 155, 156, 164, 165, 171–174, 176, 179, 180, 183–185, 188–195, 205–207, 209–212, 214–216, 320, 324, 329, 382
s-eval-ctrl-heap-temp-ws-s-fun-call-state-opener, 212
s-eval-ctrl-r, 180–182, 209, 210, 213, 319, 324, 329
s-eval-ctrl-r-funcall-opener, 209
s-eval-err-flag-not-run-fact, 101
s-eval-flag-run-car-s-apply-sub r-r-not-zero, 214
s-eval-flag-run-flag-t-s-check-te mps-setp, 183
s-eval-flag-run-flag-t-subsetp-s-collect-all-temps, 183
s-eval-flag-run-s-eval-temp-r-n ot-zero, 215
s-eval-flag-run-v&c$-not-f-fl ag-t, 382
s-eval-heap-r, 180–182, 189, 191, 206, 207, 213, 319, 324, 329, 383
s-eval-r, 178–180, 195
s-eval-subsetp-s-collect-temp-a list-s-set-pos-if, 192

s-eval-temp-r, 180–182, 213, 215, 319, 324, 329
s-eval-ws-r, 180–182, 213, 319, 324, 329
s-expand-temps, 4
s-expr, 4, 110–112, 134–140, 144–146, 148, 150–153, 155, 156, 163–165, 170–174, 179, 180, 183, 187–195, 205–216, 332
s-expr-list, 4, 110, 132, 155, 170, 175, 178, 182, 191, 216
s-expr-logic->s, 332
s-formals, 20, 128, 132, 133, 135, 145, 146, 150, 171, 208, 209, 213, 305, 318, 328, 329, 332, 379
s-formals-car-s-progs-logic->s, 332
s-formals-s-prog-logic->s, 379
s-fun-call-state, 112, 165, 174, 180, 209, 213, 214
s-good-state-logic->s, 329
s-good-statep, 4, 132, 138–140, 144, 146, 148, 150–152, 164, 170, 171, 174, 176, 183–185, 188–195, 205–212, 214–217, 281, 306, 318–320, 324, 329, 330
s-good-statep-formals-assoc-cdr-s-progs, 209
s-good-statep-length-cdr-s-expr-funcall, 170
s-good-statep-length-s-temp-list, 281
s-good-statep-program-body-car-lr-compile-programs, 132
s-heap-reqs, 242–245, 250–253, 255, 259, 261–263, 267, 275, 311, 313, 325, 326
s-heap-reqs-body, 243, 244, 275–278, 294, 314, 320, 326
s-heap-reqs-flag-list-cons-opener, 252
s-heap-reqs-flag-list-nil-opener, 252
s-heap-reqs-list, 244, 245, 277–279, 294, 314, 321, 322, 327
s-heap-reqs-object-0, 267
s-heap-reqs-object-t, 250
s-init-data-seg-restrictedp, 248, 263, 273, 278, 279, 295, 314, 325, 326, 328, 330
s-init-heap-reqs, 244, 245, 263, 278–280, 295, 315, 322, 323, 325, 326, 328
s-init-ws-reqs, 246, 247, 263, 315
s-l-eval-equiv-hyps, 4
s-l-eval-flag-run-hyps, 4
s-max-subr-reqs, 180, 198–205, 207, 216, 247, 292, 293, 309, 310, 312, 315–318, 324
s-params, 4, 22, 135, 136, 151, 152, 172, 173, 176, 180, 183–185, 188–191, 193, 194, 205–207, 214, 216, 306, 318–320, 323, 324, 328–330, 382
s-params-logic->s, 330
s-pname, 22, 101, 133, 135, 144–146, 150, 156, 187, 320, 324, 328, 329, 331, 381
s-pname-logic->s, 331
s-pos, 4, 22, 101, 110–112, 132–140, 144–146, 148, 150–153, 155, 156, 163–165, 170, 171, 173–176, 178–180, 182–184, 187–195, 205–216, 320, 324, 329, 331, 381
s-pos-logic->s, 331
s-prog, 4, 22, 132–140, 144–146, 148, 150–152, 155, 156, 163–165, 170, 171, 173–176, 182–185, 187, 188, 190–195, 205–212, 214–216, 281, 318, 379
s-programs-okp, 169, 208, 212, 305
s-programs-okp-formals-not-f, 208
s-programs-properp, 305
s-progs, 4, 22, 101, 132, 133, 135, 144–146, 150, 156, 165, 171, 172, 183, 187, 192, 208–210, 212, 213, 216, 217, 273, 274,

306, 318–320, 323, 324, 328–332, 380–383
s-proper-exprp, 217, 296, 298, 299, 303, 304
s-proper-exprp-plist-temp-list, 217
s-restrict-subrps, 217, 302–305, 331
s-restrict-subrps-list-lr-proper-get-t, 302
s-restrict-subrps-progs, 217, 305, 306, 318, 319, 329, 331
s-restrict-subrps-s-body-member-s-restrict-subrps-progs, 305
s-restrict-subrps-s-restrict-subrps-cur-expr, 303
s-restrict-subrps-t-lr-proper-get-t, 303
s-restricted-objectp, 247, 248, 253, 255, 259–262, 311–313, 324, 325, 330
s-restrictedp, 248, 280, 295, 302, 304–306, 318–320, 323, 324, 328, 329, 382
s-set-expr, 110, 111, 133, 136, 155, 179, 191, 193, 194
s-set-pos, 110–112, 136, 148, 156, 164, 165, 170–173, 179, 180, 184, 185, 188–190, 192–195, 205–207, 209–215
s-temp-eval, 17, 19–21, 32, 34, 41, 56, 62, 66, 68–70, 73, 84, 111, 129, 139, 142, 144–146, 148, 150, 152, 156, 163, 164, 179, 187, 190, 194, 195, 205, 214, 217, 243, 246, 248, 287, 290, 296, 298–300, 302, 303, 335, 343, 348, 349, 362, 363, 369, 370, 374, 377
s-temp-fetch, 17, 19–21, 32, 34, 42, 56, 63, 64, 66, 69, 70, 73, 84, 111, 129, 140, 146, 150–152, 156, 163, 164, 180, 187, 190, 194, 195, 205, 214, 217, 243, 246, 248, 286, 290, 295, 298, 300, 302, 303, 335, 337, 343, 348, 349, 365, 368–370, 374, 377
s-temp-list, 20, 128, 132, 133, 145, 146, 150, 171, 172, 209, 210, 213, 281, 305, 318, 328–330
s-temp-list-car-s-progs-logic->s, 330
s-temp-setp, 150, 179, 194
s-temp-test, 17, 19–21, 33, 34, 41, 56, 62–64, 66, 68–70, 73, 84, 111, 129, 139, 142, 144–146, 148, 150–152, 156, 163, 164, 179, 187, 190, 194, 195, 205, 214, 215, 217, 243, 246, 248, 286, 287, 290, 296, 298–300, 302, 303, 335, 337, 343, 348, 349, 364–370, 374, 377
s-temps, 4, 22, 135, 144, 147, 148, 150–152, 172, 173, 176, 179, 180, 183–185, 188–194, 205–207, 214, 216, 281, 318–320, 323, 324, 328–330
s-temps-logic->s, 330
s-total-heap-reqs, 244, 273, 280, 295, 302, 304–306, 318–320, 323, 324, 328, 329, 380–383
s-total-ws-reqs, 247, 273, 318, 319, 329
s-total-ws-reqs-not-lessp-s-max-subr-reqs, 318
s-ws-reqs, 245–247, 262, 313
s-ws-reqs-body, 245, 246, 314
s-ws-reqs-flag-not-list-t, 313
s-ws-reqs-list, 246, 247, 314
same-signature, 48–50, 63, 86, 87, 94, 95, 108, 116, 119, 120, 156, 221–224, 267, 311–318, 320
same-signature-car-lr-compile-quote, 223
uote-generalized, 221
uote-helper, 221
uote-reducer, 317
same-signature-car-lr-data-seg-t

able, 320
able-body, 223
able-list, 224
able-list-helper, 223
able-list-reducer, 317
same-signature-car-lr-init-data
-seg-table, 267
-seg-table-help-1, 267
-seg-table-reducer, 317
same-signature-commutative, 86
same-signature-cons, 119
same-signature-deposit, 116
same-signature-deposit-a-list, 120
same-signature-lr-apply-subr, 86
same-signature-nil, 119
same-signature-p-run-subr, 86
set-local-var-value, 25, 43
signature, 119
strip-cadrs, 35, 284, 292, 344
strip-cars-append, 74
strip-cars-bindings-top-p-ctrl-stk-lr-funcall, 91
strip-cars-equal-definedp-equal, 141
strip-cars-firstn, 108
strip-cars-lr-compile-programs, 132
strip-cars-lr-make-initial-temps, 259
strip-cars-lr-make-temp-name-alist, 148
st-1, 148
strip-cars-lr-make-temp-var-dcls, 135
strip-cars-lr-temps-strip-cars-temp-var-dcls, 146
strip-cars-nil-fact, 140
strip-cars-pair-formals-with-addresses, 259
strip-cars-pair-temps-with-initial-values, 74
strip-cars-pairlist, 74
strip-cars-restn, 108
strip-cdrs, 19, 34, 107, 135, 142, 143, 145–147, 149–152, 220, 263, 280, 281, 289, 297–300, 304, 305, 324, 327, 328, 336–338, 343, 344, 366, 382
strip-cdrs-append, 343
strip-cdrs-pair-temps-with-initial-values, 344
strip-cdrs-pairlist, 343
strip-logic-fnames, 35, 56, 57, 208, 289–292, 307, 308
strip-logic-fnames-cdr-lr-compile-programs, 208
strip-logic-fnames-lr-compile-programs, 208
sub-addr, 44, 54, 96, 98, 103, 105, 115, 116, 225–227, 232
sub-addr-area-name-offset-same, 116
sub1-plus-not-zerop-fact-1, 120
subr-arity-alist, 344
subseqp, 13
subseqp-append, 13
subsetp, 183, 185, 192, 304–306, 308
subsetp-cdr, 306
subsetp-not-member-both, 185

tag, 6, 14, 15, 17, 21–24, 26–29, 37–40, 75–77, 79–84, 96, 101, 106, 127, 128, 160–163, 186, 218, 225, 229–234, 256–258, 260, 261, 264–266, 268–273, 312, 313, 316, 320, 341, 352, 355, 358, 360, 361, 380, 381
tag-type-name-offset-equal-same, 230
temp-alist-to-set, 4, 183, 192, 216, 324, 329
temp-var-dcls, 9, 33, 35, 38, 56, 57, 59–61, 63, 91, 132–134, 142, 150, 165, 171, 212, 287–289, 292, 307, 309, 344
temp-var-dcls-assoc-comp-programs, 61
ms-1, 60
ms-lr-programs-properp, 91
temp-var-dcls-assoc-p-current-program-s->lr1, 133
temp-var-dcls-lr-compile-progra

ms, 132
times-quotient-lessp-fact-1, 196
top, 11, 27, 41, 63, 75, 76, 78–81, 92, 99–101, 111, 138–140, 146, 334, 336
top-stk, 11
top1, 77, 81
total-heap-reqs, 379, 383
type, 7, 26, 34, 45, 48, 52–55, 64, 75, 78, 96, 97, 102–106, 109, 112–114, 116–118, 121, 126–128, 152–154, 161, 162, 186, 219, 226–233, 240, 242, 249, 255–258, 260, 261, 264–268, 289, 302, 315, 339, 341, 353, 366
type-add-addr, 53
type-lr-p-pc, 45
type-lr-return-pc, 54
type-sub-addr, 54

unlabel, 8, 53, 71, 75–84, 163, 187, 198, 200–204, 354, 362, 364, 365, 371–373, 375
unlabel-car-last-comp-body, 375
unlabel-get-last-funcall-body-assoc-comp-programs, 375
unlabel-get-lr-p-pc-program-body-assoc-comp-programs, 347
unlabel-list-label, 71
untag, 7, 9–11, 26, 44, 45, 47, 48, 52–55, 61, 75, 78, 86, 90, 91, 96–99, 102–107, 109, 112–118, 120–122, 124–128, 130, 153–155, 162, 186, 196, 199, 200, 219, 220, 225, 227–233, 235, 240–242, 249, 250, 254–258, 260, 261, 264–268, 289, 309–311, 318, 339, 342, 366, 379, 381
untag-addr-addr-tag, 106
user-fname, 33, 37, 84, 85, 91, 92, 165, 169, 171, 172, 187, 208–210, 212, 213, 291, 331, 375
user-fnamep, 51

value, 8–10, 15, 19, 96, 120, 264

x-y-error-msg, 38, 46

zerop-lr-convert-digit-to-ascii, 283
zerop-lr-convert-num-to-ascii, 283