

#|

Copyright (C) 1994 by Computational Logic, Inc. All Rights Reserved.

This script is hereby placed in the public domain, and therefore unlimited editing and redistribution is permitted.

NO WARRANTY

Computational Logic, Inc. PROVIDES ABSOLUTELY NO WARRANTY. THE EVENT SCRIPT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SCRIPT IS WITH YOU. SHOULD THE SCRIPT PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT WILL Computational Logic, Inc. BE LIABLE TO YOU FOR ANY DAMAGES, ANY LOST PROFITS, LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

|#

```
; -----  
; 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(lst) 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
 $((k \in \mathbf{N}) \wedge (n \in \mathbf{N}))$
 $\rightarrow$ (get(*k*, put(*val*, *n*, *lst*)))
 $=$ **if** *k* = *n* **then** *val*
else get(*k*, *lst*) **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 F00 (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
;   '(F00 (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 '(F00 (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 than 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: $\text{LR-UNBOX-NAT-OFFSET} = 2$

DEFINITION: $\text{LR-NEGATIVE-GUTS-OFFSET} = 2$

DEFINITION:

$\text{lr-boundary-offsetp}(\text{offset}) = ((\text{offset} \bmod \text{LR-NODE-SIZE}) = 0)$

DEFINITION:

$\text{lr-boundary-nodep}(\text{node}) = \text{lr-boundary-offsetp}(\text{offset}(\text{node}))$

DEFINITION:

$\text{lr-nodep}(\text{addr}, \text{data-seg})$
= $((\text{type}(\text{addr}) = \text{'addr})$
 $\wedge (\text{caddr}(\text{addr}) = \text{nil})$
 $\wedge \text{listp}(\text{addr})$
 $\wedge \text{adpp}(\text{untag}(\text{addr}), \text{data-seg})$
 $\wedge \text{lr-boundary-nodep}(\text{addr})$
 $\wedge (\text{area-name}(\text{addr}) = \text{LR-HEAP-NAME}))$

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

DEFINITION:

$\text{lr-good-pointerp}(\text{addr}, \text{data-seg})$
= $(\text{lr-nodep}(\text{addr}, \text{data-seg})$
 $\wedge (\text{type}(\text{fetch}(\text{addr}, \text{LR-REF-COUNT-OFFSET}), \text{data-seg})$
 $= \text{'nat}))$

DEFINITION:

$\text{lr-expr}(p) = \text{cur-expr}(\text{offset}(\text{p-pc}(p)), \text{program-body}(\text{p-current-program}(p)))$

EVENT: Disable lr-expr.

DEFINITION:

$\text{lr-expr-list}(p)$
= $\text{restn}(\text{car}(\text{last}(\text{offset}(\text{p-pc}(p)))),$
 $\text{cur-expr}(\text{butlast}(\text{offset}(\text{p-pc}(p))),$
 $\text{program-body}(\text{p-current-program}(p))))$

EVENT: Disable lr-expr-list.

;;; Debugging Stuff.

DEFINITION:

```
mark-instr (instruction-list, n)
=  if n  $\simeq$  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))
          $\wedge$  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),
                  caddr (nodes)),
            lr-nodify (number + LR-NODE-SIZE, caddr (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* $\simeq$ 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 simpler.

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:

subseq (*list1*, *list2*)
= ((length (*list2*) $\nless$ length (*list1*))
 $\wedge$ (firstn (length (*list1*), *list2*) = *list1*))

EVENT: Disable subseq.

DEFINITION:

count-codelist (*initial*, *ascii-list*)
= **if** subseq (*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: subseq-append

subseq (plist (*x*), append (*x*, *anything*))

THEOREM: count-codelist-make-symbol

(*x* = make-symbol (*initial*, *num-list*))
→ (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*)
→ (max-count-codelist (plist (*initial*), *atom-list*)
 $\nless$ 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

$\text{car}(\text{gensym}(\text{initial}, \text{num-list}, \text{atom-list})) \notin \text{atom-list}$

;; 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:

$\text{lr-make-temp-name-alist-1}(\text{initial}, \text{num-list}, \text{temp-list}, \text{formals})$

```
=  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:

$\text{lr-make-temp-name-alist}(\text{temp-list}, \text{formals})$

= $\text{lr-make-temp-name-alist-1}(\text{unpack}('t*), \text{LIST-ASCII-0}, \text{temp-list}, \text{formals})$

DEFINITION:

$\text{lr-new-cons}(\text{car}, \text{cdr})$

= $\text{lr-new-node}(\text{tag}('nat, \text{LR-CONS-TAG}), \text{tag}('nat, 1), \text{car}, \text{cdr})$

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

DEFINITION:

$\text{deposit-a-list}(\text{list}, \text{addr}, \text{data-seg})$

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

DEFINITION:

$\text{lr-init-heap-contents}(\text{addr}, \text{size})$

```
=  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)
     $\neq$  (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
 $\text{count-list}(flag, object) \neq 0$

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

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

; 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))))),
                                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 (cдар (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, caddr (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  $\simeq$  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-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  $\neg$  listp (pos) then 0
  elseif  $\neg$  listp (expr) then 0
  elseif car (pos)  $\simeq$  0 then 0
  elseif car (expr) = 'if
  then if car (pos)  $\simeq$  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) < (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) ∧ (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: $p\text{-cons-clock}(p) = 23$

EVENT: Disable $p\text{-cons-clock}$.

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

DEFINITION: $p\text{-false-clock}(p) = 3$

EVENT: Disable $p\text{-false-clock}$.

;; 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: $p\text{-falsep-clock}(p) = 6$

EVENT: Disable $p\text{-falsep-clock}$.

;; Takes an implicit arg

DEFINITION:

$P\text{-LISTP-CODE} = p\text{-recognizer-code}('listp, LR\text{-CONS-TAG})$

DEFINITION:

$p\text{-listp-clock}(p) = p\text{-recognizer-clock}(p, LR\text{-CONS-TAG})$

EVENT: Disable $p\text{-listp-clock}$.

DEFINITION:

$P\text{-NLISTP-CODE}$

```

= 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: $p\text{-nlistp-clock}(p) = 7$

EVENT: Disable $p\text{-nlistp-clock}$.

DEFINITION:

$P\text{-TRUE-CODE}$

$= \text{list}('true, 'nil, 'nil, \text{list}('push-constant, LR\text{-T-ADDR}), '(ret))$

DEFINITION: $p\text{-true-clock}(p) = 3$

EVENT: Disable p-true-clock.

```
;; The old code for TRUEP is shown below. I used to ensure that there was
;; only one occurrence 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('1)),
 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,
                                  caddr (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, cadr (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-addr (list)
 = **if** listp (list)
 then (car (list) = LR-UNDEF-ADDR) $\wedge$ all-undef-addr (cdr (list))
 else t endif

DEFINITION:

lr-programs-properp-1 (programs, program-names, table)
 = **if** listp (programs)
 then all-litatoms (formal-vars (car (programs)))
 $\wedge$ all-litatoms (strip-cars (temp-var-dcls (car (programs))))
 $\wedge$ all-undef-addr (strip-cadrs (temp-var-dcls (car (programs))))
 $\wedge$ lr-proper-exrp (t,
 program-body (car (programs)),
 program-names,
 formal-vars (car (programs)),
 strip-cars (temp-var-dcls (car (programs))),
 table)
 $\wedge$ lr-programs-properp-1 (cdr (programs), program-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))
 $\wedge$ (caar (p-prog-segment (l)) = 'main)
 $\wedge$ all-user-fnamesp (cdr (strip-cars (p-prog-segment (l))))
 $\wedge$ 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

$(flag \neq \text{'list}) \rightarrow (\text{lr-p-c-size}(flag, expr) \neq 0)$

THEOREM: difference-decreases

$((x \not< y) \wedge (y \not= 0)) \rightarrow (((x - y) < x) = t)$

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)))
^ (p-ctrl-stk (lr-set-expr (s1, s2, pos)) = p-ctrl-stk (s1))
^ (p-temp-stk (lr-set-expr (s1, s2, pos)) = p-temp-stk (s1))
^ (p-prog-segment (lr-set-expr (s1, s2, pos)) = p-prog-segment (s2))
^ (p-data-segment (lr-set-expr (s1, s2, pos)) = p-data-segment (s1))
^ (p-max-ctrl-stk-size (lr-set-expr (s1, s2, pos))
   = p-max-ctrl-stk-size (s1))
^ (p-max-temp-stk-size (lr-set-expr (s1, s2, pos))
   = p-max-temp-stk-size (s1))
^ (p-word-size (lr-set-expr (s1, s2, pos)) = p-word-size (s1))
^ (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))
^ (p-ctrl-stk (lr-set-tstk (s, ts)) = p-ctrl-stk (s))
^ (p-temp-stk (lr-set-tstk (s, ts)) = ts)
^ (p-prog-segment (lr-set-tstk (s, ts)) = p-prog-segment (s))
^ (p-data-segment (lr-set-tstk (s, ts)) = p-data-segment (s))
^ (p-max-ctrl-stk-size (lr-set-tstk (s, ts)) = p-max-ctrl-stk-size (s))
^ (p-max-temp-stk-size (lr-set-tstk (s, ts)) = p-max-temp-stk-size (s))
^ (p-word-size (lr-set-tstk (s, ts)) = p-word-size (s))
^ (p-psw (lr-set-tstk (s, ts)) = p-psw (s))

```

EVENT: Disable lr-set-tstk.

THEOREM: p-accessors-lr-set-error

$$\begin{aligned}
& (\text{p-pc}(\text{lr-set-error}(s, \text{flag})) = \text{p-pc}(s)) \\
& \wedge (\text{p-ctrl-stk}(\text{lr-set-error}(s, \text{flag})) = \text{p-ctrl-stk}(s)) \\
& \wedge (\text{p-temp-stk}(\text{lr-set-error}(s, \text{flag})) = \text{p-temp-stk}(s)) \\
& \wedge (\text{p-prog-segment}(\text{lr-set-error}(s, \text{flag})) = \text{p-prog-segment}(s)) \\
& \wedge (\text{p-data-segment}(\text{lr-set-error}(s, \text{flag})) = \text{p-data-segment}(s)) \\
& \wedge (\text{p-max-ctrl-stk-size}(\text{lr-set-error}(s, \text{flag})) = \text{p-max-ctrl-stk-size}(s)) \\
& \wedge (\text{p-max-temp-stk-size}(\text{lr-set-error}(s, \text{flag})) = \text{p-max-temp-stk-size}(s)) \\
& \wedge (\text{p-word-size}(\text{lr-set-error}(s, \text{flag})) = \text{p-word-size}(s)) \\
& \wedge (\text{p-psw}(\text{lr-set-error}(s, \text{flag})) = \text{flag})
\end{aligned}$$

EVENT: Disable lr-set-error.

THEOREM: p-accessors-lr-set-pos

$$\begin{aligned}
& (\text{p-pc}(\text{lr-set-pos}(s, \text{pos})) = \text{tag}('pc, \text{cons}(\text{area-name}(\text{p-pc}(s)), \text{pos}))) \\
& \wedge (\text{p-ctrl-stk}(\text{lr-set-pos}(s, \text{pos})) = \text{p-ctrl-stk}(s)) \\
& \wedge (\text{p-temp-stk}(\text{lr-set-pos}(s, \text{pos})) = \text{p-temp-stk}(s)) \\
& \wedge (\text{p-prog-segment}(\text{lr-set-pos}(s, \text{pos})) = \text{p-prog-segment}(s)) \\
& \wedge (\text{p-data-segment}(\text{lr-set-pos}(s, \text{pos})) = \text{p-data-segment}(s)) \\
& \wedge (\text{p-max-ctrl-stk-size}(\text{lr-set-pos}(s, \text{pos})) = \text{p-max-ctrl-stk-size}(s)) \\
& \wedge (\text{p-max-temp-stk-size}(\text{lr-set-pos}(s, \text{pos})) = \text{p-max-temp-stk-size}(s)) \\
& \wedge (\text{p-word-size}(\text{lr-set-pos}(s, \text{pos})) = \text{p-word-size}(s)) \\
& \wedge (\text{p-psw}(\text{lr-set-pos}(s, \text{pos})) = \text{p-psw}(s))
\end{aligned}$$

EVENT: Disable lr-set-pos.

THEOREM: p-accessors-lr-pop-tstk

$$\begin{aligned}
& (\text{p-pc}(\text{lr-pop-tstk}(s)) = \text{p-pc}(s)) \\
& \wedge (\text{p-ctrl-stk}(\text{lr-pop-tstk}(s)) = \text{p-ctrl-stk}(s)) \\
& \wedge (\text{p-prog-segment}(\text{lr-pop-tstk}(s)) = \text{p-prog-segment}(s)) \\
& \wedge (\text{p-data-segment}(\text{lr-pop-tstk}(s)) = \text{p-data-segment}(s)) \\
& \wedge (\text{p-max-ctrl-stk-size}(\text{lr-pop-tstk}(s)) = \text{p-max-ctrl-stk-size}(s)) \\
& \wedge (\text{p-max-temp-stk-size}(\text{lr-pop-tstk}(s)) = \text{p-max-temp-stk-size}(s)) \\
& \wedge (\text{p-word-size}(\text{lr-pop-tstk}(s)) = \text{p-word-size}(s))
\end{aligned}$$

THEOREM: p-temp-stk-lr-pop-tstk

$$\begin{aligned}
& \text{p-temp-stk}(\text{lr-pop-tstk}(s)) \\
& = \text{if listp}(\text{p-temp-stk}(s)) \wedge (\text{p-psw}(s) = 'run) \\
& \quad \text{then pop}(\text{p-temp-stk}(s)) \\
& \quad \text{else p-temp-stk}(s) \text{ endif}
\end{aligned}$$

EVENT: Disable lr-pop-tstk.

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

THEOREM: offset-tag
 $\text{offset}(\text{tag}(tag, adp)) = \text{adp-offset}(adp)$

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

THEOREM: p-current-program-lr-set-pos
 $\text{p-current-program}(\text{lr-set-pos}(s, pos)) = \text{p-current-program}(s)$

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

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

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

THEOREM: number-cons-lr-expr-t-list
 $(\text{listp}(\text{lr-expr-list}(p)) \wedge \text{listp}(\text{offset}(\text{p-pc}(p))))$
 $\rightarrow (\text{number-cons}(\text{lr-expr}(p)) < \text{number-cons}(\text{lr-expr-list}(p)))$

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

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

```
;; 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:
 $\text{lr-eval}(flag, l, c)$
 $= \text{if } \text{p-psw}(l) \neq \text{'run} \text{ then } l$
 $\quad \text{elseif } flag = \text{'list}$


```

then if offset (p-pc (l))  $\simeq$  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  $\simeq$  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)  $\simeq$  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))  $\neq$  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)
   $\not\leq$  (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))

```

$$\begin{aligned}
& \wedge \quad (\text{p-max-ctrl-stk-size}(\text{lr-funcall}(l, \text{new-}l)) \\
& \quad = \text{p-max-ctrl-stk-size}(\text{new-}l)) \\
& \wedge \quad (\text{p-max-temp-stk-size}(\text{lr-funcall}(l, \text{new-}l)) \\
& \quad = \text{p-max-temp-stk-size}(\text{new-}l)) \\
& \wedge \quad (\text{p-word-size}(\text{lr-funcall}(l, \text{new-}l)) = \text{p-word-size}(\text{new-}l))
\end{aligned}$$

THEOREM: p-accessors-lr-push-tstk

$$\begin{aligned}
& (\text{p-pc}(\text{lr-push-tstk}(s, v)) = \text{p-pc}(s)) \\
& \wedge \quad (\text{p-ctrl-stk}(\text{lr-push-tstk}(s, v)) = \text{p-ctrl-stk}(s)) \\
& \wedge \quad (\text{p-prog-segment}(\text{lr-push-tstk}(s, v)) = \text{p-prog-segment}(s)) \\
& \wedge \quad (\text{p-data-segment}(\text{lr-push-tstk}(s, v)) = \text{p-data-segment}(s)) \\
& \wedge \quad (\text{p-max-ctrl-stk-size}(\text{lr-push-tstk}(s, v)) = \text{p-max-ctrl-stk-size}(s)) \\
& \wedge \quad (\text{p-max-temp-stk-size}(\text{lr-push-tstk}(s, v)) = \text{p-max-temp-stk-size}(s)) \\
& \wedge \quad (\text{p-word-size}(\text{lr-push-tstk}(s, v)) = \text{p-word-size}(s))
\end{aligned}$$

THEOREM: p-accessors-lr-if-ok

$$\begin{aligned}
& (\text{p-pc}(\text{lr-if-ok}(l)) = \text{p-pc}(l)) \\
& \wedge \quad (\text{p-ctrl-stk}(\text{lr-if-ok}(l)) = \text{p-ctrl-stk}(l)) \\
& \wedge \quad (\text{p-temp-stk}(\text{lr-if-ok}(l)) = \text{p-temp-stk}(l)) \\
& \wedge \quad (\text{p-prog-segment}(\text{lr-if-ok}(l)) = \text{p-prog-segment}(l)) \\
& \wedge \quad (\text{p-data-segment}(\text{lr-if-ok}(l)) = \text{p-data-segment}(l)) \\
& \wedge \quad (\text{p-max-ctrl-stk-size}(\text{lr-if-ok}(l)) = \text{p-max-ctrl-stk-size}(l)) \\
& \wedge \quad (\text{p-max-temp-stk-size}(\text{lr-if-ok}(l)) = \text{p-max-temp-stk-size}(l)) \\
& \wedge \quad (\text{p-word-size}(\text{lr-if-ok}(l)) = \text{p-word-size}(l))
\end{aligned}$$

THEOREM: p-accessors-lr-set-temp

$$\begin{aligned}
& (\text{p-pc}(\text{lr-set-temp}(s, v, n)) = \text{p-pc}(s)) \\
& \wedge \quad (\text{p-ctrl-stk}(\text{lr-set-temp}(s, v, n)) \\
& \quad = \text{if } \text{p-psw}(s) = \text{'run'} \\
& \quad \quad \text{then set-local-var-value}(v, n, \text{p-ctrl-stk}(s)) \\
& \quad \quad \text{else p-ctrl-stk}(s) \text{ endif}) \\
& \wedge \quad (\text{p-temp-stk}(\text{lr-set-temp}(s, v, n)) = \text{p-temp-stk}(s)) \\
& \wedge \quad (\text{p-prog-segment}(\text{lr-set-temp}(s, v, n)) = \text{p-prog-segment}(s)) \\
& \wedge \quad (\text{p-data-segment}(\text{lr-set-temp}(s, v, n)) = \text{p-data-segment}(s)) \\
& \wedge \quad (\text{p-max-ctrl-stk-size}(\text{lr-set-temp}(s, v, n)) = \text{p-max-ctrl-stk-size}(s)) \\
& \wedge \quad (\text{p-max-temp-stk-size}(\text{lr-set-temp}(s, v, n)) = \text{p-max-temp-stk-size}(s)) \\
& \wedge \quad (\text{p-word-size}(\text{lr-set-temp}(s, v, n)) = \text{p-word-size}(s)) \\
& \wedge \quad (\text{p-psw}(\text{lr-set-temp}(s, v, n)) = \text{p-psw}(s))
\end{aligned}$$

THEOREM: p-accessors-lr-do-temp-fetch

$$\begin{aligned}
& (\text{p-pc}(\text{lr-do-temp-fetch}(s)) = \text{p-pc}(s)) \\
& \wedge \quad (\text{p-ctrl-stk}(\text{lr-do-temp-fetch}(s)) = \text{p-ctrl-stk}(s)) \\
& \wedge \quad (\text{p-prog-segment}(\text{lr-do-temp-fetch}(s)) = \text{p-prog-segment}(s)) \\
& \wedge \quad (\text{p-data-segment}(\text{lr-do-temp-fetch}(s)) = \text{p-data-segment}(s)) \\
& \wedge \quad (\text{p-max-ctrl-stk-size}(\text{lr-do-temp-fetch}(s)) = \text{p-max-ctrl-stk-size}(s))
\end{aligned}$$

$\wedge$ (p-max-temp-stk-size (lr-do-temp-fetch (s)) = p-max-temp-stk-size (s))
 $\wedge$ (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))
 $\wedge$ (p-ctrl-stk (lr-pop-cstk (s))
 = **if** p-psw (s) = 'run **then** pop (p-ctrl-stk (s))
 else p-ctrl-stk (s) **endif**)
 $\wedge$ (p-temp-stk (lr-pop-cstk (s)) = p-temp-stk (s))
 $\wedge$ (p-prog-segment (lr-pop-cstk (s)) = p-prog-segment (s))
 $\wedge$ (p-data-segment (lr-pop-cstk (s)) = p-data-segment (s))
 $\wedge$ (p-max-ctrl-stk-size (lr-pop-cstk (s)) = p-max-ctrl-stk-size (s))
 $\wedge$ (p-max-temp-stk-size (lr-pop-cstk (s)) = p-max-temp-stk-size (s))
 $\wedge$ (p-word-size (lr-pop-cstk (s)) = p-word-size (s))
 $\wedge$ (p-psw (lr-pop-cstk (s)) = p-psw (s))

THEOREM: lr-eval-if-p-psw-1

((*flag* $\neq$ 'list)
 $\wedge$ (p-psw (lr-eval (*flag*, l , c)) = 'run)
 $\wedge$ (car (lr-expr (l)) = 'if)
 $\wedge$ (p-psw (l) = 'run)
 $\wedge$ ($c \not\approx 0$)
 $\wedge$ 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-name*, *offset*)) = *offset*

THEOREM: p-accessors-lr->p

(p-pc (lr->p (l)) = lr-p-pc (l))
 $\wedge$ (p-ctrl-stk (lr->p (l)) = p-ctrl-stk (l))
 $\wedge$ (p-temp-stk (lr->p (l)) = p-temp-stk (l))
 $\wedge$ (p-prog-segment (lr->p (l)) = comp-programs (p-prog-segment (l)))
 $\wedge$ (p-data-segment (lr->p (l)) = p-data-segment (l))

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

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

THEOREM: caddr-nil-lr-p-pc
 caddr (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*))) $\in \mathbf{N}$

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*) $\vee$ 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*)
 $\wedge$ (p-ctrl-stk (p-halt (*p*, *psw*))) = p-ctrl-stk (*p*)
 $\wedge$ (p-temp-stk (p-halt (*p*, *psw*))) = p-temp-stk (*p*)
 $\wedge$ (p-prog-segment (p-halt (*p*, *psw*))) = p-prog-segment (*p*)
 $\wedge$ (p-data-segment (p-halt (*p*, *psw*))) = p-data-segment (*p*)
 $\wedge$ (p-max-ctrl-stk-size (p-halt (*p*, *psw*))) = p-max-ctrl-stk-size (*p*)
 $\wedge$ (p-max-temp-stk-size (p-halt (*p*, *psw*))) = p-max-temp-stk-size (*p*)
 $\wedge$ (p-word-size (p-halt (*p*, *psw*))) = p-word-size (*p*)
 $\wedge$ (p-psw (p-halt (*p*, *psw*))) = *psw*

EVENT: Disable p-halt.

THEOREM: p-accessors-p-set-pc

$$\begin{aligned}
& (\text{p-pc}(\text{p-set-pc}(p, pc)) = pc) \\
& \wedge (\text{p-ctrl-stk}(\text{p-set-pc}(p, pc)) = \text{p-ctrl-stk}(p)) \\
& \wedge (\text{p-temp-stk}(\text{p-set-pc}(p, pc)) = \text{p-temp-stk}(p)) \\
& \wedge (\text{p-prog-segment}(\text{p-set-pc}(p, pc)) = \text{p-prog-segment}(p)) \\
& \wedge (\text{p-data-segment}(\text{p-set-pc}(p, pc)) = \text{p-data-segment}(p)) \\
& \wedge (\text{p-max-ctrl-stk-size}(\text{p-set-pc}(p, pc)) = \text{p-max-ctrl-stk-size}(p)) \\
& \wedge (\text{p-max-temp-stk-size}(\text{p-set-pc}(p, pc)) = \text{p-max-temp-stk-size}(p)) \\
& \wedge (\text{p-word-size}(\text{p-set-pc}(p, pc)) = \text{p-word-size}(p)) \\
& \wedge (\text{p-psw}(\text{p-set-pc}(p, pc)) = \text{p-psw}(p))
\end{aligned}$$

EVENT: Disable p-set-pc.

THEOREM: p-psw-not-run

$$(\text{p-psw}(p\text{-state}) \neq \text{'run'}) \rightarrow (\text{p}(p\text{-state}, \text{clock}) = p\text{-state})$$

THEOREM: p-psw-p-halt-x-y-error-msg

$$\begin{aligned}
& \text{p}(\text{p-halt}(p\text{-state}, \text{x-y-error-msg}(x, y)), n) \\
& = \text{p-halt}(p\text{-state}, \text{x-y-error-msg}(x, y))
\end{aligned}$$

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

THEOREM: p-accessors-p-run-subr

$$\begin{aligned}
& (\text{p-prog-segment}(\text{p-run-subr}(subr, p)) = \text{p-prog-segment}(p)) \\
& \wedge (\text{p-max-ctrl-stk-size}(\text{p-run-subr}(subr, p)) = \text{p-max-ctrl-stk-size}(p)) \\
& \wedge (\text{p-max-temp-stk-size}(\text{p-run-subr}(subr, p)) = \text{p-max-temp-stk-size}(p)) \\
& \wedge (\text{p-word-size}(\text{p-run-subr}(subr, p)) = \text{p-word-size}(p))
\end{aligned}$$

THEOREM: p-accessors-lr-apply-subr

$$\begin{aligned}
& (\text{p-pc}(\text{lr-apply-subr}(l1, l2)) = \text{p-pc}(l2)) \\
& \wedge (\text{p-prog-segment}(\text{lr-apply-subr}(l1, l2)) = \text{p-prog-segment}(l2)) \\
& \wedge (\text{p-max-ctrl-stk-size}(\text{lr-apply-subr}(l1, l2)) \\
& \quad = \text{p-max-ctrl-stk-size}(l2)) \\
& \wedge (\text{p-max-temp-stk-size}(\text{lr-apply-subr}(l1, l2)) \\
& \quad = \text{p-max-temp-stk-size}(l2)) \\
& \wedge (\text{p-word-size}(\text{lr-apply-subr}(l1, l2)) = \text{p-word-size}(l2))
\end{aligned}$$

THEOREM: p-prog-segment-lr-eval

$$\text{p-prog-segment}(\text{lr-eval}(flag, l, c)) = \text{p-prog-segment}(l)$$

THEOREM: p-max-ctrl-stk-size-lr-eval

$$\text{p-max-ctrl-stk-size}(\text{lr-eval}(flag, l, c)) = \text{p-max-ctrl-stk-size}(l)$$

THEOREM: p-max-temp-stk-size-lr-eval

$$\text{p-max-temp-stk-size}(\text{lr-eval}(flag, l, c)) = \text{p-max-temp-stk-size}(l)$$

THEOREM: p-word-size-lr-eval
 $\text{p-word-size}(\text{lr-eval}(flag, l, c)) = \text{p-word-size}(l)$

THEOREM: area-name-p-pc-lr-eval
 $\text{area-name}(\text{p-pc}(\text{lr-eval}(flag, l, c))) = \text{area-name}(\text{p-pc}(l))$

THEOREM: lr-programs-properp-lr-eval
 $\text{lr-programs-properp}(\text{lr-eval}(flag, l, c), table)$
 $= \text{lr-programs-properp}(l, table)$

THEOREM: definedp-deposit
 $\text{definedp}(tag, \text{deposit}(anything, addr, data-seg)) = \text{definedp}(tag, data-seg)$

THEOREM: deposit-a-list-cons-opener
 $\text{deposit-a-list}(\text{cons}(x, list), addr, data-seg)$
 $= \text{deposit}(x, addr, \text{deposit-a-list}(list, \text{add1-addr}(addr), data-seg))$

THEOREM: deposit-a-list-nil
 $\text{deposit-a-list}(\text{nil}, addr, data-seg) = data-seg$

EVENT: Disable deposit-a-list.

THEOREM: assoc-put-assoc-3
 $\text{assoc}(name1, \text{put-assoc}(val, name2, alist))$
 $= \text{if } name1 = name2$
 $\quad \text{then if definedp}(name1, alist) \text{ then } \text{cons}(name1, val)$
 $\quad \text{else f endif}$
 $\text{else } \text{assoc}(name1, alist) \text{ endif}$

EVENT: Disable assoc-put-assoc-3.

THEOREM: adpp-lessp-offset-deposit
 $((offset < \text{length}(\text{cdr}(\text{assoc}(name, data-seg)))) \wedge \text{definedp}(name, data-seg))$
 $\rightarrow (offset < \text{length}(\text{cdr}(\text{assoc}(name, \text{deposit}(anything, anywhere, data-seg))))))$

THEOREM: adpp-deposit-anything-at-all
 $\text{adpp}(adp, data-seg) \rightarrow \text{adpp}(adp, \text{deposit}(anything, addr2, data-seg))$

EVENT: Disable adpp-lessp-offset-deposit.

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

THEOREM: adpp-untag-definedp-area-name
 $\text{adpp}(\text{untag}(addr), data-seg) \rightarrow \text{definedp}(\text{area-name}(addr), data-seg)$

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

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

THEOREM: adpp-untag-numberp-offset
 $\text{adpp}(\text{untag}(addr), \text{data-seg}) \rightarrow (\text{offset}(addr) \in \mathbf{N})$

EVENT: Disable adpp-untag-numberp-offset.

THEOREM: adpp-untag-listp
 $\text{adpp}(\text{untag}(addr), \text{data-seg}) \rightarrow \text{listp}(\text{untag}(addr))$

EVENT: Disable adpp-untag-listp.

THEOREM: adpp-add-addr-0
 $(\text{adpp}(\text{untag}(addr), \text{data-seg})$
 $\wedge \text{caddr}(addr) = \mathbf{nil})$
 $\wedge (\text{type}(addr) = \mathbf{'addr})$
 $\wedge (n \simeq 0))$
 $\rightarrow (\text{add-addr}(addr, n) = addr)$

EVENT: Disable adpp-add-addr-0.

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

EVENT: Disable adpp-untag-lessp-offset.

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

EVENT: Disable adpp.

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

THEOREM: all-p-objectps-lr->p-similar-states

$$\begin{aligned}
& (\text{all-p-objectps } (lst, p0) \\
& \quad \wedge \quad (\text{p-word-size } (p0) = \text{p-word-size } (p1)) \\
& \quad \wedge \quad (\text{p-prog-segment } (p0) = \text{p-prog-segment } (p1)) \\
& \quad \wedge \quad \text{same-signature } (\text{p-data-segment } (p0), \text{p-data-segment } (p1))) \\
& \rightarrow \quad \text{all-p-objectps } (lst, p1)
\end{aligned}$$

THEOREM: proper-p-data-segmentp-lr->p-similar-states

$$\begin{aligned}
& (\text{proper-p-data-segmentp } (data-seg, p0) \\
& \quad \wedge \quad (\text{p-word-size } (p0) = \text{p-word-size } (p1)) \\
& \quad \wedge \quad (\text{p-prog-segment } (p0) = \text{p-prog-segment } (p1)) \\
& \quad \wedge \quad \text{same-signature } (\text{p-data-segment } (p0), \text{p-data-segment } (p1))) \\
& \rightarrow \quad \text{proper-p-data-segmentp } (data-seg, p1)
\end{aligned}$$

THEOREM: proper-p-temp-var-dclsp-lr->p-similar-states

$$\begin{aligned}
& (\text{proper-p-temp-var-dclsp } (temp-var-dcls, p0) \\
& \quad \wedge \quad (\text{p-word-size } (p0) = \text{p-word-size } (p1)) \\
& \quad \wedge \quad (\text{p-prog-segment } (p0) = \text{p-prog-segment } (p1)) \\
& \quad \wedge \quad \text{same-signature } (\text{p-data-segment } (p0), \text{p-data-segment } (p1))) \\
& \rightarrow \quad \text{proper-p-temp-var-dclsp } (temp-var-dcls, p1)
\end{aligned}$$

THEOREM: proper-p-instructionp-similar-p-states

$$\begin{aligned}
& (\text{proper-p-instructionp } (ins, name, p0) \\
& \quad \wedge \quad \text{same-signature } (\text{p-data-segment } (p0), \text{p-data-segment } (p1)) \\
& \quad \wedge \quad (\text{p-prog-segment } (p0) = \text{p-prog-segment } (p1)) \\
& \quad \wedge \quad (\text{p-word-size } (p0) = \text{p-word-size } (p1))) \\
& \rightarrow \quad \text{proper-p-instructionp } (ins, name, p1)
\end{aligned}$$

THEOREM: proper-labeled-p-instructionsp-lr->p-similar-states

$$\begin{aligned}
& (\text{proper-labeled-p-instructionsp } (lst, name, p0) \\
& \quad \wedge \quad \text{same-signature } (\text{p-data-segment } (p0), \text{p-data-segment } (p1)) \\
& \quad \wedge \quad (\text{p-prog-segment } (p0) = \text{p-prog-segment } (p1)) \\
& \quad \wedge \quad (\text{p-word-size } (p0) = \text{p-word-size } (p1))) \\
& \rightarrow \quad \text{proper-labeled-p-instructionsp } (lst, name, p1)
\end{aligned}$$

THEOREM: proper-p-prog-segmentp-lr->p-similar-states

$$\begin{aligned}
& (\text{proper-p-prog-segmentp } (programs, p0) \\
& \quad \wedge \quad \text{same-signature } (\text{p-data-segment } (p0), \text{p-data-segment } (p1)) \\
& \quad \wedge \quad (\text{p-prog-segment } (p0) = \text{p-prog-segment } (p1)) \\
& \quad \wedge \quad (\text{p-word-size } (p0) = \text{p-word-size } (p1))) \\
& \rightarrow \quad \text{proper-p-prog-segmentp } (programs, p1)
\end{aligned}$$

THEOREM: proper-p-temp-stkp-lr->p-similar-states

$$\begin{aligned}
& (\text{proper-p-temp-stkp } (temp-stk, p0) \\
& \quad \wedge \quad \text{same-signature } (\text{p-data-segment } (p0), \text{p-data-segment } (p1))
\end{aligned}$$

$\wedge$ (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

$(\text{all-user-fnamesp}(\text{strip-cars}(\text{programs})) \wedge (\neg \text{user-fnamep}(x)))$
 $\rightarrow (\neg \text{definedp}(x, \text{programs}))$

THEOREM: definitions-subrps-lr-programs-properp

lr-programs-properp(*l*, *table*)

$\rightarrow ((\text{assoc}(' \text{car}, \text{comp-programs}(\text{p-prog-segment}(l))) = \text{P-CAR-CODE})$
 $\wedge (\text{assoc}(' \text{cdr}, \text{comp-programs}(\text{p-prog-segment}(l))) = \text{P-CDR-CODE})$
 $\wedge (\text{assoc}(' \text{cons}, \text{comp-programs}(\text{p-prog-segment}(l))) = \text{P-CONS-CODE})$
 $\wedge (\text{assoc}(' \text{false}, \text{comp-programs}(\text{p-prog-segment}(l))) = \text{P-FALSE-CODE})$
 $\wedge (\text{assoc}(' \text{falsep}, \text{comp-programs}(\text{p-prog-segment}(l))) = \text{P-FALSEP-CODE})$
 $\wedge (\text{assoc}(' \text{listp}, \text{comp-programs}(\text{p-prog-segment}(l))) = \text{P-LISTP-CODE})$
 $\wedge (\text{assoc}(' \text{nlistp}, \text{comp-programs}(\text{p-prog-segment}(l))) = \text{P-NLISTP-CODE})$
 $\wedge (\text{assoc}(' \text{true}, \text{comp-programs}(\text{p-prog-segment}(l))) = \text{P-TRUE-CODE})$
 $\wedge (\text{assoc}(' \text{truep}, \text{comp-programs}(\text{p-prog-segment}(l))) = \text{P-TRUEP-CODE}))$

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

$\text{p-good-resultp}(\text{p-state}(pc,$
 $\quad \text{ctrl-stk},$
 $\quad \text{temp-stk},$
 $\quad \text{prog-seg},$
 $\quad \text{data-seg},$
 $\quad \text{max-ctrl-stk-size},$
 $\quad \text{max-temp-stk-size},$
 $\quad \text{word-size},$
 $\quad ' \text{run}),$
 $\quad \text{result-data-seg},$
 $\quad \text{result-temp-stk},$
 $\quad \text{result-ctrl-stk},$
 $\quad \text{result-pc})$
 $= ((\text{data-seg} = \text{result-data-seg})$

$$\begin{aligned}
& \wedge \quad (temp-stk = result-temp-stk) \\
& \wedge \quad listp(result-ctrl-stk) \\
& \wedge \quad (ctrl-stk = result-ctrl-stk) \\
& \wedge \quad (pc = result-pc)
\end{aligned}$$

THEOREM: p-good-resultp-p-halt-errorp-opener

$$\begin{aligned}
& (psw \neq 'run) \\
& \rightarrow \quad p-good-resultp(p-halt(p, psw), data-seg, temp-stk, ctrl-stk, pc)
\end{aligned}$$

EVENT: Disable p-good-resultp.

THEOREM: all-p-objectps-bad-type

$$\begin{aligned}
& ((get(offset, lst) \neq list(type(get(offset, lst)), untag(get(offset, lst)))) \\
& \wedge \quad (offset \in \mathbf{N}) \\
& \wedge \quad (offset < length(lst)) \\
& \rightarrow \quad (\neg all-p-objectps(lst, p))
\end{aligned}$$

THEOREM: proper-p-data-segmentp-bad-type

$$\begin{aligned}
& ((fetch(addr, data-seg) \\
& \neq \quad list(type(fetch(addr, data-seg)), untag(fetch(addr, data-seg)))) \\
& \wedge \quad adpp(untag(addr), data-seg)) \\
& \rightarrow \quad (\neg proper-p-data-segmentp(data-seg, p))
\end{aligned}$$

THEOREM: p-current-program-p-state

$$\begin{aligned}
& p-current-program(p-state(pc, \\
& \quad ctrl-stk, \\
& \quad temp-stk, \\
& \quad prog-seg, \\
& \quad data-seg, \\
& \quad max-ctrl-stk-size, \\
& \quad max-temp-stk-size, \\
& \quad word-size, \\
& \quad psw)) \\
& = \quad assoc(area-name(pc), prog-seg)
\end{aligned}$$

THEOREM: p-current-instruction-opener

$$\begin{aligned}
& p-current-instruction(p-state(pc, \\
& \quad temp-stk, \\
& \quad ctrl-stk, \\
& \quad prog-segment, \\
& \quad data-segment, \\
& \quad max-ctrl-stk-size, \\
& \quad max-temp-stk-size, \\
& \quad word-size,
\end{aligned}$$

$$= \text{unlabel}(\text{get}(\text{offset}(pc), \text{program-body}(\text{assoc}(\text{area-name}(pc), \text{prog-segment}))))$$

EVENT: Disable p-current-instruction-opener.

THEOREM: fetch-deposit

$$\begin{aligned}
& ((\text{offset}(\text{addr1}) \in \mathbf{N}) \wedge (\text{offset}(\text{addr2}) \in \mathbf{N})) \\
& \rightarrow (\text{fetch}(\text{addr1}, \text{deposit}(\text{value}, \text{addr2}, \text{data-seg})) \\
& \quad = \text{if definedp}(\text{area-name}(\text{addr2}), \text{data-seg}) \\
& \quad \quad \text{then if area-name}(\text{addr1}) = \text{area-name}(\text{addr2}) \\
& \quad \quad \quad \text{then if offset}(\text{addr1}) = \text{offset}(\text{addr2}) \text{ then value} \\
& \quad \quad \quad \quad \text{else fetch}(\text{addr1}, \text{data-seg}) \text{ endif} \\
& \quad \quad \quad \text{else fetch}(\text{addr1}, \text{data-seg}) \text{ endif} \\
& \quad \text{else fetch}(\text{addr1}, \text{data-seg}) \text{ endif}
\end{aligned}$$

```
;; add-addr
```

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

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

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

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

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

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

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

THEOREM: `cddr-add-addr`
`cddr (add-addr (addr, n)) = 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
 $\text{listp}(\text{untag}(\text{lr-return-pc}(l)))$

THEOREM: type-lr-return-pc
 $\text{type}(\text{lr-return-pc}(l)) = \text{'pc}$

THEOREM: caddr-lr-return-pc
 $\text{caddr}(\text{lr-return-pc}(l)) = \mathbf{nil}$

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

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

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

:: sub-addr

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

THEOREM: caddr-sub-addr
 $\text{caddr}(\text{sub-addr}(addr, n)) = \mathbf{nil}$

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

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

THEOREM: offset-sub-addr
 $\text{offset}(\text{sub-addr}(addr, n)) = (\text{offset}(addr) - n)$

:: LR-BOUNDARY-NODEP

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

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

EVENT: Disable lr-boundary-nodep.

;; LR-NODEP

THEOREM: lr-noddep-opener

lr-noddep (*addr*, *data-seg*)
 = ((type (*addr*) = 'addr)
 $\wedge$ (cddr (*addr*) = nil)
 $\wedge$ listp (*addr*)
 $\wedge$ adpp (untag (*addr*), *data-seg*)
 $\wedge$ lr-boundary-noddep (*addr*)
 $\wedge$ (area-name (*addr*) = identity (LR-HEAP-NAME))))

EVENT: Disable lr-noddep.

;; LR-GOOD-POINTERP

THEOREM: lr-good-pointerp-opener

lr-good-pointerp (*addr*, *data-seg*)
 = ((type (*addr*) = 'addr)
 $\wedge$ (cddr (*addr*) = nil)
 $\wedge$ listp (*addr*)
 $\wedge$ adpp (untag (*addr*), *data-seg*)
 $\wedge$ lr-boundary-noddep (*addr*)
 $\wedge$ (area-name (*addr*) = identity (LR-HEAP-NAME))
 $\wedge$ (type (fetch (add-addr (*addr*, identity (LR-REF-COUNT-OFFSET)),
 data-seg)))
 = 'nat)))

EVENT: Disable lr-good-pointerp.

THEOREM: equal-plus-remainder-0-fact

(((*offset1* mod *max*) = 0)
 $\wedge$ ((*offset2* mod *max*) = 0)
 $\wedge$ (*n* < *max*)
 $\wedge$ (*m* < *max*)
 $\wedge$ (*offset1* $\in \mathbf{N}$)
 $\wedge$ (*offset2* $\in \mathbf{N}$))
 $\rightarrow$ (((*n* + *offset1*) = (*m* + *offset2*))
 = ((fix (*n*) = fix (*m*)) $\wedge$ (*offset1* = *offset2*))))

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

(lr-boundary-offsetp (*offset1*)
 $\wedge$ lr-boundary-offsetp (*offset2*)
 $\wedge$ (*n* < LR-NODE-SIZE)
 $\wedge$ (*m* < LR-NODE-SIZE)

$$\begin{aligned}
& \wedge \text{ (offset1} \in \mathbf{N}) \\
& \wedge \text{ (offset2} \in \mathbf{N}) \\
\rightarrow & \text{ (((n + offset1) = (m + offset2))} \\
& \quad = \text{ ((fix(n) = fix(m))} \wedge \text{ (offset1 = offset2)))}
\end{aligned}$$

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

$$\begin{aligned}
& \text{(good-posp ('list, pos, body))} \\
& \wedge \text{ listp (pos)} \\
& \wedge \text{ (car (last (pos)) < length (cur-expr (butlast (pos), body)))} \\
\rightarrow & \text{ (good-posp ('list, nx (pos), body) } \wedge \text{ good-posp1 (pos, body))}
\end{aligned}$$

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

$$\begin{aligned}
& \text{(lr-programs-properp-1 (progs, program-names, table) } \wedge \text{ (prog} \in \text{progs))} \\
\rightarrow & \text{ lr-proper-exprp (t,} \\
& \quad \text{program-body (prog),} \\
& \quad \text{program-names,} \\
& \quad \text{formal-vars (prog),} \\
& \quad \text{strip-cars (temp-var-dcls (prog)),} \\
& \quad \text{table)}
\end{aligned}$$

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

$$\begin{aligned}
& \text{lr-proper-exprp ('list, expr, pnames, formals, temps, table)} \\
\rightarrow & \text{ (lr-proper-exprp (t, get (n, expr), pnames, formals, temps, table)} \\
& \quad = \text{ (n < length (expr)))}
\end{aligned}$$

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

$$\begin{aligned}
& \text{((car (expr) } \neq \text{ S-TEMP-FETCH)} \\
& \wedge \text{ (car (expr) } \neq \text{ S-TEMP-EVAL)} \\
& \wedge \text{ (car (expr) } \neq \text{ S-TEMP-TEST)} \\
& \wedge \text{ (car (expr) } \neq \text{ 'quote)} \\
& \wedge \text{ (n} \neq \text{0)} \\
& \wedge \text{ lr-proper-exprp (t, expr, pnames, formals, temps, table))} \\
\rightarrow & \text{ (lr-proper-exprp (t, get (n, expr), pnames, formals, temps, table)} \\
& \quad = \text{ (n < length (expr)))}
\end{aligned}$$

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

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

$$\begin{aligned}
& \text{(lr-proper-exprp (t, body, pnames, formals, temps, table) } \wedge \text{ good-posp1 (pos, body))} \\
\rightarrow & \text{ lr-proper-exprp (t, cur-expr (pos, body), pnames, formals, temps, table)}
\end{aligned}$$

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

$$\begin{aligned}
& \text{(lr-programs-properp (l, table) } \wedge \text{ (prog-seg = p-prog-segment (l)))} \\
\rightarrow & \text{ (lr-programs-properp-1 (p-prog-segment (l),} \\
& \quad \text{strip-logic-fnames (cdr (prog-seg))),}
\end{aligned}$$

$$\wedge \text{ definedp}(\text{area-name}(\text{p-pc}(l)), \text{prog-seg}))$$

THEOREM: lr-programs-properp-lr-proper-exrp-lr-expr
 (lr-programs-properp(l , $table$)
 $\wedge$ good-posp1(offset(p-pc(l)), program-body(p-current-program(l)))
 $\rightarrow$ lr-proper-exrp(t ,
 lr-expr(l),
 strip-logic-fnames(cdr(p-prog-segment(l))),
 formal-vars(p-current-program(l)),
 strip-cars(temp-var-dcls(p-current-program(l))),
 $table$))

THEOREM: lr-proper-exrp-length-cur-expr
 (lr-proper-exrp(t , $expr$, $pnames$, $formals$, $temps$, $table$)
 $\wedge$ listp($expr$)
 $\wedge$ (subrp(car($expr$)) $\vee$ body(car($expr$)))
 $\wedge$ (car($expr$) $\neq$ 'quote)
 $\rightarrow$ (length($expr$) = (1 + arity(car($expr$))))

THEOREM: listp-comp-body-1
 listp(comp-body-1($flag$, $body$, n))
 = if $flag = \text{'list}$ then listp($body$)
 else t endif

THEOREM: car-append
 listp(x) $\rightarrow$ (car(append(x , y)) = car(x))

THEOREM: length-cdr-comp-if-comp-body
 length(comp-if(comp-body-1(t , $test$, $n1$),
 comp-body-1(t , $then$, $n2$),
 comp-body-1(t , $else$, $n3$),
 n))
 = (length(comp-body-1(t , $test$, $n1$))
 + length(comp-body-1(t , $then$, $n2$))
 + length(comp-body-1(t , $else$, $n3$))
 + 4)

THEOREM: lr-p-c-size-list-0-opener
 lr-p-c-size-list(0, $expr$) = 0

THEOREM: lr-p-c-size-list-add1-opener
 ((1 + n) < length($expr$))
 $\rightarrow$ (lr-p-c-size-list(1 + n , $expr$)
 = (lr-p-c-size(t , cadr($expr$)) + lr-p-c-size-list(n , cdr($expr$))))

THEOREM: length-comp-body-1-lr-p-c-size
 $\text{length}(\text{comp-body-1}(\text{flag}, \text{body}, n)) = \text{lr-p-c-size}(\text{flag}, \text{body})$

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

THEOREM: length-label-instrs
 $\text{length}(\text{label-instrs}(\text{instrs}, n)) = \text{length}(\text{instrs})$

THEOREM: length-comp-body-lr-p-c-size
 $\text{length}(\text{comp-body}(\text{body})) = (1 + \text{lr-p-c-size}(\mathbf{t}, \text{body}))$

THEOREM: lr-p-c-size-flag-list
 $\text{lr-p-c-size}(\text{'list}, \text{cdr}(\text{expr})) = \text{lr-p-c-size-list}(\text{length}(\text{expr}) - 1, \text{expr})$

THEOREM: lr-proper-exrp-car-if-cadr
 $(\text{lr-proper-exrp}(\mathbf{t}, \text{body}, \text{pnames}, \text{formals}, \text{temps}, \text{table})$
 $\wedge \text{listp}(\text{body})$
 $\wedge (\text{car}(\text{body}) = \text{'if}))$
 $\rightarrow \text{lr-proper-exrp}(\mathbf{t}, \text{cadr}(\text{body}), \text{pnames}, \text{formals}, \text{temps}, \text{table})$

THEOREM: lr-proper-exrp-car-if-caddr
 $(\text{lr-proper-exrp}(\mathbf{t}, \text{body}, \text{pnames}, \text{formals}, \text{temps}, \text{table})$
 $\wedge \text{listp}(\text{body})$
 $\wedge (\text{car}(\text{body}) = \text{'if}))$
 $\rightarrow \text{lr-proper-exrp}(\mathbf{t}, \text{caddr}(\text{body}), \text{pnames}, \text{formals}, \text{temps}, \text{table})$

THEOREM: lr-proper-exrp-car-if-caddr
 $(\text{lr-proper-exrp}(\mathbf{t}, \text{body}, \text{pnames}, \text{formals}, \text{temps}, \text{table})$
 $\wedge \text{listp}(\text{body})$
 $\wedge (\text{car}(\text{body}) = \text{'if}))$
 $\rightarrow \text{lr-proper-exrp}(\mathbf{t}, \text{caddr}(\text{body}), \text{pnames}, \text{formals}, \text{temps}, \text{table})$

THEOREM: good-posp-list-t-offset-program-body
 $(\text{good-posp}(\text{'list}, \text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{listp}(\text{lr-expr-list}(l))$
 $\wedge \text{listp}(\text{offset}(\text{p-pc}(l))))$
 $\rightarrow \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))$

THEOREM: good-posp-list-nx-offset-program-body
 $(\text{good-posp}(\text{'list}, \text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{listp}(\text{offset}(\text{p-pc}(l)))$
 $\wedge \text{listp}(\text{lr-expr-list}(l)))$
 $\rightarrow \text{good-posp}(\text{'list},$
 $\quad \text{nx}(\text{offset}(\text{p-pc}(l))),$
 $\quad \text{program-body}(\text{p-current-program}(l)))$

→ ((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*)
 ∧ (*n* ≠ 0)
 ∧ (lr-p-pc-1 (get (*n*, *body*), *pos*) < lr-p-c-size (**t**, get (*n*, *body*)))
 ∧ (lr-p-pc-1 (get (*n*, *body*), *pos*) ≠ 0))
 → (((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)
 → (p-max-temp-stk-size (*l*)
 ≠ 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*) ∧ definedp (*name*, *bindings*))
 → p-objectp (cdr (assoc (*name*, *bindings*)), *l*)

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

THEOREM: formal-vars-assoc-comp-programs
 definedp (*name*, *programs*)
 → (formal-vars (assoc (*name*, comp-programs (*programs*)))
 = formal-vars (assoc (*name*, *programs*)))

THEOREM: temp-var-dcls-assoc-comp-programs-1
 definedp (*name*, *programs*)
 → (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*)
→ (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*)
→ definedp (car (untag (p-pc (*l*))), p-prog-segment (*l*))

THEOREM: p-objectp-cdr-assoc-litatom-proper-p-alistp
(proper-p-alistp (*bindings*, *lp*)
^ good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
^ lr-programs-properp (*l*, *table*)
^ (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*))))))
^ litatom (lr-expr (*l*)))
→ p-objectp (cdr (assoc (lr-expr (*l*), *bindings*)), *lp*)

THEOREM: proper-p-temp-stkp-lr-push-tstk-assoc-bindings
(lr-programs-properp (*l*, *table*)
^ good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
^ proper-p-alistp (bindings (car (p-ctrl-stk (*l*))), lr->p (*l*))
^ litatom (lr-expr (*l*))
^ (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*))))))
→ (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*))
^ lr-programs-properp (*l*, *table*)
^ good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
^ (p-psw (lr-push-tstk (*l*,

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

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

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

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

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

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

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

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

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

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

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

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

$$\begin{aligned}
& (((\text{car (lr-expr } (l)) = \text{S-TEMP-EVAL}}) \vee (\text{car (lr-expr } (l)) = \text{S-TEMP-TEST})) \\
& \wedge \text{good-posp1 (offset (p-pc } (l)), \text{program-body (p-current-program } (l))}) \\
\rightarrow & \text{good-posp1 (dv (offset (p-pc } (l)), 1), \text{program-body (p-current-program } (l))})
\end{aligned}$$

THEOREM: proper-p-alistp-put-assoc

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

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) $\not\leq$ 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
 $((\text{p-psw}(\text{lr-do-temp-fetch}(l)) = \text{'run})$
 $\wedge \text{lr-programs-properp}(l, \text{table})$
 $\wedge ((\text{car}(\text{lr-expr}(l)) = \text{S-TEMP-FETCH})$
 $\vee (\text{car}(\text{lr-expr}(l)) = \text{S-TEMP-TEST}))$
 $\wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{proper-p-statep}(\text{lr->p}(l)))$
 $\rightarrow \text{proper-p-statep}(\text{lr->p}(\text{lr-do-temp-fetch}(l)))$

THEOREM: length-lr-push-tstk
 $(\text{p-psw}(\text{lr-push-tstk}(l, \text{object})) = \text{'run})$
 $\rightarrow (\text{p-max-temp-stk-size}(l)$
 $\quad \neq \text{length}(\text{p-temp-stk}(\text{lr-push-tstk}(l, \text{object}))))$

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

THEOREM: proper-p-prog-segmentp-append
 $\text{listp}(\text{segment1})$
 $\rightarrow (\text{proper-p-prog-segmentp}(\text{append}(\text{segment1}, \text{segment2}), p)$
 $\quad = (\text{proper-p-prog-segmentp}(\text{segment1}, p)$
 $\quad \wedge \text{proper-p-prog-segmentp}(\text{segment2}, p)))$

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

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

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

THEOREM: lessp-4-not-zerop-not-1-not-2-3
 $((n \neq 0) \wedge (n \neq 1) \wedge (n \neq 2) \wedge (n < 4)) \rightarrow (n = 3)$

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

$$\begin{aligned} & ((\text{car}(pos) \neq 0) \\ & \wedge (\text{car}(pos) \neq 1) \\ & \wedge (\text{car}(pos) \neq 2) \\ & \wedge (\text{car}(pos) < 4)) \\ \rightarrow & (\text{get}(\text{car}(pos), body) = \text{caddr}(body)) \end{aligned}$$

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

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

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

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

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

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

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

DEFINITION:

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

$$\begin{aligned} = & \text{ if } pos \simeq \mathbf{nil} \text{ then } \mathbf{t} \\ & \text{ elseif } expr \simeq \mathbf{nil} \text{ then } \mathbf{t} \\ & \text{ elseif } \text{car}(expr) = ' \text{ if} \\ & \text{ then let } then\text{-}n \text{ be } n + 3 + \text{lr-p-c-size}(\mathbf{t}, \text{cadr}(expr)) \\ & \quad \text{ in} \\ & \quad \text{ case on } \text{car}(pos): \\ & \quad \text{ case } = 1 \\ & \quad \text{ then induct-hint-7}(\text{cdr}(pos), \text{cadr}(expr), n) \\ & \quad \text{ case } = 2 \\ & \quad \text{ then induct-hint-7}(\text{cdr}(pos), \text{caddr}(expr), then\text{-}n) \\ & \quad \text{ otherwise induct-hint-7}(\text{cdr}(pos), \\ & \quad \quad \text{caddr}(expr), \\ & \quad \quad 1 \\ & \quad \quad + \text{ then-}n \\ & \quad \quad + \text{lr-p-c-size}(\mathbf{t}, \text{caddr}(expr))) \text{ endcase endlet} \end{aligned}$$

```

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

```

THEOREM: $\text{lr-p-c-size-s-temp-test-eval-cadr-not-lessp-fact}$

```

(listp  $expr$ )
 $\wedge ((\text{car}(expr) = \text{S-TEMP-EVAL}) \vee (\text{car}(expr) = \text{S-TEMP-TEST}))$ 
 $\rightarrow (\text{lr-p-c-size}(\mathbf{t}, \text{cadr}(expr)) < \text{lr-p-c-size}(\mathbf{t}, expr))$ 

```

THEOREM: $\text{length-comp-temp-test}$

```

(listp  $body$ )  $\wedge (\text{car}(body) = \text{S-TEMP-TEST})$ 
 $\rightarrow (\text{length}(\text{comp-temp-test}(\text{any-body}, \text{comp-body-1}(\mathbf{t}, \text{cadr}(body), n), \text{any-n}))$ 
     $= \text{lr-p-c-size}(\mathbf{t}, body))$ 

```

THEOREM: $\text{plistp-comp-temp-test}$

```

plistp( $\text{comp-temp-test}(body, instrs, n)$ )

```

THEOREM: $\text{length-comp-if-alt}$

```

(listp  $body$ )  $\wedge (\text{car}(body) = \text{'if})$ 
 $\rightarrow (\text{length}(\text{comp-if}(\text{comp-body-1}(\mathbf{t}, \text{cadr}(body), n1),$ 
                      $\text{comp-body-1}(\mathbf{t}, \text{caddr}(body), n2),$ 
                      $\text{comp-body-1}(\mathbf{t}, \text{caddr}(body), n3),$ 
                      $\text{any-n}))$ 
     $= \text{lr-p-c-size}(\mathbf{t}, body))$ 

```

THEOREM: plistp-comp-if

```

(plistp  $else-instrs$ )  $\wedge \text{listp}(else-instrs)$ 
 $\rightarrow \text{plistp}(\text{comp-if}(\text{test-instrs}, \text{then-instrs}, \text{else-instrs}, n))$ 

```

THEOREM: $\text{plistp-comp-body-1}$

```

plistp( $\text{comp-body-1}(flag, body, n)$ )

```

THEOREM: $\text{lr-p-c-size-list-funcall-not-lessp-fact}$

```

(listp  $expr$ )
 $\wedge (\text{car}(expr) \neq \text{S-TEMP-FETCH})$ 
 $\wedge (\text{car}(expr) \neq \text{S-TEMP-EVAL})$ 
 $\wedge (\text{car}(expr) \neq \text{S-TEMP-TEST})$ 
 $\wedge (\text{car}(expr) \neq \text{'quote})$ 
 $\wedge (\text{car}(expr) \neq \text{'if})$ 
 $\rightarrow (\text{lr-p-c-size-list}(\text{length}(expr) - 1, expr) < \text{lr-p-c-size}(\mathbf{t}, expr))$ 

```

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

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-1
 $(\text{good-posp1}(\text{pos}, \text{cadr}(\text{body}))$
 $\wedge (\text{car}(\text{body}) = \text{'if})$
 $\wedge \text{listp}(\text{body})$
 $\wedge \text{lr-proper-exprp}(\mathbf{t}, \text{body}, \text{pnames}, \text{formals}, \text{temps}, \text{table}))$
 $\rightarrow (\text{firstn}(\text{lr-p-c-size}(\mathbf{t}, \text{cur-expr}(\text{pos}, \text{cadr}(\text{body}))),$
 $\quad \text{restn}(\text{lr-p-pc-1}(\text{cadr}(\text{body}), \text{pos}),$
 $\quad \text{comp-if}(\text{comp-body-1}(\mathbf{t}, \text{cadr}(\text{body}), n),$
 $\quad \quad \text{then-instrs},$
 $\quad \quad \text{else-instrs},$
 $\quad \quad n)))$
 $= \text{firstn}(\text{lr-p-c-size}(\mathbf{t}, \text{cur-expr}(\text{pos}, \text{cadr}(\text{body}))),$
 $\quad \text{restn}(\text{lr-p-pc-1}(\text{cadr}(\text{body}), \text{pos}),$
 $\quad \text{comp-body-1}(\mathbf{t}, \text{cadr}(\text{body}), n)))$

THEOREM: firstn-restn-plus-comp-if-1
 $((j = \text{length}(\text{test}))$
 $\wedge \text{listp}(\text{then})$
 $\wedge (m < \text{length}(\text{then}))$
 $\wedge (m \in \mathbf{N})$
 $\wedge \text{listp}(\text{test})$
 $\wedge (\text{length}(\text{then}) \not\leq (k + m)))$
 $\rightarrow (\text{firstn}(k, \text{restn}(3 + j + m, \text{comp-if}(\text{test}, \text{then}, \text{else}, n)))$
 $\quad = \text{firstn}(k, \text{restn}(m, \text{then})))$

THEOREM: firstn-unlabel-instrs-comp-body-1-lr-p-pc-1-helper-2
 $(\text{good-posp1}(\text{pos}, \text{caddr}(\text{body}))$
 $\wedge (\text{car}(\text{body}) = \text{'if})$
 $\wedge \text{listp}(\text{body})$
 $\wedge \text{lr-proper-exprp}(\mathbf{t}, \text{body}, \text{pnames}, \text{formals}, \text{temps}, \text{table}))$
 $\rightarrow (\text{lr-p-c-size}(\mathbf{t}, \text{caddr}(\text{body}))$
 $\quad \not\leq (\text{lr-p-pc-1}(\text{caddr}(\text{body}), \text{pos})$
 $\quad + \text{lr-p-c-size}(\mathbf{t}, \text{cur-expr}(\text{pos}, \text{caddr}(\text{body}))))$

THEOREM: firstn-restn-plus-comp-if-2
 $((j = \text{length}(\text{test}))$
 $\wedge (i = \text{length}(\text{then}))$
 $\wedge \text{listp}(\text{then})$
 $\wedge (m < \text{length}(\text{else}))$
 $\wedge (m \in \mathbf{N})$
 $\wedge \text{listp}(\text{test})$
 $\wedge \text{listp}(\text{else}))$

$$\begin{aligned}
& \wedge (\text{length}(\text{else}) \not\leq (m + k))) \\
\rightarrow & (\text{firstn}(k, \text{restn}(j + i + m + 4, \text{comp-if}(\text{test}, \text{then}, \text{else}, n))) \\
& = \text{firstn}(k, \text{restn}(m, \text{else})))
\end{aligned}$$

THEOREM: firstn-unlabel-instrs-comp-body-1-lr-p-pc-1-helper-3

$$\begin{aligned}
& (\text{good-posp1}(\text{pos}, \text{caddr}(\text{body}))) \\
& \wedge (\text{car}(\text{body}) = \text{'if}) \\
& \wedge \text{listp}(\text{body}) \\
& \wedge \text{lr-proper-exprp}(\mathbf{t}, \text{body}, \text{pnames}, \text{formals}, \text{temps}, \text{table})) \\
\rightarrow & (\text{lr-p-c-size}(\mathbf{t}, \text{caddr}(\text{body})) \\
& \not\leq (\text{lr-p-pc-1}(\text{caddr}(\text{body}), \text{pos}) \\
& + \text{lr-p-c-size}(\mathbf{t}, \text{cur-expr}(\text{pos}, \text{caddr}(\text{body}))))))
\end{aligned}$$

THEOREM: plus-constant-fact-helper-1

$$(1 + n + 3 + x + y) = (n + 4 + x + y)$$

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-4

$$\begin{aligned}
& (\text{good-posp1}(\text{pos}, \text{cadr}(\text{body}))) \\
& \wedge \text{listp}(\text{body}) \\
& \wedge (\text{car}(\text{body}) = \text{S-TEMP-TEST}) \\
& \wedge \text{lr-proper-exprp}(\mathbf{t}, \text{body}, \text{pnames}, \text{formals}, \text{temps}, \text{table})) \\
\rightarrow & (\text{firstn}(\text{lr-p-c-size}(\mathbf{t}, \text{cur-expr}(\text{pos}, \text{cadr}(\text{body}))), \\
& \text{restn}(\text{lr-p-pc-1}(\text{cadr}(\text{body}), \text{pos}) + 4, \\
& \text{comp-temp-test}(\text{body-1}, \text{comp-body-1}(\mathbf{t}, \text{cadr}(\text{body}), n), m))) \\
& = \text{firstn}(\text{lr-p-c-size}(\mathbf{t}, \text{cur-expr}(\text{pos}, \text{cadr}(\text{body}))), \\
& \text{restn}(\text{lr-p-pc-1}(\text{cadr}(\text{body}), \text{pos}), \\
& \text{comp-body-1}(\mathbf{t}, \text{cadr}(\text{body}), n))))
\end{aligned}$$

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-5

$$(4 + x) = (x + 4)$$

THEOREM: good-posp1-lr-proper-exprp-get-caddr

$$\begin{aligned}
& (\text{listp}(\text{pos})) \\
& \wedge \text{listp}(\text{body}) \\
& \wedge (\text{car}(\text{body}) = \text{'if}) \\
& \wedge (\text{car}(\text{pos}) \neq 1) \\
& \wedge (\text{car}(\text{pos}) \neq 2) \\
& \wedge (\text{car}(\text{pos}) \neq 0) \\
& \wedge (\text{car}(\text{pos}) \in \mathbf{N}) \\
& \wedge \text{lr-proper-exprp}(\mathbf{t}, \text{body}, \text{pnames}, \text{formals}, \text{temps}, \text{table})) \\
& \wedge (((\text{car}(\text{pos}) - 1) - 1) - 1) < \text{length}(\text{caddr}(\text{body}))) \\
\rightarrow & (\text{get}(\text{car}(\text{pos}), \text{body}) = \text{caddr}(\text{body}))
\end{aligned}$$

THEOREM: lr-proper-exprp-cadr-temps

$$\begin{aligned}
& (\text{lr-proper-exprp}(\mathbf{t}, \text{expr}, \text{pnames}, \text{formals}, \text{temps}, \text{table})) \\
& \wedge ((\text{car}(\text{expr}) = \text{S-TEMP-EVAL}) \vee (\text{car}(\text{expr}) = \text{S-TEMP-TEST})) \\
\rightarrow & \text{lr-proper-exprp}(\mathbf{t}, \text{cadr}(\text{expr}), \text{pnames}, \text{formals}, \text{temps}, \text{table})
\end{aligned}$$

THEOREM: lessp-plus-lr-p-c-size-lr-p-pc-1-temps
 (good-posp1 (*pos*, cadr (*body*))
 $\wedge$ listp (*body*)
 $\wedge$ ((car (*body*) = S-TEMP-EVAL) $\vee$ (car (*body*) = S-TEMP-TEST))
 $\wedge$ lr-proper-exprp (**t**, *body*, *pnames*, *formals*, *temps*, *table*)
 $\rightarrow$ (lr-p-c-size (**t**, cadr (*body*))
 $\not\prec$ (lr-p-pc-1 (cadr (*body*), *pos*)
 $+$ lr-p-c-size (**t**, cur-expr (*pos*, cadr (*body*))))))

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-6
 (listp (*body*)
 $\wedge$ (car (*body*) $\neq$ 'if)
 $\wedge$ (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)
 $\wedge$ lr-proper-exprp (**t**, *body*, *pnames*, *formals*, *temps*, *table*)
 $\wedge$ (*n* $\neq$ 0))
 $\rightarrow$ (lr-p-c-size-list (length (*body*) - 1, *body*)
 $\not\prec$ (lr-p-c-size-list (*n* - 1, *body*) + lr-p-pc-1 (get (*n*, *body*), *pos*)))

DEFINITION:
 induct-hint-10 (*n*, *l*, *x*)
 = **if** $\neg$ listp (*l*) **then** **t**
 elseif *n* $\simeq$ 0 **then** **t**
 elseif listp (cdr (*l*))
 then induct-hint-10 (*n* - 1, cdr (*l*), *x* + lr-p-c-size (**t**, cadr (*l*)))
 else **t** **endif**

THEOREM: lr-p-c-size-list-car-opener
 ((*n* $\neq$ 0) $\wedge$ (*n* < length (*body*)))
 $\rightarrow$ (lr-p-c-size-list (*n*, *body*)
 = (lr-p-c-size (**t**, cadr (*body*))
 $+$ lr-p-c-size-list (*n* - 1, cdr (*body*))))

THEOREM: restn-comp-body-1-list-fact
 ((lr-p-c-size (**t**, get (*m*, cdr (*body*))) $\not\prec$ *j*)
 $\wedge$ (*m* < length (cdr (*body*)))
 $\wedge$ (*m* $\in \mathbf{N}$)
 $\wedge$ (*n* $\in \mathbf{N}$)
 $\wedge$ (*j* $\in \mathbf{N}$))
 $\rightarrow$ (restn (lr-p-c-size-list (*m*, *body*) + *j*,
 comp-body-1 ('list, cdr (*body*), *n*))
 = restn (*j*,
 comp-body-1 ('list,

$$\text{restn}(m, \text{cdr}(\text{body})), \\ n + \text{lr-p-c-size-list}(m, \text{body}))))$$

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

THEOREM: firstn-restn-small-enough-cdr-comp-body-1-list
 $(\text{listp}(\text{body}) \wedge (\text{lr-p-c-size}(\mathbf{t}, \text{car}(\text{body})) \not\leq (j + k)))$
 $\rightarrow (\text{firstn}(j, \text{restn}(k, \text{comp-body-1}(\text{'list}, \text{body}, n)))$
 $= \text{firstn}(j, \text{restn}(k, \text{comp-body-1}(\mathbf{t}, \text{car}(\text{body}), n))))$

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-7
 $(\text{listp}(\text{body})$
 $\wedge (\text{car}(\text{body}) \neq \text{'if})$
 $\wedge (\text{car}(\text{body}) \neq \text{S-TEMP-FETCH})$
 $\wedge (\text{car}(\text{body}) \neq \text{S-TEMP-EVAL})$
 $\wedge (\text{car}(\text{body}) \neq \text{S-TEMP-TEST})$
 $\wedge (\text{car}(\text{body}) \neq \text{'quote})$
 $\wedge (n \in \mathbf{N})$
 $\wedge \text{lr-proper-exprp}(\mathbf{t}, \text{body}, \text{pnames}, \text{formals}, \text{temps}, \text{table})$
 $\wedge (m \neq 0)$
 $\wedge (m < \text{length}(\text{body}))$
 $\wedge \text{good-pospl}(\text{pos}, \text{get}(m, \text{body}))$
 $\rightarrow (\text{firstn}(\text{lr-p-c-size}(\mathbf{t}, \text{cur-expr}(\text{pos}, \text{get}(m, \text{body}))),$
 $\quad \text{restn}(\text{lr-p-c-size-list}(m - 1, \text{body})$
 $\quad \quad + \text{lr-p-pc-1}(\text{get}(m, \text{body}), \text{pos},$
 $\quad \quad \text{comp-body-1}(\text{'list}, \text{cdr}(\text{body}), n)))$
 $= \text{firstn}(\text{lr-p-c-size}(\mathbf{t}, \text{cur-expr}(\text{pos}, \text{get}(m, \text{body}))),$
 $\quad \text{restn}(\text{lr-p-pc-1}(\text{get}(m, \text{body}), \text{pos},$
 $\quad \quad \text{comp-body-1}(\mathbf{t},$
 $\quad \quad \quad \text{get}(m, \text{body}),$
 $\quad \quad \quad n + \text{lr-p-c-size-list}(m - 1, \text{body}))))$

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1-helper-8
 $(\text{listp}(\text{body})$
 $\wedge (\text{car}(\text{body}) \neq \text{'if})$
 $\wedge (\text{car}(\text{body}) \neq \text{S-TEMP-FETCH})$
 $\wedge (\text{car}(\text{body}) \neq \text{S-TEMP-EVAL})$
 $\wedge (\text{car}(\text{body}) \neq \text{S-TEMP-TEST})$
 $\wedge (\text{car}(\text{body}) \neq \text{'quote})$
 $\wedge \text{lr-proper-exprp}(\mathbf{t}, \text{body}, \text{pnames}, \text{formals}, \text{temps}, \text{table})$
 $\wedge (n \neq 0)$
 $\wedge (n < \text{length}(\text{body}))$
 $\wedge \text{good-pospl}(\text{pos}, \text{get}(n, \text{body}))$
 $\rightarrow ((\text{lr-p-c-size-list}(\text{length}(\text{body}) - 1, \text{body})$

$$\begin{aligned}
& - \text{lr-p-c-size-list}(n - 1, \text{body}) + \text{lr-p-pc-1}(\text{get}(n, \text{body}), \text{pos})) \\
& \not\leq \text{lr-p-c-size}(\mathbf{t}, \text{cur-expr}(\text{pos}, \text{get}(n, \text{body})))
\end{aligned}$$

THEOREM: firstn-lr-p-c-size-restn-lr-p-pc-1-comp-body-1
 (good-posp1(*pos*, *body*)
 $\wedge$ lr-proper-expr(*t*, *body*, *pnames*, *formals*, *temps*, *table*)
 $\wedge$ ($n \in \mathbf{N}$)
 $\rightarrow$ (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*) $\not\leq$ 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
 (($k < n$) $\wedge$ (firstn(*n*, *list1*) = firstn(*n*, *list2*)))
 $\rightarrow$ (get(*k*, *list1*) = get(*k*, *list2*))

THEOREM: unlabel-list-label
 unlabel(list('d1, *lab*, *comment*, *instr*)) = *instr*

THEOREM: legal-labelp-label-make-label
 legal-labelp(list('d1, 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
 ($m < \text{length}(\text{instrs})$)
 $\rightarrow$ (get(*m*, label-instrs(*instrs*, *n*))
 = list('d1, 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-pospl (pos, body)
  ∧ lr-proper-exrp (t, body, pnames, formals, temps, table)
  ∧ (n ∈ N)
  ∧ (m < lr-p-c-size (t, cur-expr (pos, body))))
→ (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-pospl (offset (p-pc (l)), program-body (prog))
  ∧ listp (lr-expr (l))
  ∧ (car (lr-expr (l)) = 'quote)
  ∧ lr-programs-properp (l, table)
  ∧ (prog = p-current-program (l)))
→ (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-pospl (offset (p-pc (l)), program-body (car (p-prog-segment (l))))
  ∧ listp (lr-expr (l))
  ∧ (car (lr-expr (l)) = 'quote)
  ∧ lr-programs-properp (l, table)
  ∧ (area-name (p-pc (l)) = caar (p-prog-segment (l))))
→ (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
 $\text{proper-p-temp-stkp}(\text{temp-stk}, p) \rightarrow \text{all-p-objectps}(\text{temp-stk}, p)$

THEOREM: all-p-objectps-first-n
 $((\text{length}(lst) \not\leq n) \wedge \text{all-p-objectps}(lst, p))$
 $\rightarrow \text{all-p-objectps}(\text{first-n}(n, lst), p)$

THEOREM: strip-cars-append
 $\text{strip-cars}(\text{append}(x, y)) = \text{append}(\text{strip-cars}(x), \text{strip-cars}(y))$

EVENT: Disable strip-cars-append.

THEOREM: strip-cars-pairlist
 $\text{strip-cars}(\text{pairlist}(x, y)) = \text{plist}(x)$

EVENT: Disable strip-cars-pairlist.

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

THEOREM: length-popn
 $(\text{length}(list) \not\leq n) \rightarrow (\text{length}(\text{popn}(n, list)) = (\text{length}(list) - n))$

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

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

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

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

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

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

THEOREM: length-pairlist
 $\text{length}(\text{pairlist}(x, y)) = \text{length}(x)$

THEOREM: length-pair-temps-with-initial-values
 $\text{length}(\text{pair-temps-with-initial-values}(\text{temp-var-dcls}))$
 $= \text{length}(\text{temp-var-dcls})$

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

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

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

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

```

    = '(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),
                                             topl(p-temp-stk(l)),

```

```

                                top (p-temp-stk (l)),
                                fetch (LR-FP-ADDR,
                                        p-data-segment (l)),
                                p-data-segment (l)),
                                p-data-segment (l))),
                                cons (fetch (LR-FP-ADDR, p-data-segment (l)),
                                        caddr (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))

```

$$\begin{aligned}
& \wedge \quad (\text{p-data-segment } (\text{p } (\text{p-set-pc } (\text{lr->p } (l), pc), \\
& \quad \quad \quad \text{p-car-clock } (\text{p-set-pc } (\text{lr->p } (l), pc)))) \\
& = \quad \text{p-data-segment } (l))
\end{aligned}$$

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

$$\begin{aligned}
& (\text{proper-p-statep } (\text{lr->p } (l)) \\
& \wedge \quad (\text{p-psw } (l) = \text{'run}) \\
& \wedge \quad (\text{p-psw } (\text{p } (\text{p-set-pc } (\text{lr->p } (l), pc), \text{p-cdr-clock } (\text{p-set-pc } (\text{lr->p } (l), pc)))) \\
& \quad = \text{'run}) \\
& \wedge \quad \text{lr-programs-properp } (l, \text{table}) \\
& \wedge \quad (\text{unlabel } (\text{get } (\text{offset } (pc), \\
& \quad \quad \quad \text{program-body } (\text{assoc } (\text{area-name } (pc), \\
& \quad \quad \quad \text{p-prog-segment } (\text{lr->p } (l))))) \\
& \quad = \text{'(call cdr)})) \\
\rightarrow & \quad ((\text{p-temp-stk } (\text{p } (\text{p-set-pc } (\text{lr->p } (l), pc), \\
& \quad \quad \quad \text{p-cdr-clock } (\text{p-set-pc } (\text{lr->p } (l), pc)))) \\
& \quad = \text{if fetch } (\text{top } (\text{p-temp-stk } (l)), \text{p-data-segment } (l)) \\
& \quad \quad = \text{tag } (\text{'nat}, \text{LR-CONS-TAG}) \\
& \quad \quad \text{then cons } (\text{fetch } (\text{add-addr } (\text{top } (\text{p-temp-stk } (l)), \text{LR-CDR-OFFSET}), \\
& \quad \quad \quad \text{p-data-segment } (l)), \\
& \quad \quad \quad \text{cdr } (\text{p-temp-stk } (l))) \\
& \quad \quad \text{else cons } (\text{LR-0-ADDR}, \text{cdr } (\text{p-temp-stk } (l))) \text{ endif}) \\
& \wedge \quad (\text{p-ctrl-stk } (\text{p } (\text{p-set-pc } (\text{lr->p } (l), pc), \\
& \quad \quad \quad \text{p-cdr-clock } (\text{p-set-pc } (\text{lr->p } (l), pc)))) \\
& \quad = \text{p-ctrl-stk } (l)) \\
& \wedge \quad (\text{p-data-segment } (\text{p } (\text{p-set-pc } (\text{lr->p } (l), pc), \\
& \quad \quad \quad \text{p-cdr-clock } (\text{p-set-pc } (\text{lr->p } (l), pc)))) \\
& \quad = \text{p-data-segment } (l))
\end{aligned}$$

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

$$\begin{aligned}
& \text{let } fp\text{-addr} \text{ be fetch } (\text{LR-FP-ADDR}, \text{p-data-segment } (l)) \\
& \text{in} \\
& (\text{proper-p-statep } (\text{lr->p } (l)) \\
& \wedge \quad (\text{p-psw } (l) = \text{'run}) \\
& \wedge \quad (\text{p-psw } (\text{p } (\text{p-set-pc } (\text{lr->p } (l), pc), \\
& \quad \quad \quad \text{p-cons-clock } (\text{p-set-pc } (\text{lr->p } (l), pc)))) \\
& \quad = \text{'run}) \\
& \wedge \quad \text{lr-programs-properp } (l, \text{table}) \\
& \wedge \quad (\text{unlabel } (\text{get } (\text{offset } (pc), \\
& \quad \quad \quad \text{program-body } (\text{assoc } (\text{area-name } (pc), \\
& \quad \quad \quad \text{p-prog-segment } (\text{lr->p } (l))))) \\
& \quad = \text{'(call cons)})) \\
\rightarrow & \quad ((\text{p-temp-stk } (\text{p } (\text{p-set-pc } (\text{lr->p } (l), pc), \\
& \quad \quad \quad \text{p-cons-clock } (\text{p-set-pc } (\text{lr->p } (l), pc))))
\end{aligned}$$

$$\begin{aligned}
&= \text{cons}(\text{fetch}(\text{LR-FP-ADDR}, \text{p-data-segment}(l)), \\
&\quad \text{caddr}(\text{p-temp-stk}(l))) \\
\wedge \quad &(\text{p-ctrl-stk}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \\
&\quad \text{p-cons-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
&= \text{p-ctrl-stk}(l) \\
\wedge \quad &(\text{p-data-segment}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \\
&\quad \text{p-cons-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
&= \text{deposit}(\text{fetch}(\text{add-addr}(\text{fetch}(\text{LR-FP-ADDR}, \\
&\quad \text{p-data-segment}(l)), \\
&\quad \text{LR-REF-COUNT-OFFSET}), \\
&\quad \text{p-data-segment}(l)), \\
&\quad \text{LR-FP-ADDR}, \\
&\quad \text{deposit-a-list}(\text{list}(\text{tag}('nat, \\
&\quad \text{LR-CONS-TAG}), \\
&\quad \text{tag}('nat, 1), \\
&\quad \text{top1}(\text{p-temp-stk}(l)), \\
&\quad \text{top}(\text{p-temp-stk}(l))), \\
&\quad \text{fetch}(\text{LR-FP-ADDR}, \\
&\quad \text{p-data-segment}(l)), \\
&\quad \text{p-data-segment}(l)))) \mathbf{endlet}
\end{aligned}$$

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

$$\begin{aligned}
&(\text{proper-p-statep}(\text{lr->p}(l)) \\
&\wedge \quad (\text{p-psw}(l) = \text{'run}) \\
&\wedge \quad (\text{p-psw}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \text{p-false-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
&\quad = \text{'run}) \\
&\wedge \quad \text{lr-programs-properp}(l, \text{table}) \\
&\wedge \quad (\text{unlabel}(\text{get}(\text{offset}(pc), \\
&\quad \text{program-body}(\text{assoc}(\text{area-name}(pc), \\
&\quad \text{p-prog-segment}(\text{lr->p}(l))))) \\
&\quad = \text{'(call false)})) \\
\rightarrow \quad &((\text{p-temp-stk}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \\
&\quad \text{p-false-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
&\quad = \text{cons}(\text{LR-F-ADDR}, \text{p-temp-stk}(l))) \\
&\wedge \quad (\text{p-ctrl-stk}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \\
&\quad \text{p-false-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
&\quad = \text{p-ctrl-stk}(l)) \\
&\wedge \quad (\text{p-data-segment}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \\
&\quad \text{p-false-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
&\quad = \text{p-data-segment}(l)))
\end{aligned}$$

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

$$\begin{aligned}
&(\text{proper-p-statep}(\text{lr->p}(l)) \\
&\wedge \quad (\text{p-psw}(l) = \text{'run})
\end{aligned}$$

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

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

$$\begin{aligned}
& \text{(proper-p-statep (lr->p (l))} \\
& \wedge \text{ (p-psw (l) = 'run)} \\
& \wedge \text{ (p-psw (p (p-set-pc (lr->p (l), pc), p-listp-clock (p-set-pc (lr->p (l), pc))))} \\
& \quad = \text{ 'run)} \\
& \wedge \text{ lr-programs-properp (l, table)} \\
& \wedge \text{ (unlabel (get (offset (pc),} \\
& \quad \text{program-body (assoc (area-name (pc),} \\
& \quad \quad \text{p-prog-segment (lr->p (l))))))} \\
& \quad = \text{ '(call listp))} \\
\rightarrow & \text{ ((p-temp-stk (p (p-set-pc (lr->p (l), pc),} \\
& \quad \text{p-listp-clock (p-set-pc (lr->p (l), pc))))} \\
& \quad = \text{ if fetch (car (p-temp-stk (l)), p-data-segment (l))} \\
& \quad \quad = \text{ tag ('nat, LR-CONS-TAG)} \\
& \quad \quad \text{then cons (LR-T-ADDR, cdr (p-temp-stk (l)))} \\
& \quad \quad \text{else cons (LR-F-ADDR, cdr (p-temp-stk (l))) endif)} \\
& \wedge \text{ (p-ctrl-stk (p (p-set-pc (lr->p (l), pc),} \\
& \quad \text{p-listp-clock (p-set-pc (lr->p (l), pc))))} \\
& \quad = \text{ p-ctrl-stk (l)} \\
& \wedge \text{ (p-data-segment (p (p-set-pc (lr->p (l), pc),} \\
& \quad \text{p-listp-clock (p-set-pc (lr->p (l), pc))))} \\
& \quad = \text{ p-data-segment (l))}
\end{aligned}$$

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

(proper-p-statep (lr->p (l))

$$\begin{aligned}
& \wedge \text{ (p-psw } (l) = \text{'run})} \\
& \wedge \text{ (p-psw (p (p-set-pc (lr->p } (l), pc), \text{ p-nlistp-clock (p-set-pc (lr->p } (l), pc))))} \\
& \quad = \text{'run})} \\
& \wedge \text{ lr-programs-properp } (l, table) \\
& \wedge \text{ (unlabel (get (offset } (pc), \\
& \quad \text{program-body (assoc (area-name } (pc), \\
& \quad \quad \text{p-prog-segment (lr->p } (l))))))} \\
& \quad = \text{'(call nlistp)}) \\
\rightarrow & \text{ ((p-temp-stk (p (p-set-pc (lr->p } (l), pc), \\
& \quad \text{p-nlistp-clock (p-set-pc (lr->p } (l), pc))))} \\
& \quad = \text{ if fetch (car (p-temp-stk } (l)), \text{ p-data-segment } (l))} \\
& \quad \quad = \text{ tag ('nat, LR-CONS-TAG)} \\
& \quad \quad \text{ then cons (LR-F-ADDR, cdr (p-temp-stk } (l)))} \\
& \quad \quad \text{ else cons (LR-T-ADDR, cdr (p-temp-stk } (l))) endif} \\
& \wedge \text{ (p-ctrl-stk (p (p-set-pc (lr->p } (l), pc), \\
& \quad \text{p-nlistp-clock (p-set-pc (lr->p } (l), pc))))} \\
& \quad = \text{ p-ctrl-stk } (l)) \\
& \wedge \text{ (p-data-segment (p (p-set-pc (lr->p } (l), pc), \\
& \quad \text{p-nlistp-clock (p-set-pc (lr->p } (l), pc))))} \\
& \quad = \text{ p-data-segment } (l)))
\end{aligned}$$

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

$$\begin{aligned}
& \text{(proper-p-statep (lr->p } (l))} \\
& \wedge \text{ (p-psw } (l) = \text{'run})} \\
& \wedge \text{ (p-psw (p (p-set-pc (lr->p } (l), pc), \text{ p-true-clock (p-set-pc (lr->p } (l), pc))))} \\
& \quad = \text{'run})} \\
& \wedge \text{ lr-programs-properp } (l, table) \\
& \wedge \text{ (unlabel (get (offset } (pc), \\
& \quad \text{program-body (assoc (area-name } (pc), \\
& \quad \quad \text{p-prog-segment (lr->p } (l))))))} \\
& \quad = \text{'(call true)}) \\
\rightarrow & \text{ ((p-temp-stk (p (p-set-pc (lr->p } (l), pc), \\
& \quad \text{p-true-clock (p-set-pc (lr->p } (l), pc))))} \\
& \quad = \text{ cons (LR-T-ADDR, p-temp-stk } (l))} \\
& \wedge \text{ (p-ctrl-stk (p (p-set-pc (lr->p } (l), pc), \\
& \quad \text{p-true-clock (p-set-pc (lr->p } (l), pc))))} \\
& \quad = \text{ p-ctrl-stk } (l)) \\
& \wedge \text{ (p-data-segment (p (p-set-pc (lr->p } (l), pc), \\
& \quad \text{p-true-clock (p-set-pc (lr->p } (l), pc))))} \\
& \quad = \text{ p-data-segment } (l)))
\end{aligned}$$

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

$$\begin{aligned}
& \text{(proper-p-statep (lr->p } (l))} \\
& \wedge \text{ (p-psw } (l) = \text{'run})}
\end{aligned}$$

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

THEOREM: get-last-funcall-cur-expr

$$\begin{aligned}
& \text{(listp (expr))} \\
& \wedge \text{ (car (expr) \neq 'if)} \\
& \wedge \text{ (car (expr) \neq S-TEMP-EVAL)} \\
& \wedge \text{ (car (expr) \neq S-TEMP-TEST)} \\
& \wedge \text{ (car (expr) \neq S-TEMP-FETCH)} \\
& \wedge \text{ (car (expr) \neq 'quote)} \\
\rightarrow & \text{ (get (lr-p-c-size-list (length (expr) - 1, expr), comp-body-1 (t, expr, n))} \\
& \quad = \text{ if definedp (car (expr), P-RUNTIME-SUPPORT-PROGRAMS)} \\
& \quad \quad \text{ then list ('call, car (expr))} \\
& \quad \quad \text{ else list ('call, user-fname (car (expr))) endif)}
\end{aligned}$$

THEOREM: not-listp-p-prog-segment-lr-expr

$$(\neg \text{listp (p-prog-segment (l))}) \rightarrow (\neg \text{listp (lr-expr (l))})$$

THEOREM: get-offset-return-pc-program-body-assoc-comp-programs

$$\begin{aligned}
& \text{(good-posp1 (offset (p-pc (l)),} \\
& \quad \text{program-body (assoc (area-name (p-pc (l)), p-prog-segment (l))))} \\
& \wedge \text{ lr-programs-properp (l, table)} \\
& \wedge \text{ (car (lr-expr (l)) \neq 'if)} \\
& \wedge \text{ (car (lr-expr (l)) \neq S-TEMP-EVAL)} \\
& \wedge \text{ (car (lr-expr (l)) \neq S-TEMP-TEST)} \\
& \wedge \text{ (car (lr-expr (l)) \neq S-TEMP-FETCH)} \\
& \wedge \text{ (car (lr-expr (l)) \neq 'quote)} \\
& \wedge \text{ listp (lr-expr (l))}
\end{aligned}$$

$\rightarrow$ (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)))
 $\wedge$ lr-programs-properp ($new-l$, $table$)
 $\wedge$ lr-programs-properp (l , $table$)
 $\wedge$ listp (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 ($new-l$)))
 $\rightarrow$ (listp (p-temp-stk (p-run-subr (car (lr-expr (l)),
 p-set-pc (lr->p ($new-l$), lr-return-pc (l))))))
 $\wedge$ (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)))
 $\wedge$ lr-programs-properp (l , $table$)
 $\wedge$ lr-programs-properp ($new-l$, $table$)
 $\wedge$ listp (lr-expr (l))
 $\wedge$ subrp (car (lr-expr (l)))
 $\wedge$ (p-psw (lr-apply-subr (l , $new-l$)) = 'run)
 $\wedge$ proper-p-statep (lr->p ($new-l$))
 $\wedge$ (p-psw ($new-l$) = 'run)
 $\wedge$ (p-prog-segment (l) = p-prog-segment ($new-l$)))
 $\rightarrow$ (listp (p-temp-stk (lr-apply-subr (l , $new-l$)))
 $\wedge$ (p-ctrl-stk (lr-apply-subr (l , $new-l$)) = p-ctrl-stk ($new-l$))

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

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

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

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

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

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

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

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

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

$\wedge$ 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.

$\wedge$ 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)
 $\wedge$ ($progs$ = 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) $\wedge$ (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 ($new-l$)))
 $\wedge$ 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)
 $\rightarrow$ 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) $\wedge$ (p-psw (p-run-subr ($subr$, p)) = 'run))
 $\rightarrow$ (p-max-temp-stk-size (p) $\not\leq$ length (p-temp-stk (p-run-subr ($subr$, p))))

THEOREM: not-lessp-length-p-temp-stk-lr-apply-subr
(proper-p-statep (lr->p ($new-l$)))
 $\wedge$ 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)
 $\wedge$ (p-max-temp-stk-size (l) = p-max-temp-stk-size ($new-l$))
 $\rightarrow$ (p-max-temp-stk-size (l)
 $\not\leq$ length (p-temp-stk (lr-apply-subr (l , $new-l$))))

THEOREM: proper-p-framep-top-p-ctrl-stk-lr-funcall
 $(\text{p-psw}(\text{lr-funcall}(l, \text{new-}l)) = \text{'run})$
 $\rightarrow (\text{listp}(\text{car}(\text{p-ctrl-stk}(\text{lr-funcall}(l, \text{new-}l))))$
 $\quad \wedge \text{listp}(\text{cdr}(\text{car}(\text{p-ctrl-stk}(\text{lr-funcall}(l, \text{new-}l))))$
 $\quad \wedge (\text{caddr}(\text{car}(\text{p-ctrl-stk}(\text{lr-funcall}(l, \text{new-}l)))) = \text{nil})$
 $\quad \wedge (\text{ret-pc}(\text{car}(\text{p-ctrl-stk}(\text{lr-funcall}(l, \text{new-}l))))$
 $\quad = \text{add-addr}(\text{lr-return-pc}(l, 1)))$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

EVENT: Disable popn-nlistp.

THEOREM: not-lessp-p-max-temp-stk-size-lr-funcall
 $((\text{p-max-temp-stk-size}(l) \not\leq \text{length}(\text{p-temp-stk}(\text{new-l})))$
 $\wedge (\text{p-max-temp-stk-size}(l) = \text{p-max-temp-stk-size}(\text{new-l})))$
 $\rightarrow (\text{p-max-temp-stk-size}(l) \not\leq \text{length}(\text{p-temp-stk}(\text{lr-funcall}(l, \text{new-l}))))$

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

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

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

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

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

THEOREM: proper-p-statep-lr-set-expr-lr-pop-cstk

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

THEOREM: p-psw-lr-eval-flag-list-flag-t

$$\begin{aligned}
& ((\text{p-psw}(\text{lr-eval}(\text{'list}, \text{lr-set-expr}(\text{lr-eval}(\mathbf{t}, l, c), l, \text{pos}), c)) = \text{'run}) \\
& \wedge \text{listp}(\text{offset}(\text{p-pc}(l))) \\
& \wedge \text{listp}(\text{lr-expr-list}(l))) \\
\rightarrow & (\text{p-psw}(\text{lr-eval}(\mathbf{t}, l, c)) = \text{'run})
\end{aligned}$$

THEOREM: lr-programs-properp-lr-set-expr

$$\begin{aligned}
& \text{lr-programs-properp}(\text{lr-set-expr}(l1, l2, \text{pos}), \text{table}) \\
& = \text{lr-programs-properp}(l2, \text{table})
\end{aligned}$$

THEOREM: lr-programs-properp-lr-pop-tstk

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

THEOREM: lr-programs-properp-lr-funcall

$$\begin{aligned}
& (\text{listp}(\text{lr-expr}(l)) \\
& \wedge (\neg \text{subrp}(\text{car}(\text{lr-expr}(l)))) \\
& \wedge (\text{car}(\text{lr-expr}(l)) \neq \text{'quote})
\end{aligned}$$

$\wedge$ 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:

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

EVENT: Disable lr-check-f-addrp.

DEFINITION:

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

EVENT: Disable lr-check-undef-addrp.

DEFINITION:

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

EVENT: Disable lr-check-numberp-addrp.

DEFINITION:

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

EVENT: Disable lr-check-listp-addrp.

DEFINITION:

```

lr-proper-heapp-nodop (addr, data-seg)
=  if  $\neg$  lr-nodop (addr, data-seg) then f
    elseif type (fetch (add-addr (addr, LR-REF-COUNT-OFFSET), data-seg))
        = 'addr
    then offset (addr)  $\not\leq$  (LR-NODE-SIZE + offset (LR-F-ADDR))
    elseif type (fetch (add-addr (addr, LR-REF-COUNT-OFFSET), data-seg))
         $\neq$  'nat then f
    elseif type (fetch (addr, data-seg))  $\neq$  'nat then f
    elseif untag (fetch (addr, data-seg)) = LR-UNDEFINED-TAG
    then lr-check-undef-addrp (addr, data-seg)
    elseif untag (fetch (addr, data-seg)) = LR-FALSE-TAG
    then lr-check-f-addrp (addr, data-seg)

```

```

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

```

EVENT: Disable lr-proper-heapp-nodep.

DEFINITION:

```

lr-proper-heapp2 (addr, data-seg)
= if offset (addr) < LR-NODE-SIZE then t
  else let sub-addr be sub-addr (addr, LR-NODE-SIZE)
    in
      lr-proper-heapp-nodep (sub-addr, data-seg)
  ^ lr-proper-heapp2 (sub-addr, data-seg) endlet endif

```

DEFINITION:

```

lr-valp (value, addr, data-seg)
= if lr-good-pointerp (addr, data-seg)
  then let tag be untag (fetch (addr, data-seg))
    in
      if listp (value)
        then (tag = LR-CONS-TAG)
          ^ lr-valp (car (value),
                    fetch (add-addr (addr, LR-CAR-OFFSET),
                             data-seg),
                    data-seg)
          ^ lr-valp (cdr (value),
                    fetch (add-addr (addr, LR-CDR-OFFSET),
                             data-seg),
                    data-seg)
        elseif truep (value) then tag = LR-TRUE-TAG
        elseif falsep (value) then tag = LR-FALSE-TAG
        elseif value ∈ N
          then (tag = LR-ADD1-TAG)
            ^ (value = untag (fetch (add-addr (addr,
                                                LR-UNBOX-NAT-OFFSET),
                                                data-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),

```

$$\begin{aligned} & \text{p-data-segment}(l) \\ \wedge & \text{lr-s-similar-temps}(s\text{-temps}, \\ & \quad \text{lr-temps}(\text{top}(\text{p-ctrl-stk}(l)), l), \\ & \quad \text{p-data-segment}(l)) \\ \wedge & \text{lr-s-similar-const-table}(table, \text{p-data-segment}(l)) \end{aligned}$$

EVENT: Disable lr-s-similar-statesp.

THEOREM: p-accessors-s->lr1

$$\begin{aligned} & (\text{p-pc}(s\text{->lr1}(s, l, table)) = \text{tag}('pc, \text{cons}(s\text{-pname}(s), s\text{-pos}(s)))) \\ \wedge & (\text{p-ctrl-stk}(s\text{->lr1}(s, l, table)) = \text{p-ctrl-stk}(l)) \\ \wedge & (\text{p-temp-stk}(s\text{->lr1}(s, l, table)) = \text{p-temp-stk}(l)) \\ \wedge & (\text{p-prog-segment}(s\text{->lr1}(s, l, table)) \\ & \quad = \text{lr-compile-programs}(s\text{-progs}(s), table)) \\ \wedge & (\text{p-data-segment}(s\text{->lr1}(s, l, table)) = \text{p-data-segment}(l)) \\ \wedge & (\text{p-max-ctrl-stk-size}(s\text{->lr1}(s, l, table)) = \text{p-max-ctrl-stk-size}(l)) \\ \wedge & (\text{p-max-temp-stk-size}(s\text{->lr1}(s, l, table)) = \text{p-max-temp-stk-size}(l)) \\ \wedge & (\text{p-word-size}(s\text{->lr1}(s, l, table)) = \text{p-word-size}(l)) \\ \wedge & (\text{p-psw}(s\text{->lr1}(s, l, table)) = s\text{-err-flag}(s)) \end{aligned}$$

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

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

;; OFFSET

THEOREM: offset-tag-cons

$$\text{offset}(\text{tag}(tag, \text{cons}(area, offset))) = offset$$

;; ADP-NAME

THEOREM: adp-name-cons

$$\text{adp-name}(\text{cons}(x, y)) = x$$

;; OFFSET-SUB-ADDR -- see above

;; LR-PROPER-P-AREASP

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

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

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

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

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

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

;; GET

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

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

;; LR-MINIMUM-HEAPP

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

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

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

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

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

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

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

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

THEOREM: lr-minimum-heapp-opener-2
 $\text{lr-minimum-heapp}(data\text{-}seg)$
 $\rightarrow (\text{identity}(\text{LR-MINIMUM-HEAP-SIZE})$
 $< \text{length}(\text{cdr}(\text{assoc}(\text{identity}(\text{LR-HEAP-NAME}), data\text{-}seg))))$

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

THEOREM: lr-minimum-heapp-opener-3
 $\text{lr-minimum-heapp}(data\text{-}seg) \rightarrow \text{definedp}(\text{identity}(\text{LR-HEAP-NAME}), data\text{-}seg)$

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

;; LR-PROPER-FREE-LISTP

DEFINITION:
 $\text{lr-node-listp}(list, data\text{-}seg)$
 $= \text{if listp}(list)$
 $\quad \text{then lr-nodep}(\text{car}(list), data\text{-}seg)$
 $\quad \quad \wedge \text{lr-node-listp}(\text{cdr}(list), data\text{-}seg)$
 $\quad \text{else t endif}$

EVENT: Disable lr-node-listp.

THEOREM: adpp-adpp-sub-addr
 $\text{adpp}(\text{untag}(addr), data\text{-}seg) \rightarrow \text{adpp}(\text{untag}(\text{sub-addr}(addr, n)), data\text{-}seg)$

THEOREM: lr-node-listp-lr-free-list-nodes
 $(\text{lr-boundary-nodep}(addr)$
 $\wedge (\text{area-name}(addr) = \text{LR-HEAP-NAME})$
 $\wedge \text{adpp}(\text{untag}(addr), data\text{-}seg2)$
 $\wedge (\text{type}(addr) = \text{'addr}))$
 $\rightarrow \text{lr-node-listp}(\text{lr-free-list-nodes}(addr, data\text{-}seg1), data\text{-}seg2)$

THEOREM: lr-nodep-member-lr-node-listp
 $(\text{lr-node-listp}(list, data\text{-}seg) \wedge (node \in list))$
 $\rightarrow ((\text{type}(node) = \text{'addr})$
 $\quad \wedge (\text{caddr}(node) = \mathbf{nil})$
 $\quad \wedge \text{listp}(node)$
 $\quad \wedge \text{adpp}(\text{untag}(node), data\text{-}seg)$
 $\quad \wedge \text{lr-boundary-nodep}(node)$
 $\quad \wedge (\text{area-name}(node) = \text{LR-HEAP-NAME}))$

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

THEOREM: lr-max-node-lr-noddep-opener-facts
(type (lr-max-node (*data-seg*)) = 'addr)
^ (caddr (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-noddep (lr-max-node (*data-seg*)))
→ ((type (fetch (identity (LR-FP-ADDR), *data-seg*)) = 'addr)
^ (caddr (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-noddep (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-noddep (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-noddep (lr-max-node (*data-seg*)))
→ listp (untag (fetch (identity (LR-FP-ADDR), *data-seg*)))

THEOREM: plus-times-fact-1

$$(n \neq 0) \rightarrow (((n + (n * w)) < (d * n)) = ((1 + w) < d))$$

EVENT: Disable plus-times-fact-1.

THEOREM: lessp-difference-fact-1

$$(((x \bmod n) = 0) \wedge ((y \bmod n) = 0) \wedge (x < y) \wedge (x \in \mathbf{N})) \\ \rightarrow ((x < (y - n)) = (x \neq (y - n)))$$

THEOREM: lessp-difference-lr-boundary-offsetp-fact-1

$$((offset \in \mathbf{N}) \\ \wedge \text{lr-boundary-offsetp}(offset) \\ \wedge \text{lr-boundary-offsetp}(y) \\ \wedge (offset < y)) \\ \rightarrow ((offset < (y - \text{identity}(\text{LR-NODE-SIZE}))) \\ = (offset \neq (y - \text{identity}(\text{LR-NODE-SIZE}))))$$

THEOREM: lessp-lr-node-on-boundaryp-node-size

$$(\text{lr-boundary-nodep}(addr) \wedge (offset(addr) \in \mathbf{N})) \\ \rightarrow ((offset(addr) < \text{identity}(\text{LR-NODE-SIZE})) = (offset(addr) = 0))$$

THEOREM: lessp-difference-node-size-sub-addr

$$((offset(addr) < offset(max-addr)) \\ \wedge (\text{area-name}(addr) = \text{area-name}(max-addr)) \\ \wedge \text{lr-boundary-nodep}(max-addr) \\ \wedge (\text{type}(max-addr) = \text{'addr'}) \\ \wedge (offset(max-addr) \in \mathbf{N}) \\ \wedge (\text{caddr}(max-addr) = \mathbf{nil}) \\ \wedge \text{lr-boundary-nodep}(addr) \\ \wedge (\text{type}(addr) = \text{'addr'}) \\ \wedge (offset(addr) \in \mathbf{N}) \\ \wedge (\text{caddr}(addr) = \mathbf{nil}) \\ \wedge \text{listp}(\text{untag}(addr))) \\ \rightarrow ((offset(addr) < (offset(max-addr) - \text{identity}(\text{LR-NODE-SIZE}))) \\ = (\text{sub-addr}(max-addr, \text{identity}(\text{LR-NODE-SIZE})) \neq addr))$$

THEOREM: lr-nodep-lr-proper-heapp-nodep

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

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

THEOREM: adpp-area-name-offset-same

$$\begin{aligned}
& (\text{listp}(\text{untag}(addr1))) \\
& \wedge (\text{offset}(addr1) \in \mathbf{N}) \\
& \wedge (\text{cddr}(addr1) = \mathbf{nil}) \\
& \wedge (\text{listp}(\text{untag}(addr2))) \\
& \wedge (\text{offset}(addr2) \in \mathbf{N}) \\
& \wedge (\text{cddr}(addr2) = \mathbf{nil}) \\
& \wedge (\text{type}(addr1) = \text{type}(addr2))) \\
\rightarrow & ((addr1 = addr2) \\
& \quad = ((\text{offset}(addr1) = \text{offset}(addr2)) \\
& \quad \quad \wedge (\text{area-name}(addr1) = \text{area-name}(addr2))))
\end{aligned}$$

THEOREM: lr-proper-heapp-nodep-tag-cons

$$\begin{aligned}
& ((\text{untag}(\text{fetch}(addr, data-seg)) = \text{LR-CONS-TAG}) \\
& \quad \wedge (\text{type}(\text{fetch}(\text{add-addr}(addr, \text{LR-REF-COUNT-OFFSET}), data-seg)) = \mathbf{'nat'}) \\
& \quad \wedge \text{lr-proper-heapp-nodep}(addr, data-seg) \\
& \quad \wedge ((\text{offset} = \text{LR-CAR-OFFSET}) \vee (\text{offset} = \text{LR-CDR-OFFSET}))) \\
\rightarrow & \text{lr-good-pointerp}(\text{fetch}(\text{add-addr}(addr, offset), data-seg), data-seg)
\end{aligned}$$

THEOREM: adpp-add-addr-fact-2

$$\begin{aligned}
& (\text{adpp}(\text{untag}(addr1), data-seg) \\
& \quad \wedge \text{adpp}(\text{untag}(\text{add-addr}(addr1, n)), data-seg) \\
& \quad \wedge \text{adpp}(\text{untag}(addr2), data-seg) \\
& \quad \wedge (\neg \text{adpp}(\text{untag}(\text{add-addr}(addr2, n)), data-seg)) \\
& \quad \wedge (\text{area-name}(addr1) = \text{area-name}(addr2))) \\
\rightarrow & (\text{offset}(addr1) < \text{offset}(addr2))
\end{aligned}$$

THEOREM: fetch-lr-nodep-add-addr

$$\begin{aligned}
& ((\neg \text{adpp}(\text{untag}(\text{add-addr}(addr, n)), data-seg)) \wedge \text{lr-nodep}(addr, data-seg)) \\
\rightarrow & (\text{fetch}(\text{add-addr}(addr, n), data-seg) = 0)
\end{aligned}$$

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

THEOREM: untag-addr-addr-tag

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

THEOREM: lr-good-pointerp-lessp-offset-max-heap-node

$$\begin{aligned}
& (\text{adpp}(\text{untag}(addr), data-seg) \\
& \quad \wedge \text{lr-boundary-nodep}(addr) \\
& \quad \wedge \text{listp}(addr) \\
& \quad \wedge (\text{cddr}(addr) = \mathbf{nil}) \\
& \quad \wedge (\text{type}(addr) = \mathbf{'addr'}) \\
& \quad \wedge (\text{area-name}(addr) = \mathbf{'heap'}) \\
& \quad \wedge (\text{type}(\text{fetch}(\text{add-addr}(addr, \text{LR-REF-COUNT-OFFSET}), data-seg)) = \mathbf{'nat'})
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{ adpp}(\text{untag}(\text{lr-max-node}(data-seg)), data-seg) \\
& \wedge \text{ lr-boundary-nodep}(\text{lr-max-node}(data-seg)) \\
\rightarrow & (\text{offset}(addr) \\
& < (\text{length}(\text{cdr}(\text{assoc}(\text{identity}(\text{LR-HEAP-NAME}), data-seg))) - 1))
\end{aligned}$$

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

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

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

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

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

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

EVENT: Disable lr-node-listp-delete.

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

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

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

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

THEOREM: lr-set-error-lr->p
 $\text{lr->p}(\text{lr-set-error}(p, flag)) = \text{lr-set-error}(\text{lr->p}(p), flag)$

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

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

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

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

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

EVENT: Disable strip-cars-restn.

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

EVENT: Disable strip-cars-firstn.

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

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

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

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

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

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

EVENT: Disable put-zero.

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

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

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

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

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

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

```

→ (deposit (value1, addr1, deposit (value2, addr2, data-seg))
   =  if (area-name (addr1) = area-name (addr2))
       ∧ (offset (addr1) = offset (addr2))
   then deposit (value1, addr1, data-seg)
   else deposit (value2,
                 addr2,
                 deposit (value1, addr1, data-seg)) endif)

```

THEOREM: deposit-ref-count-move-outward

```

(offset (addr) ∈ N)
→ (deposit (value1,
            addr,
            deposit (value2,
                    add-addr (addr, identity (LR-REF-COUNT-OFFSET)),
                    data-seg)))
=  deposit (value2,
            add-addr (addr, LR-REF-COUNT-OFFSET),
            deposit (value1, addr, data-seg)))

```

DEFINITION:

```

ihint-2 (flag, s, l, table, c)
=  if s-err-flag (s) ≠ 'run then t
    elseif flag = 'list
    then if s-pos (s) ≃ nil then t
          elseif listp (s-expr-list (s))
          then ihint-2 (t, s, l, table, c)
               ∧ ihint-2 ('list,
                           s-set-expr (s-eval (t, s, c), s, nx (s-pos (s))),
                           lr-eval (t, s->lr1 (s, l, table), c),
                           table,
                           c)
          else t endif
    elseif c ≃ 0 then t
    elseif litatom (s-expr (s)) then t
    elseif s-expr (s) ≃ nil then t
    elseif car (s-expr (s)) = 'if
    then let lrtest be lr-if-ok (lr-eval (t,
                                           s->lr1 (s-set-pos (s,
                                                         dv (s-pos (s),
                                                         1)),
                                           l,
                                           table),
                                           c)),
          stest be s-eval (t, s-set-pos (s, dv (s-pos (s), 1)), c)

```

```

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)

```


$$\begin{aligned}
& \wedge \text{ (offset (addr) = offset (free-addr))} \\
\rightarrow & \text{ (type (fetch (add-addr (free-addr, identity (LR-REF-COUNT-OFFSET)),} \\
& \quad \text{data-seg))} \\
& = \text{ 'nat)}
\end{aligned}$$

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

$$\begin{aligned}
& ((\text{type (addr1) = 'addr}) \\
& \wedge \text{ (caddr (addr1) = nil)} \\
& \wedge \text{ listp (addr1)} \\
& \wedge \text{ listp (untag (addr1))} \\
& \wedge \text{ (offset (addr1) \in \mathbf{N})} \\
& \wedge \text{ lr-boundary-nodep (addr1)} \\
& \wedge \text{ (type (addr2) = 'addr)} \\
& \wedge \text{ (caddr (addr2) = nil)} \\
& \wedge \text{ listp (addr2)} \\
& \wedge \text{ listp (untag (addr2))} \\
& \wedge \text{ (offset (addr2) \in \mathbf{N})} \\
& \wedge \text{ lr-boundary-nodep (addr2)} \\
& \wedge \text{ (area-name (addr2) = area-name (addr1))} \\
& \wedge \text{ (m < LR-NODE-SIZE)}) \\
\rightarrow & ((\text{offset (addr1) = (m + offset (addr2))}) \\
& = \text{ ((m \simeq 0) \wedge (addr1 = addr2))})
\end{aligned}$$

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

$$\begin{aligned}
& ((\text{type (addr1) = 'addr}) \\
& \wedge \text{ (caddr (addr1) = nil)} \\
& \wedge \text{ listp (addr1)} \\
& \wedge \text{ listp (untag (addr1))} \\
& \wedge \text{ (offset (addr1) \in \mathbf{N})} \\
& \wedge \text{ lr-boundary-nodep (addr1)} \\
& \wedge \text{ (type (addr2) = 'addr)} \\
& \wedge \text{ (caddr (addr2) = nil)} \\
& \wedge \text{ listp (addr2)} \\
& \wedge \text{ listp (untag (addr2))} \\
& \wedge \text{ (offset (addr2) \in \mathbf{N})} \\
& \wedge \text{ lr-boundary-nodep (addr2)} \\
& \wedge \text{ (n < LR-NODE-SIZE)} \\
& \wedge \text{ (m < LR-NODE-SIZE)} \\
& \wedge \text{ (area-name (addr1) = area-name (addr2))}) \\
\rightarrow & (((\text{n + offset (addr1) = (m + offset (addr2))}) \\
& = \text{ ((fix (n) = fix (m)) \wedge (addr1 = addr2))})
\end{aligned}$$

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

$$\begin{aligned}
& ((\text{type (fetch (add-addr (free-addr, LR-REF-COUNT-OFFSET), data-seg))} \neq \text{'nat}) \\
& \wedge \text{ lr-nodep (free-addr, data-seg)})
\end{aligned}$$

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

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
 $((\text{type}(\text{fetch}(\text{add-addr}(\text{free-addr}, \text{LR-REF-COUNT-OFFSET}), data-seg)) \neq \text{'nat}))$
 $\wedge \quad \text{lr-noddep}(\text{free-addr}, data-seg)$
 $\wedge \quad \text{lr-valp}(value, addr, data-seg))$
 $\rightarrow \quad \text{lr-valp}(value, addr, \text{deposit}(\text{anything}, \text{free-addr}, data-seg))$

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
 $(\text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l))))$
 $\wedge \quad \text{lr-programs-properp}(l, table)$
 $\wedge \quad \text{lr-programs-properp}(new-l, table)$
 $\wedge \quad \text{listp}(\text{lr-expr}(l))$
 $\wedge \quad \text{proper-p-statep}(\text{lr->p}(new-l))$
 $\wedge \quad \text{lr-proper-free-listp}(\text{p-data-segment}(new-l))$
 $\wedge \quad \text{adpp}(\text{untag}(\text{lr-max-node}(\text{p-data-segment}(new-l))), \text{p-data-segment}(new-l))$
 $\wedge \quad \text{lr-boundary-noddep}(\text{lr-max-node}(\text{p-data-segment}(new-l)))$
 $\wedge \quad (\text{p-psw}(new-l) = \text{'run})$
 $\wedge \quad (\text{p-psw}(\text{p-run-subr}(\text{car}(\text{lr-expr}(l)),$
 $\qquad \qquad \qquad \text{p-set-pc}(\text{lr->p}(new-l), \text{lr-return-pc}(l))))$
 $\qquad \qquad \qquad = \text{'run})$
 $\wedge \quad \text{lr-valp}(value, addr, \text{p-data-segment}(new-l))$
 $\wedge \quad (\text{p-prog-segment}(l) = \text{p-prog-segment}(new-l)))$
 $\rightarrow \quad \text{lr-valp}(value,$
 $\qquad \qquad \qquad addr,$
 $\qquad \qquad \qquad \text{p-data-segment}(\text{p-run-subr}(\text{car}(\text{lr-expr}(l)),$
 $\qquad \qquad \qquad \qquad \qquad \qquad \text{p-set-pc}(\text{lr->p}(new-l),$
 $\qquad \qquad \qquad \qquad \qquad \qquad \qquad \text{lr-return-pc}(l))))$

THEOREM: numberp-offset-sub-addr
 $\text{offset}(\text{sub-addr}(addr, n)) \in \mathbf{N}$

THEOREM: lr-free-list-nodes-deposit-non-ref-count
 $(\text{lr-nodep}(addr, data-seg)$
 $\wedge \text{lr-nodep}(max-addr, data-seg)$
 $\wedge (\text{offset} \neq \text{LR-REF-COUNT-OFFSET})$
 $\wedge (\text{offset} \in \mathbf{N})$
 $\wedge (\text{offset} < \text{LR-NODE-SIZE}))$
 $\rightarrow (\text{lr-free-list-nodes}(max-addr,$
 $\text{deposit}(\text{anything}, \text{add-addr}(addr, \text{offset}), data-seg))$
 $= \text{lr-free-list-nodes}(max-addr, data-seg))$

THEOREM: lr-nodep-member-lr-node-listp-adpp-untag-listp
 $(\text{lr-node-listp}(list, data-seg) \wedge (node \in list)) \rightarrow \text{listp}(\text{untag}(node))$

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

THEOREM: lr-nodep-member-lr-node-listp-adpp-untag-numberp-offset
 $(\text{lr-node-listp}(list, data-seg) \wedge (node \in list)) \rightarrow (\text{offset}(node) \in \mathbf{N})$

THEOREM: lr-nodep-member-lr-node-listp-lr-boundaryp-offsetp
 $(\text{lr-node-listp}(list, data-seg) \wedge (node \in list))$
 $\rightarrow \text{lr-boundaryp-offsetp}(\text{offset}(node))$

THEOREM: lr-check-free-nodes-deposit-non-ref-count
 $(\text{lr-nodep}(addr2, data-seg)$
 $\wedge \text{lr-nodep}(max-addr, data-seg)$
 $\wedge (\text{offset} \neq \text{LR-REF-COUNT-OFFSET})$
 $\wedge (\text{offset} \in \mathbf{N})$
 $\wedge (\text{offset} < \text{LR-NODE-SIZE})$
 $\wedge \text{lr-node-listp}(node-list, data-seg))$
 $\rightarrow (\text{lr-check-free-nodes}(addr1,$
 $node-list,$
 $\text{deposit}(\text{anything}, \text{add-addr}(addr2, \text{offset}), data-seg),$
 $max-addr)$
 $= \text{lr-check-free-nodes}(addr1, node-list, data-seg, max-addr))$

EVENT: Disable lr-nodep-member-lr-node-listp-adpp-untag-numberp-offset.

THEOREM: adpp-deposit-other-area
 $(\text{adp-name}(adp) \neq \text{area-name}(addr))$
 $\rightarrow (\text{adpp}(adp, \text{deposit}(\text{anything}, addr, data-seg)) = \text{adpp}(adp, data-seg))$

EVENT: Disable adpp-deposit-other-area.

THEOREM: length-deposit

$$\begin{aligned} & \text{length}(\text{cdr}(\text{assoc}(\text{name}, \text{deposit}(\text{anything}, \text{addr}, \text{data-seg})))) \\ = & \text{if definedp}(\text{area-name}(\text{addr}), \text{data-seg}) \\ & \text{then if area-name}(\text{addr}) = \text{name} \\ & \quad \text{then if offset}(\text{addr}) < \text{length}(\text{cdr}(\text{assoc}(\text{name}, \text{data-seg}))) \\ & \quad \quad \text{then length}(\text{cdr}(\text{assoc}(\text{name}, \text{data-seg}))) \\ & \quad \quad \text{else } 1 + \text{offset}(\text{addr}) \text{ endif} \\ & \quad \text{else length}(\text{cdr}(\text{assoc}(\text{name}, \text{data-seg}))) \text{ endif} \\ & \text{else length}(\text{cdr}(\text{assoc}(\text{name}, \text{data-seg}))) \text{ endif} \end{aligned}$$

THEOREM: same-signature-deposit

$$\begin{aligned} & (\text{adpp}(\text{untag}(\text{addr}), \text{segment2}) \wedge \text{lr-proper-p-areasp}(\text{segment2})) \\ \rightarrow & (\text{same-signature}(\text{segment1}, \text{deposit}(\text{anything}, \text{addr}, \text{segment2}))) \\ = & \text{same-signature}(\text{segment1}, \text{segment2}) \end{aligned}$$

THEOREM: lr-max-node-same-signature

$$\begin{aligned} & \text{same-signature}(\text{data-seg1}, \text{data-seg2}) \\ \rightarrow & (\text{lr-max-node}(\text{data-seg2}) = \text{lr-max-node}(\text{data-seg1})) \end{aligned}$$

EVENT: Disable lr-max-node-same-signature.

THEOREM: lr-max-node-deposit

$$\begin{aligned} & (\text{adpp}(\text{untag}(\text{addr}), \text{data-seg}) \wedge \text{lr-proper-p-areasp}(\text{data-seg})) \\ \rightarrow & (\text{lr-max-node}(\text{deposit}(\text{anything}, \text{addr}, \text{data-seg}))) \\ = & \text{lr-max-node}(\text{data-seg}) \end{aligned}$$

THEOREM: not-adpp-untag-node-not-definedp-lr-heap-name

$$\begin{aligned} & (\neg \text{definedp}(\text{area-name}(\text{addr}), \text{data-seg})) \\ \rightarrow & (\neg \text{adpp}(\text{untag}(\text{addr}), \text{data-seg})) \end{aligned}$$

THEOREM: sub-addr-area-name-offset-same

$$\begin{aligned} & (\text{listp}(\text{untag}(\text{addr1}))) \\ & \wedge (\text{offset}(\text{addr1}) \in \mathbf{N}) \\ & \wedge (\text{caddr}(\text{addr1}) = \mathbf{nil}) \\ & \wedge (\text{type}(\text{addr1}) = \text{type}(\text{addr2})) \\ \rightarrow & ((\text{addr1} = \text{sub-addr}(\text{addr2}, n)) \\ & = ((\text{offset}(\text{addr1}) = (\text{offset}(\text{addr2}) - n)) \\ & \wedge (\text{area-name}(\text{addr1}) = \text{area-name}(\text{addr2})))) \end{aligned}$$

THEOREM: lr-free-list-nodes-member-greater-offset

$$\begin{aligned} & (\text{offset}(\text{addr}) \not\leq \text{offset}(\text{max-addr})) \\ \rightarrow & (\text{addr} \notin \text{lr-free-list-nodes}(\text{max-addr}, \text{data-seg})) \end{aligned}$$

THEOREM: lr-free-list-nodes-deposit-lr-ref-count-offset

$$((\text{type}(\text{addr}) = \text{'addr}))$$

$$\begin{aligned}
& \wedge (\text{caddr}(\text{addr}) = \mathbf{nil}) \\
& \wedge \text{adpp}(\text{untag}(\text{addr}), \text{data-seg}) \\
& \wedge \text{lr-boundary-nodep}(\text{addr}) \\
& \wedge (\text{area-name}(\text{addr}) = \mathbf{'heap}) \\
& \wedge (\text{type}(\text{max-addr}) = \mathbf{'addr}) \\
& \wedge \text{adpp}(\text{untag}(\text{max-addr}), \text{data-seg}) \\
& \wedge \text{lr-boundary-nodep}(\text{max-addr}) \\
& \wedge (\text{area-name}(\text{max-addr}) = \mathbf{'heap}) \\
& \wedge (\text{type}(\text{ref-count}) = \mathbf{'nat}) \\
& \wedge (\text{untag}(\text{ref-count}) \in \mathbf{N}) \\
& \rightarrow (\text{lr-free-list-nodes}(\text{max-addr}, \\
& \quad \text{deposit}(\text{ref-count}, \\
& \quad \quad \text{add-addr}(\text{addr}, \\
& \quad \quad \quad \text{identity}(\text{LR-REF-COUNT-OFFSET})), \\
& \quad \quad \text{data-seg})) \\
& \quad = \text{delete}(\text{addr}, \text{lr-free-list-nodes}(\text{max-addr}, \text{data-seg}))
\end{aligned}$$

EVENT: Disable lr-free-list-nodes-member-greater-offset.

EVENT: Disable not-adpp-untag-node-not-definedp-lr-heap-name.

DEFINITION:

$$\begin{aligned}
& \text{no-duplicatesp}(\text{list}) \\
& = \text{if listp}(\text{list}) \\
& \quad \text{then if car}(\text{list}) \in \text{cdr}(\text{list}) \text{ then f} \\
& \quad \quad \text{else no-duplicatesp}(\text{cdr}(\text{list})) \text{ endif} \\
& \quad \text{else t endif}
\end{aligned}$$

THEOREM: not-member-occurences-0
 $(x \notin z) \rightarrow (\text{occurences}(x, z) = 0)$

EVENT: Disable not-member-occurences-0.

THEOREM: no-duplicatesp-occurences-1
 $(\text{no-duplicatesp}(\text{list}) \wedge (e \in \text{list})) \rightarrow (\text{occurences}(e, \text{list}) = 1)$

THEOREM: no-duplicatesp-lr-free-list-nodes
 $\text{no-duplicatesp}(\text{lr-free-list-nodes}(\text{addr}, \text{data-seg}))$

THEOREM: member-area-name-offset-same
 $((\text{addr1} \in \text{node-list})$
 $\wedge (\text{offset}(\text{addr1}) \in \mathbf{N})$
 $\wedge (\text{caddr}(\text{addr1}) = \mathbf{nil})$

$$\begin{aligned}
& \wedge \text{listp}(\text{untag}(\text{addr1})) \\
& \wedge \text{listp}(\text{untag}(\text{addr2})) \\
& \wedge (\text{offset}(\text{addr2}) \in \mathbf{N}) \\
& \wedge (\text{caddr}(\text{addr2}) = \mathbf{nil}) \\
& \wedge (\text{type}(\text{addr1}) = \text{type}(\text{addr2})) \\
& \wedge (\text{area-name}(\text{addr1}) = \text{area-name}(\text{addr2})) \\
& \wedge (\text{offset}(\text{addr2}) = \text{offset}(\text{addr1})) \\
& \rightarrow (\text{addr2} \in \text{node-list})
\end{aligned}$$

THEOREM: lr-check-free-nodes-delete-deposit

$$\begin{aligned}
& (\text{lr-check-free-nodes}(\text{addr2}, \text{node-list}, \text{data-seg}, \text{max-addr}) \\
& \wedge (\text{addr1} \notin \text{node-list}) \\
& \wedge \text{lr-nodep}(\text{addr1}, \text{data-seg}) \\
& \wedge \text{lr-node-listp}(\text{node-list}, \text{data-seg}) \\
& \wedge (\text{type}(\text{ref-count}) = \mathbf{'nat'}) \\
& \wedge (\text{untag}(\text{ref-count}) \in \mathbf{N})) \\
& \rightarrow \text{lr-check-free-nodes}(\text{addr2}, \\
& \quad \text{node-list}, \\
& \quad \text{deposit}(\text{ref-count}, \\
& \quad \quad \text{add-addr}(\text{addr1}, \\
& \quad \quad \quad \text{identity}(\text{LR-REF-COUNT-OFFSET})), \\
& \quad \quad \text{data-seg}), \\
& \quad \text{max-addr})
\end{aligned}$$

EVENT: Disable lr-check-free-nodes-delete-deposit.

EVENT: Disable member-area-name-offset-same.

THEOREM: lr-check-free-nodes-deposit-free-ptr

$$\begin{aligned}
& (\text{adpp}(\text{identity}(\text{untag}(\text{LR-FP-ADDR})), \text{data-seg}) \\
& \wedge \text{lr-node-listp}(\text{node-list}, \text{data-seg})) \\
& \rightarrow (\text{lr-check-free-nodes}(\text{addr}, \\
& \quad \text{node-list}, \\
& \quad \text{deposit}(\text{anything}, \text{identity}(\text{LR-FP-ADDR}), \text{data-seg}), \\
& \quad \text{max-addr}) \\
& = \text{lr-check-free-nodes}(\text{addr}, \text{node-list}, \text{data-seg}, \text{max-addr}))
\end{aligned}$$

THEOREM: lr-free-list-nodes-deposit-free-ptr

$$\begin{aligned}
& (\text{lr-nodep}(\text{max-addr}, \text{data-seg}) \wedge \text{adpp}(\text{identity}(\text{untag}(\text{LR-FP-ADDR})), \text{data-seg})) \\
& \rightarrow (\text{lr-free-list-nodes}(\text{max-addr}, \\
& \quad \text{deposit}(\text{anything}, \text{identity}(\text{LR-FP-ADDR}), \text{data-seg})) \\
& = \text{lr-free-list-nodes}(\text{max-addr}, \text{data-seg}))
\end{aligned}$$

THEOREM: deposit-ref-count-move-inward-2

```

(lr-nodep (addr, data-seg)
  ∧ (offset ≠ LR-REF-COUNT-OFFSET)
  ∧ (offset ≠ 0)
  ∧ (offset < LR-NODE-SIZE))
→ (deposit (any1,
            add-addr (addr, identity (LR-REF-COUNT-OFFSET)),
            deposit (any2, add-addr (addr, offset), data-seg))
    = deposit (any2,
            add-addr (addr, offset),
            deposit (any1,
                    add-addr (addr, identity (LR-REF-COUNT-OFFSET)),
                    data-seg)))

```

EVENT: Disable deposit-ref-count-move-inward-2.

THEOREM: lr-free-list-nodes-deposit-lr-nodep

```

(lr-nodep (addr, data-seg) ∧ lr-nodep (max-addr, data-seg))
→ (lr-free-list-nodes (max-addr, deposit (anything, addr, data-seg))
    = lr-free-list-nodes (max-addr, data-seg))

```

THEOREM: lr-check-free-nodes-deposit-lr-nodep

```

(lr-nodep (addr2, data-seg)
  ∧ lr-nodep (max-addr, data-seg)
  ∧ lr-node-listp (node-list, data-seg))
→ (lr-check-free-nodes (addr1,
                        node-list,
                        deposit (anything, addr2, data-seg),
                        max-addr)
    = lr-check-free-nodes (addr1, node-list, data-seg, max-addr))

```

THEOREM: same-signature-cons

```

same-signature (data-seg1, cons (x, data-seg2))
= if listp (data-seg1)
  then (signature (car (data-seg1)) = signature (x))
      ∧ same-signature (cdr (data-seg1), data-seg2)
  else f endif

```

THEOREM: same-signature-nil

```

(data-seg1 ≃ nil)
→ (same-signature (data-seg1, data-seg2) = (data-seg2 ≃ nil))

```

THEOREM: listp-put-assoc

```

listp (put-assoc (val, name, alist)) = listp (alist)

```

THEOREM: not-same-signature-deposit-too-large-addr

$$\begin{aligned} & (\text{definedp}(\text{area-name}(addr), data-seg2) \\ & \wedge \text{lr-proper-p-areasp}(data-seg2) \\ & \wedge (\text{offset}(addr) \not\leq \text{length}(\text{value}(\text{area-name}(addr), data-seg1)))) \\ \rightarrow & (\neg \text{same-signature}(data-seg1, \text{deposit}(any, addr, data-seg2))) \end{aligned}$$

EVENT: Disable same-signature-cons.

EVENT: Disable same-signature-nil.

THEOREM: adpp-deposit-a-list

$$\text{adpp}(adp, data-seg) \rightarrow \text{adpp}(adp, \text{deposit-a-list}(list, addr2, data-seg))$$

THEOREM: lr-proper-p-areasp-deposit-a-list

$$\begin{aligned} & \text{lr-proper-p-areasp}(data-seg) \\ \rightarrow & \text{lr-proper-p-areasp}(\text{deposit-a-list}(list, addr, data-seg)) \end{aligned}$$

THEOREM: definedp-deposit-a-list

$$\text{definedp}(tag, \text{deposit-a-list}(list, addr, data-seg)) = \text{definedp}(tag, data-seg)$$

THEOREM: sub1-plus-not-zero-p-fact-1

$$(x \neq 0) \rightarrow (((y + x) - 1) < y) = \mathbf{f}$$

THEOREM: not-adpp-untag-add-addr-adpp-untag

$$\begin{aligned} & \text{adpp}(\text{untag}(addr), data-seg) \\ \rightarrow & (\text{adpp}(\text{untag}(\text{add-addr}(addr, n)), data-seg) \\ & = ((\text{offset}(addr) + n) \\ & < \text{length}(\text{cdr}(\text{assoc}(\text{area-name}(addr), data-seg))))) \end{aligned}$$

THEOREM: not-same-signature-deposit-a-list-too-large-addr

$$\begin{aligned} & (\text{definedp}(\text{area-name}(addr), data-seg2) \\ & \wedge \text{lr-proper-p-areasp}(data-seg2) \\ & \wedge (\text{offset}(addr) \not\leq \text{length}(\text{value}(\text{area-name}(addr), data-seg1)))) \\ \rightarrow & (\text{same-signature}(data-seg1, \text{deposit-a-list}(list, addr, data-seg2)) \\ & = \text{if listp}(list) \text{ then } \mathbf{f} \\ & \text{ else same-signature}(data-seg1, data-seg2) \text{ endif}) \end{aligned}$$

THEOREM: same-signature-deposit-a-list

$$\begin{aligned} & (\text{adpp}(\text{untag}(addr), data-seg2) \\ & \wedge \text{lr-proper-p-areasp}(data-seg2) \\ & \wedge \text{same-signature}(data-seg1, data-seg2)) \\ \rightarrow & (\text{same-signature}(data-seg1, \text{deposit-a-list}(list, addr, data-seg2)) \\ & = ((\text{offset}(addr) + (\text{length}(list) - 1)) \\ & < \text{length}(\text{cdr}(\text{assoc}(\text{area-name}(addr), data-seg1))))) \end{aligned}$$

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-pospl (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

```
(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 (adp, p-data-segment (lr-eval (flag, l, c)))
   = adpp (adp, p-data-segment (l)))
```

THEOREM: lr-eval-preserves-length-assoc-data-segment

```
(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))
→ (length (cdr (assoc (name, p-data-segment (lr-eval (flag, l, c)))))
   = length (cdr (assoc (name, p-data-segment (l)))))
```

THEOREM: lr-eval-preserves-proper-p-statep-lr->p-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))
→ proper-p-statep (lr->p (lr-eval (flag, lr-set-pos (l, pos), c)))
```

THEOREM: lr-eval-preserves-strip-cars-bindings-car-p-ctrl-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))
→ (strip-cars (bindings (car (p-ctrl-stk (lr-eval (t, lr-set-pos (l, pos), c)))))
   = strip-cars (bindings (car (p-ctrl-stk (l)))))
```

THEOREM: lr-eval-preserves-cdr-p-ctrl-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))
→ (cdr (p-ctrl-stk (lr-eval (t, lr-set-pos (l, pos), c)))
   = cdr (p-ctrl-stk (l)))
```

THEOREM: lr-eval-preserves-adpp-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 (adp, p-data-segment (lr-eval (flag, lr-set-pos (l, pos), c)))
   = adpp (adp, p-data-segment (l)))
```

THEOREM: lr-eval-preserves-lr-max-node-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))
 $\rightarrow$ (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))
 $\wedge$ 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)
 $\wedge$ 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-proper-free-listp (p-data-segment (l))
 $\rightarrow$ 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)))
 $\wedge$ 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-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 (lr-apply-subr (*l*, *new-l*)) = 'run)
 $\wedge$ lr-valp (*value*, *addr*, p-data-segment (*new-l*))
 $\rightarrow$ 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))
 $\wedge$ 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)
 $\wedge$ 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-proper-free-listp (p-data-segment (l))
 $\rightarrow$ lr-proper-free-listp (p-data-segment (lr-eval (*flag*, lr-set-pos (*l*, *pos*), *c*)))

THEOREM: lr-eval-preserves-lr-valp
 $(\text{proper-p-statep } (\text{lr} \rightarrow \text{p } (l)))$
 $\wedge \text{good-posp } (flag, \text{offset } (\text{p-pc } (l)), \text{program-body } (\text{p-current-program } (l)))$
 $\wedge \text{lr-programs-properp } (l, table)$
 $\wedge (\text{p-psw } (\text{lr-eval } (flag, l, c)) = \text{'run})$
 $\wedge \text{lr-proper-free-listp } (\text{p-data-segment } (l))$
 $\wedge \text{adpp } (\text{untag } (\text{lr-max-node } (\text{p-data-segment } (l))), \text{p-data-segment } (l))$
 $\wedge \text{lr-boundary-nodep } (\text{lr-max-node } (\text{p-data-segment } (l)))$
 $\wedge \text{lr-valp } (value, addr, \text{p-data-segment } (l))$
 $\rightarrow \text{lr-valp } (value, addr, \text{p-data-segment } (\text{lr-eval } (flag, l, c)))$

THEOREM: lr-check-f-addrp-deposit-anything-anywhere
 $\text{lr-check-f-addrp } (addr, \text{deposit } (anything, anywhere, data-seg))$
 $= \text{lr-check-f-addrp } (addr, data-seg)$

THEOREM: lr-check-undef-addrp-deposit-anything-anywhere
 $\text{lr-check-undef-addrp } (addr, \text{deposit } (anything, anywhere, data-seg))$
 $= \text{lr-check-undef-addrp } (addr, data-seg)$

THEOREM: lr-check-listp-addrp-deposit-free-ptr-0
 $\text{lr-check-listp-addrp } (addr, \text{deposit } (any, \text{identity } (\text{LR-FP-ADDR}), data-seg))$
 $= \text{lr-check-listp-addrp } (addr, data-seg)$

THEOREM: lr-check-numberp-addrp-deposit-free-ptr-0
 $\text{lr-check-numberp-addrp } (addr, \text{deposit } (any, \text{identity } (\text{LR-FP-ADDR}), data-seg))$
 $= \text{lr-check-numberp-addrp } (addr, data-seg)$

THEOREM: lr-proper-heapp-nodep-deposit-free-ptr-0
 $\text{lr-proper-heapp-nodep } (addr, \text{deposit } (any, \text{identity } (\text{LR-FP-ADDR}), data-seg))$
 $= \text{lr-proper-heapp-nodep } (addr, data-seg)$

THEOREM: lr-proper-heapp2-deposit-free-ptr-0
 $\text{lr-proper-heapp2 } (addr, \text{deposit } (any, \text{identity } (\text{LR-FP-ADDR}), data-seg))$
 $= \text{lr-proper-heapp2 } (addr, data-seg)$

THEOREM: lr-boundary-offsetp-equal-plus-fact-zero
 $(\text{lr-boundary-offsetp } (offset1))$
 $\wedge \text{lr-boundary-offsetp } (offset2)$
 $\wedge (n < \text{LR-NODE-SIZE})$
 $\wedge (offset1 \in \mathbf{N})$
 $\wedge (offset2 \in \mathbf{N})$
 $\rightarrow (((n + offset1) = offset2)$
 $= ((n \simeq 0) \wedge (offset1 = offset2)))$

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)
  ∧ (caddr (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)
  ∧ (caddr (addr) = nil)
  ∧ listp (addr)
  ∧ adpp (untag (addr), data-seg)

```

$$\begin{aligned}
& \wedge \text{lr-boundary-nodep}(\text{addr}) \\
& \wedge (\text{area-name}(\text{addr}) = \text{LR-HEAP-NAME}) \\
& \wedge \text{lr-check-numberp-addrp}(\text{max-addr}, \text{data-seg}) \\
& \wedge (\text{type}(\text{tag}) = \text{'nat}) \\
& \wedge (\text{untag}(\text{tag}) \in \mathbf{N}) \\
\rightarrow & \text{lr-check-numberp-addrp}(\text{max-addr}, \\
& \quad \text{deposit-a-list}(\text{list}(\text{x0}, \text{tag}, \text{x2}, \text{x3}), \\
& \quad \quad \text{addr}, \\
& \quad \quad \text{data-seg}))
\end{aligned}$$

THEOREM: lr-check-listp-addrp-deposit-a-list-cons

$$\begin{aligned}
& (\text{lr-good-pointerp}(\text{good-pointer1}, \text{data-seg}) \\
& \wedge \text{lr-good-pointerp}(\text{good-pointer2}, \text{data-seg}) \\
& \wedge \text{adpp}(\text{untag}(\text{addr}), \text{data-seg}) \\
& \wedge \text{lr-boundary-nodep}(\text{addr}) \\
& \wedge \text{adpp}(\text{untag}(\text{max-addr}), \text{data-seg}) \\
& \wedge \text{lr-boundary-nodep}(\text{max-addr}) \\
& \wedge (\text{area-name}(\text{addr}) = \text{LR-HEAP-NAME}) \\
& \wedge (\text{area-name}(\text{max-addr}) = \text{LR-HEAP-NAME}) \\
& \wedge (\text{type}(\text{tag}) = \text{'nat}) \\
& \wedge (\text{untag}(\text{tag}) \in \mathbf{N}) \\
& \wedge (\text{offset}(\text{addr}) = \text{offset}(\text{max-addr}))) \\
\rightarrow & \text{lr-check-listp-addrp}(\text{max-addr}, \\
& \quad \text{deposit-a-list}(\text{list}(\text{identity}(\text{tag}(\text{'nat}, \\
& \quad \quad \text{LR-CONS-TAG})), \\
& \quad \quad \text{tag}, \\
& \quad \quad \text{good-pointer1}, \\
& \quad \quad \text{good-pointer2}), \\
& \quad \text{addr}, \\
& \quad \text{data-seg}))
\end{aligned}$$

THEOREM: lr-check-listp-addrp-deposit-a-list-other-place

$$\begin{aligned}
& (\text{adpp}(\text{untag}(\text{addr}), \text{data-seg}) \\
& \wedge \text{lr-boundary-nodep}(\text{addr}) \\
& \wedge \text{adpp}(\text{untag}(\text{max-addr}), \text{data-seg}) \\
& \wedge \text{lr-boundary-nodep}(\text{max-addr}) \\
& \wedge (\text{area-name}(\text{addr}) = \text{LR-HEAP-NAME}) \\
& \wedge (\text{area-name}(\text{max-addr}) = \text{LR-HEAP-NAME}) \\
& \wedge \text{lr-check-listp-addrp}(\text{max-addr}, \text{data-seg}) \\
& \wedge (\text{offset}(\text{addr}) \neq \text{offset}(\text{max-addr})) \\
& \wedge (\text{type}(\text{ref-count}) = \text{'nat}) \\
\rightarrow & \text{lr-check-listp-addrp}(\text{max-addr}, \\
& \quad \text{deposit-a-list}(\text{list}(\text{x0}, \text{ref-count}, \text{x2}, \text{x3}), \\
& \quad \quad \text{addr}, \\
& \quad \quad \text{data-seg}))
\end{aligned}$$

THEOREM: lr-proper-heapp-nodep-deposit-a-list-cons

$$\begin{aligned}
& (\text{lr-nodep } (max\text{-}addr, data\text{-}seg) \\
& \wedge \text{lr-nodep } (addr, data\text{-}seg) \\
& \wedge \text{lr-proper-heapp-nodep } (max\text{-}addr, data\text{-}seg) \\
& \wedge (\text{type } (\text{fetch } (\text{add-addr } (addr, \text{LR-REF-COUNT-OFFSET}), data\text{-}seg)) \neq \text{'nat}) \\
& \wedge \text{lr-good-pointerp } (good\text{-}pointer1, data\text{-}seg) \\
& \wedge \text{lr-good-pointerp } (good\text{-}pointer2, data\text{-}seg) \\
& \wedge (\text{type } (tag) = \text{'nat}) \\
& \wedge (\text{untag } (tag) \in \mathbf{N})) \\
\rightarrow & \text{lr-proper-heapp-nodep } (max\text{-}addr, \\
& \quad \text{deposit-a-list } (\text{list } (\text{identity } (\text{tag } (\text{'nat}, \\
& \quad \quad \quad \text{LR-CONS-TAG})), \\
& \quad \quad \quad tag, \\
& \quad \quad \quad good\text{-}pointer1, \\
& \quad \quad \quad good\text{-}pointer2), \\
& \quad addr, \\
& \quad data\text{-}seg))
\end{aligned}$$

THEOREM: lr-proper-heapp2-deposit-a-list-cons

$$\begin{aligned}
& (\text{lr-nodep } (max\text{-}addr, data\text{-}seg) \\
& \wedge \text{lr-nodep } (addr, data\text{-}seg) \\
& \wedge \text{lr-proper-heapp2 } (max\text{-}addr, data\text{-}seg) \\
& \wedge (\text{type } (\text{fetch } (\text{add-addr } (addr, \text{LR-REF-COUNT-OFFSET}), data\text{-}seg)) \neq \text{'nat}) \\
& \wedge \text{lr-good-pointerp } (good\text{-}pointer1, data\text{-}seg) \\
& \wedge \text{lr-good-pointerp } (good\text{-}pointer2, data\text{-}seg) \\
& \wedge (\text{type } (tag) = \text{'nat}) \\
& \wedge (\text{untag } (tag) \in \mathbf{N})) \\
\rightarrow & \text{lr-proper-heapp2 } (max\text{-}addr, \\
& \quad \text{deposit-a-list } (\text{list } (\text{identity } (\text{tag } (\text{'nat}, \text{LR-CONS-TAG})), \\
& \quad \quad \quad tag, \\
& \quad \quad \quad good\text{-}pointer1, \\
& \quad \quad \quad good\text{-}pointer2), \\
& \quad addr, \\
& \quad data\text{-}seg))
\end{aligned}$$

THEOREM: not-psw-run-lr-eval

$$(\text{p-psw } (l) \neq \text{'run}) \rightarrow (\text{lr-eval } (flag, l, c) = l)$$

THEOREM: program-body-assoc-lr-compile-programs

$$\begin{aligned}
& \text{program-body } (\text{assoc } (name, \text{lr-compile-programs } (progs, table))) \\
= & \text{lr-compile-body } (t, \\
& \quad \text{s-body } (\text{assoc } (name, progs)), \\
& \quad \text{lr-make-temp-name-alist } (\text{s-temp-list } (\text{assoc } (name, progs)), \\
& \quad \quad \quad \text{s-formals } (\text{assoc } (name, progs))), \\
& \quad table)
\end{aligned}$$

THEOREM: listp-lr-compile-body
 $\text{listp}(\text{lr-compile-body}(flag, body, temp\text{-}name\text{-}alist, table)) = \text{listp}(body)$

THEOREM: car-lr-compile-body
 $(flag \neq \text{'list})$
 $\rightarrow (\text{car}(\text{lr-compile-body}(flag, body, temp\text{-}name\text{-}alist, table)) = \text{car}(body))$

THEOREM: good-posp1-expand-list-temps
 $((temp = \text{S-TEMP-EVAL}) \vee (temp = \text{S-TEMP-TEST})) \wedge \text{listp}(pos)$
 $\rightarrow (\text{good-posp1}(pos, \text{list}(temp, body, name))$
 $\quad = ((\text{car}(pos) = 1) \wedge \text{good-posp1}(\text{cdr}(pos), body)))$

THEOREM: length-lr-compile-body-list
 $\text{length}(\text{lr-compile-body}(\text{'list}, body, temp\text{-}name\text{-}alist, table))$
 $= \text{length}(body)$

THEOREM: get-lr-compile-body-list
 $\text{get}(n, \text{lr-compile-body}(\text{'list}, body, temp\text{-}name\text{-}alist, table))$
 $= \text{lr-compile-body}(\mathbf{t}, \text{get}(n, body), temp\text{-}name\text{-}alist, table)$

THEOREM: get-lr-compile-body
 $(\text{listp}(body))$
 $\wedge (\text{car}(body) \neq \text{S-TEMP-FETCH})$
 $\wedge (\text{car}(body) \neq \text{S-TEMP-EVAL})$
 $\wedge (\text{car}(body) \neq \text{S-TEMP-TEST})$
 $\wedge (\text{car}(body) \neq \text{'quote})$
 $\wedge (n \neq 0)$
 $\rightarrow (\text{get}(n, \text{lr-compile-body}(\mathbf{t}, body, temp\text{-}name\text{-}alist, table))$
 $\quad = \text{lr-compile-body}(\mathbf{t}, \text{get}(n, body), temp\text{-}name\text{-}alist, table))$

THEOREM: length-lr-compile-body-t
 $(\text{listp}(body))$
 $\wedge (\text{car}(body) \neq \text{S-TEMP-FETCH})$
 $\wedge (\text{car}(body) \neq \text{S-TEMP-EVAL})$
 $\wedge (\text{car}(body) \neq \text{S-TEMP-TEST})$
 $\wedge (\text{car}(body) \neq \text{'quote})$
 $\rightarrow (\text{length}(\text{lr-compile-body}(\mathbf{t}, body, temp\text{-}name\text{-}alist, table))$
 $\quad = \text{length}(body))$

THEOREM: good-posp1-lr-compile-body
 $\text{good-posp1}(pos, \text{lr-compile-body}(\mathbf{t}, body, temp\text{-}name\text{-}alist, table))$
 $= \text{good-posp1}(pos, body)$

THEOREM: cur-expr-lr-compile-body-t
 $\text{good-posp1}(pos, body)$
 $\rightarrow (\text{cur-expr}(pos, \text{lr-compile-body}(\mathbf{t}, body, temp\text{-}name\text{-}alist, table))$
 $\quad = \text{lr-compile-body}(\mathbf{t}, \text{cur-expr}(pos, body), temp\text{-}name\text{-}alist, table))$

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*) → plistp (*temp-stk*)

THEOREM: proper-p-statep-lr->p-plistp-p-temp-stk
 proper-p-statep (lr->p (*l*)) → plistp (p-temp-stk (*l*))

THEOREM: proper-p-statep-lr->p-not-0-p-temp-stk
 proper-p-statep (lr->p (*l*)) → (p-temp-stk (*l*) ≠ 0)

THEOREM: plistp-lastcdr-nil
 plistp (*list*) → (lastcdr (*list*) = **nil**)

THEOREM: lr-eval-preserves-lr-valp-lr-set-expr
 (proper-p-statep (lr->p (*l*))
 ∧ proper-p-statep (lr->p (lr-set-expr (*l1*, *l*, *pos*)))
 ∧ good-posp (*flag*, *pos*, program-body (p-current-program (*l*)))
 ∧ lr-programs-properp (*l*, *table*)
 ∧ (p-psw (lr-eval (*flag*, lr-set-expr (*l1*, *l*, *pos*), *c*)) = 'run)
 ∧ lr-proper-free-listp (p-data-segment (*l*))
 ∧ lr-proper-free-listp (p-data-segment (*l1*))
 ∧ adpp (untag (lr-max-node (p-data-segment (*l*))), p-data-segment (*l*))
 ∧ lr-boundary-nodep (lr-max-node (p-data-segment (*l*)))
 ∧ adpp (untag (lr-max-node (p-data-segment (*l1*))), p-data-segment (*l1*))
 ∧ lr-boundary-nodep (lr-max-node (p-data-segment (*l1*)))
 ∧ lr-valp (*value*, *addr*, p-data-segment (*l1*))
 ∧ (length (cdr (assoc (LR-HEAP-NAME, p-data-segment (*l1*))))
 = length (cdr (assoc (LR-HEAP-NAME, p-data-segment (*l*))))))
 → 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*))
 ∧ good-posp (*flag*, *pos*, program-body (p-current-program (*l*)))
 ∧ lr-programs-properp (*l*, *table*)
 ∧ (p-psw (lr-eval (*flag*, lr-set-expr (*l1*, *l*, *pos*), *c*)) = 'run)
 ∧ lr-programs-properp (*l1*, *table*)
 ∧ proper-p-statep (lr->p (*l1*))
 ∧ (cdr (p-ctrl-stk (*l1*)) = cdr (p-ctrl-stk (*l*)))
 ∧ (strip-cars (bindings (car (p-ctrl-stk (*l1*))))
 = strip-cars (bindings (car (p-ctrl-stk (*l*))))))

$\wedge$ (p-prog-segment ($l1$) = p-prog-segment (l))
 $\wedge$ (p-word-size ($l1$) = p-word-size (l))
 $\wedge$ (p-max-ctrl-stk-size ($l1$) = p-max-ctrl-stk-size (l))
 $\wedge$ (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 ($\mathbf{t}$, l , c), l , nx(pos)), c)
in
 (good-posp ('list, pos , program-body (p-current-program (l)))
 $\wedge$ lr-programs-properp (l , $table$)
 $\wedge$ proper-p-statep (lr->p (l))
 $\wedge$ listp (lr-expr-list (l))
 $\wedge$ listp (offset (p-pc (l)))
 $\wedge$ lr-proper-heapp (p-data-segment (l))
 $\wedge$ lr-check-result ($\mathbf{t}$,
 $value1$,
 p-temp-stk (lr-eval ($\mathbf{t}$, l , c)),
 p-data-segment (lr-eval ($\mathbf{t}$, l , c)),
 p-temp-stk (l))
 $\wedge$ lr-check-result ('list,
 $value2$,
 p-temp-stk ($l2$),
 p-data-segment ($l2$),
 p-temp-stk (lr-eval ($\mathbf{t}$, l , c)))
 $\wedge$ (p-psw ($l2$) = 'run)
 $\wedge$ (pos = offset (p-pc (l)))
 $\wedge$ ($temp-stk$ = 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 ($\mathbf{t}$, $body$, $temp-name-alist$, $table$)) = 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: litatom-lr-expr-s->lr1
 (good-posp1 (s-pos (s), s-body (s-prog (s))) $\wedge$ litatom (s-expr (s)))
 $\rightarrow$ (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)))
 $\wedge$ (flag $\neq$ 'list)
 $\wedge$ (c $\neq$ 0)
 $\wedge$ litatom (s-expr (s))
 $\wedge$ (s-err-flag (s) = 'run))
 $\rightarrow$ (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)
 $\wedge$ definedp (name, lr-params))
 $\rightarrow$ 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))
 $\wedge$ definedp (area-name (p-pc (l)), p-prog-segment (l)))
 $\rightarrow$ (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))
 $\rightarrow$ ((append (strip-cars (v), w) = append (y, z))
 = ((strip-cars (v) = plist (y)) $\wedge$ (w = z)))

THEOREM: definedp-strip-cars-append-member-x
 (strip-cars (x) = append (y, z))
 $\rightarrow$ ((e $\in$ y) = definedp (e, firstn (length (y), x)))

THEOREM: proper-p-statep-lr->p-member-formals-definedp-bindings
 (proper-p-statep (lr->p (s->lr1 (s, l, table)))
 $\wedge$ definedp (s-pname (s), s-progs (s))
 $\wedge$ ($x \in$ s-formals (assoc (s-pname (s), s-progs (s)))))
 $\rightarrow$ definedp (x,
 firstn (length (s-formals (assoc (s-pname (s), s-progs (s)))),
 bindings (car (p-ctrl-stk (l)))))

THEOREM: lr-valp-addr-0
 $\neg$ lr-valp (addr, 0, data-seg)

THEOREM: lr-valp-cdr-assoc-firstn-cdr-assoc
 lr-valp (addr, cdr (assoc (name, firstn (n, list))), data-seg)
 $\rightarrow$ lr-valp (addr, cdr (assoc (name, list)), data-seg)

THEOREM: lr-s-similar-statesp-lr-s-similar-params-opener
 (lr-s-similar-statesp (s-params, s-temps, l, table)
 $\wedge$ (frame = car (p-ctrl-stk (l)))
 $\wedge$ (data-seg = p-data-segment (l)))
 $\rightarrow$ lr-s-similar-params (s-params, lr-params (frame, l), data-seg)

THEOREM: lr-s-similar-statesp-lr-s-similar-temps-opener
 (lr-s-similar-statesp (s-params, s-temps, l, table)
 $\wedge$ (frame = car (p-ctrl-stk (l)))
 $\wedge$ (data-seg = p-data-segment (l)))
 $\rightarrow$ lr-s-similar-temps (s-temps, lr-temps (frame, l), data-seg)

THEOREM: strip-cars-lr-make-temp-var-dcls
 strip-cars (lr-make-temp-var-dcls (temp-alist)) = strip-cdrs (temp-alist)

THEOREM: lr-check-result-lr-push-tstk
let value **be** cdr (assoc (s-expr (s), bindings (car (p-ctrl-stk (l)))))
in
 (good-posp1 (s-pos (s), s-body (s-prog (s)))
 $\wedge$ definedp (s-pname (s), s-progs (s))
 $\wedge$ (s-err-flag (s) = 'run)
 $\wedge$ litatom (s-expr (s))
 $\wedge$ proper-p-statep (lr->p (s->lr1 (s, l, table)))
 $\wedge$ lr-proper-heapp (p-data-segment (l))
 $\wedge$ lr-programs-properp (s->lr1 (s, l, table), table)
 $\wedge$ (p-psw (lr-push-tstk (s->lr1 (s, l, table), value)) = 'run)
 $\wedge$ lr-s-similar-statesp (s-params (s),
 s-temps (s),
 s->lr1 (s, l, table),
 table))

$\rightarrow$ 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
 $s\text{->lr1}(s\text{-set-pos}(s, pos), l, table) = lr\text{-set-pos}(s\text{->lr1}(s, l, table), pos)$

THEOREM: lr-params-lr-set-pos
 $lr\text{-params}(frame, lr\text{-set-pos}(l, pos)) = lr\text{-params}(frame, l)$

THEOREM: lr-temps-lr-set-pos
 $lr\text{-temps}(frame, lr\text{-set-pos}(l, pos)) = lr\text{-temps}(frame, l)$

THEOREM: lr-s-similar-statesp-lr-s-set-pos
 $lr\text{-s-similar-statesp}(s\text{-params}, s\text{-temps}, lr\text{-set-pos}(l, pos), table)$
 $= lr\text{-s-similar-statesp}(s\text{-params}, s\text{-temps}, l, table)$

THEOREM: lr-set-expr-s->lr1-s-set-expr-lr-pop-tstk
 $s\text{->lr1}(s\text{-set-expr}(s\text{-eval}(\mathbf{t}, s\text{-set-pos}(s, pos), c), s, dv(s\text{-pos}(s), n)),$
 $lr\text{-pop-tstk}(lr\text{-if-ok}(lr\text{-eval}(\mathbf{t}, lr\text{-set-pos}(s\text{->lr1}(s, l, table), pos), c))),$
 $table)$
 $= lr\text{-set-error}(lr\text{-set-expr}(lr\text{-pop-tstk}(lr\text{-if-ok}(lr\text{-eval}(\mathbf{t},$
 $lr\text{-set-pos}(s\text{->lr1}(s,$
 $l,$
 $table),$
 $pos),$
 $c))),$
 $s\text{->lr1}(s, l, table),$
 $dv(s\text{-pos}(s), n)),$
 $s\text{-err-flag}(s\text{-eval}(\mathbf{t}, s\text{-set-pos}(s, pos), c)))$

THEOREM: lr-s-similar-statesp-lr-pop-tstk
 $lr\text{-s-similar-statesp}(s\text{-params}, s\text{-temps}, lr\text{-pop-tstk}(l, table)$
 $= lr\text{-s-similar-statesp}(s\text{-params}, s\text{-temps}, l, table)$

THEOREM: listp-lr-expr-s->lr1
 $good\text{-posp1}(s\text{-pos}(s), s\text{-body}(s\text{-prog}(s)))$
 $\rightarrow (listp(lr\text{-expr}(s\text{->lr1}(s, l, table))) = listp(s\text{-expr}(s)))$

THEOREM: litatom-lr-expr-s->lr1-s-expr
 $good\text{-posp1}(s\text{-pos}(s), s\text{-body}(s\text{-prog}(s)))$
 $\rightarrow (litatom(lr\text{-expr}(s\text{->lr1}(s, l, table))) = litatom(s\text{-expr}(s)))$

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-noddep
 (lr-good-pointerp (addr, data-seg)
 $\wedge$ lr-proper-p-areasp (data-seg)
 $\wedge$ lr-proper-heapp (data-seg))
 $\rightarrow$ lr-proper-heapp-noddep (addr, data-seg)

THEOREM: lr-check-result-f-not-lr-f-addr
 ((car (temp-stk) $\neq$ LR-F-ADDR)
 $\wedge$ lr-proper-p-areasp (data-seg)
 $\wedge$ 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)
 $\wedge$ lr-check-result (t, ans, temp-stk2, data-seg2, cdr (temp-stk1))
 $\wedge$ 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)
 $\wedge$ listp (temp-stk)
 $\wedge$ lr-proper-p-areasp (data-seg)
 $\wedge$ (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))
 $\wedge$ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
 $\wedge$ (flag $\neq$ 'list)
 $\wedge$ lr-programs-properp (l, table)
 $\wedge$ (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)))
 $\wedge$ listp (s-expr (s))
 $\wedge$ (car (s-expr (s)) = 'if)
 $\wedge$ (*flag* $\neq$ 'list)
 $\wedge$ (p-psw (*lr-test*) = 'run)
 $\wedge$ (top (p-temp-stk (*lr-test*)) $\neq$ LR-F-ADDR))
 $\rightarrow$ (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)))
 $\wedge$ listp (s-expr (s))
 $\wedge$ (car (s-expr (s)) = 'if)
 $\wedge$ (*flag* $\neq$ 'list)
 $\wedge$ (p-psw (*lr-test*) = 'run)
 $\wedge$ (top (p-temp-stk (*lr-test*)) = LR-F-ADDR))
 $\rightarrow$ (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)))
 $\wedge$ (*flag* $\neq$ 'list)
 $\wedge$ listp (s-expr (s))
 $\wedge$ (car (s-expr (s)) = 'if)
 $\wedge$ (*c* $\neq$ 0)
 $\wedge$ s-good-statep (s, c)
 $\wedge$ (p-psw (lr-if-ok (lr-eval (t,
 lr-set-pos (s->lr1 (s, l, table), dv (s-pos (s), 1)),
 c)))

$\neq \text{'run}))$
 $\rightarrow (\text{lr-eval}(\text{flag}, \text{s->lr1}(s, l, \text{table}), c)$
 $\quad = \text{lr-if-ok}(\text{lr-eval}(\mathbf{t},$
 $\quad \quad \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), \text{dv}(\text{s-pos}(s), 1)),$
 $\quad \quad c)))$

THEOREM: lr-eval-s->lr1-temp-eval-opener
 $(\text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s)))$
 $\wedge (\text{car}(\text{s-expr}(s)) = \text{S-TEMP-EVAL})$
 $\wedge (\text{flag} \neq \text{'list})$
 $\wedge (c \neq 0)$
 $\wedge \text{listp}(\text{s-expr}(s))$
 $\wedge \text{s-good-statep}(s, c))$
 $\rightarrow (\text{lr-eval}(\text{flag}, \text{s->lr1}(s, l, \text{table}), c)$
 $\quad = \text{lr-set-temp}(\text{lr-eval}(\mathbf{t},$
 $\quad \quad \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}),$
 $\quad \quad \quad \text{dv}(\text{s-pos}(s), 1)),$
 $\quad \quad c),$
 $\quad \text{top}(\text{p-temp-stk}(\text{lr-eval}(\mathbf{t},$
 $\quad \quad \text{lr-set-pos}(\text{s->lr1}(s,$
 $\quad \quad \quad l,$
 $\quad \quad \quad \text{table}),$
 $\quad \quad \quad \text{dv}(\text{s-pos}(s), 1)),$
 $\quad \quad c))),$
 $\quad \text{caddr}(\text{lr-expr}(\text{s->lr1}(s, l, \text{table}))))))$

THEOREM: lr-eval-s->lr1-temp-test-opener
 $(\text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s)))$
 $\wedge (\text{car}(\text{s-expr}(s)) = \text{S-TEMP-TEST})$
 $\wedge (\text{flag} \neq \text{'list})$
 $\wedge (c \neq 0)$
 $\wedge \text{listp}(\text{s-expr}(s))$
 $\wedge \text{s-good-statep}(s, c))$
 $\rightarrow (\text{lr-eval}(\text{flag}, \text{s->lr1}(s, l, \text{table}), c)$
 $\quad = \text{if } \text{p-max-temp-stk-size}(l) \not\leq (2 + \text{length}(\text{p-temp-stk}(l)))$
 $\quad \quad \text{then if } \text{lr-eval-temp-setp}(\text{s->lr1}(s, l, \text{table}))$
 $\quad \quad \quad \text{then } \text{lr-do-temp-fetch}(\text{s->lr1}(s, l, \text{table}))$
 $\quad \quad \quad \text{else } \text{lr-set-temp}(\text{lr-eval}(\mathbf{t},$
 $\quad \quad \quad \text{lr-set-pos}(\text{s->lr1}(s,$
 $\quad \quad \quad \quad l,$
 $\quad \quad \quad \quad \text{table}),$
 $\quad \quad \quad \quad \text{dv}(\text{s-pos}(s), 1)),$
 $\quad \quad \quad c),$
 $\quad \quad \text{top}(\text{p-temp-stk}(\text{lr-eval}(\mathbf{t},$

```

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)))
 $\wedge$ (car (s-expr (s)) = S-TEMP-FETCH)
 $\wedge$ (*flag* $\neq$ 'list)
 $\wedge$ (*c* $\neq$ 0)
 $\wedge$ listp (s-expr (s))
 $\wedge$ s-good-statep (s, *c*)
 $\rightarrow$ (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)))
 $\wedge$ (car (s-expr (s)) = 'quote)
 $\wedge$ (*flag* $\neq$ 'list)
 $\wedge$ (*c* $\neq$ 0)
 $\wedge$ listp (s-expr (s))
 $\wedge$ s-good-statep (s, *c*)
 $\rightarrow$ (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*)) = ($\neg$ listp (*y*))

DEFINITION:
 induct-hint-13 (*e*, *x*, *y*)

```

=  if listp(x)
    then if listp(y)
        then if e = caar(x) then t
              else induct-hint-13(e, cdr(x), 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
 $(\text{proper-p-statep}(\text{lr-}>\text{p}(l))$
 $\wedge \text{good-posp}(flag, \text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{lr-programs-properp}(l, table)$
 $\wedge (\text{p-psw}(\text{lr-eval}(flag, l, c)) = \text{'run}))$
 $\rightarrow (\text{definedp}(x, \text{firstn}(n, \text{bindings}(\text{car}(\text{p-ctrl-stk}(\text{lr-eval}(flag, l, c)))))$
 $= \text{definedp}(x, \text{firstn}(n, \text{bindings}(\text{car}(\text{p-ctrl-stk}(l)))))$

DEFINITION:
 $\text{disjointp}(list1, list2)$
 $= \text{if listp}(list1)$
 $\text{then } (\text{car}(list1) \notin list2) \wedge \text{disjointp}(\text{cdr}(list1), list2)$
 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
 $(\text{proper-p-statep}(\text{lr-}>\text{p}(l))$
 $\wedge \text{good-posp1}(pos, \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{lr-programs-properp}(l, table)$
 $\wedge (\text{p-psw}(\text{lr-eval}(\mathbf{t}, \text{lr-set-pos}(l, pos), c)) = \text{'run}))$
 $\rightarrow (\text{definedp}(x,$
 $\text{firstn}(n,$
 $\text{bindings}(\text{car}(\text{p-ctrl-stk}(\text{lr-eval}(\mathbf{t},$
 $\text{lr-set-pos}(l, pos),$
 $c)))))$
 $= \text{definedp}(x, \text{firstn}(n, \text{bindings}(\text{car}(\text{p-ctrl-stk}(l)))))$

THEOREM: lr-params-p-frame-not-definedp-put-assoc-anything
 $(\text{proper-p-statep}(\text{lr-}>\text{p}(l))$
 $\wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{good-posp1}(pos, \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{lr-programs-properp}(l, table)$
 $\wedge \text{disjointp}(\text{formal-vars}(\text{p-current-program}(l)),$

$$\begin{aligned}
& \text{strip-cars}(\text{temp-var-dcls}(\text{p-current-program}(l))) \\
\wedge & \text{listp}(\text{lr-expr}(l)) \\
\wedge & ((\text{car}(\text{lr-expr}(l)) = \text{S-TEMP-EVAL}) \\
& \vee (\text{car}(\text{lr-expr}(l)) = \text{S-TEMP-TEST})) \\
\wedge & (\text{p-psw}(\text{lr-eval}(\mathbf{t}, \text{lr-set-pos}(l, \text{pos}), c)) = \text{'run})) \\
\rightarrow & (\text{lr-params}(\text{p-frame}(\text{put-assoc}(\text{anything}, \\
& \text{caddr}(\text{lr-expr}(l)), \\
& \text{bindings}(\text{car}(\text{p-ctrl-stk}(\text{lr-eval}(\mathbf{t}, \\
& \text{lr-set-pos}(l, \\
& \text{pos}), \\
& c))))), \\
& \text{ret-pc}), \\
& l)) \\
= & \text{lr-params}(\text{car}(\text{p-ctrl-stk}(\text{lr-eval}(\mathbf{t}, \text{lr-set-pos}(l, \text{pos}), c))), l))
\end{aligned}$$

DEFINITION:

$$\begin{aligned}
& \text{induct-hint-14}(s\text{-temps}, lr\text{-temps}, temp\text{-alist}) \\
= & \text{if listp}(s\text{-temps}) \\
& \text{then if listp}(lr\text{-temps}) \\
& \quad \text{then if listp}(temp\text{-alist}) \\
& \quad \quad \text{then if caddr}(lr\text{-temps}) = \text{LR-UNDEF-ADDR} \\
& \quad \quad \quad \text{then induct-hint-14}(\text{cdr}(s\text{-temps}), \\
& \quad \quad \quad \text{cdr}(lr\text{-temps}), \\
& \quad \quad \quad \text{cdr}(temp\text{-alist})) \\
& \quad \quad \quad \text{else induct-hint-14}(\text{cdr}(s\text{-temps}), \\
& \quad \quad \quad \text{cdr}(lr\text{-temps}), \\
& \quad \quad \quad \text{cdr}(temp\text{-alist})) \text{ endif} \\
& \quad \text{else t endif} \\
& \text{else t endif} \\
& \text{else t endif}
\end{aligned}$$

THEOREM: put-assoc-opener-1

$$\begin{aligned}
& ((\text{name} \neq \text{caar}(\text{alist})) \wedge \text{listp}(\text{alist})) \\
\rightarrow & (\text{put-assoc}(\text{val}, \text{name}, \text{alist}) \\
& = \text{cons}(\text{car}(\text{alist}), \text{put-assoc}(\text{val}, \text{name}, \text{cdr}(\text{alist}))))
\end{aligned}$$

THEOREM: put-assoc-opener-2

$$\begin{aligned}
& (\text{listp}(\text{alist3}) \\
& \wedge (\text{caar}(\text{alist3}) \notin \text{strip-cars}(\text{cdr}(\text{alist3}))) \\
& \wedge \text{definedp}(s\text{-expr}, \text{alist1}) \\
& \wedge (\text{strip-cars}(\text{alist1}) = \text{strip-cars}(\text{alist2})) \\
& \wedge (\text{strip-cdrs}(\text{alist2}) = \text{strip-cars}(\text{cdr}(\text{alist3})))) \\
\rightarrow & (\text{put-assoc}(\text{val}, \text{cdr}(\text{assoc}(s\text{-expr}, \text{alist2})), \text{alist3}) \\
& = \text{cons}(\text{car}(\text{alist3}), \\
& \quad \text{put-assoc}(\text{val}, \text{cdr}(\text{assoc}(s\text{-expr}, \text{alist2})), \text{cdr}(\text{alist3}))))
\end{aligned}$$

THEOREM: not-lr-valp-lr-undef-addr
 $(\text{lr-proper-heapp} (data-seg) \wedge \text{lr-proper-p-areasp} (data-seg))$
 $\rightarrow (\neg \text{lr-valp} (value, \text{identity} (\text{LR-UNDEF-ADDR}), data-seg))$

THEOREM: lr-s-similar-temps-put-assoc-put-assoc-helper-1
 $(\text{listp} (s-temps))$
 $\wedge \text{listp} (lr-temps)$
 $\wedge \text{lr-s-similar-temps} (s-temps, lr-temps, data-seg)$
 $\wedge \text{lr-valp} (value, addr, data-seg)$
 $\wedge \text{lr-proper-heapp} (data-seg)$
 $\wedge \text{lr-proper-p-areasp} (data-seg)$
 $\wedge (name1 = \text{caar} (s-temps))$
 $\wedge (name2 = \text{caar} (lr-temps))$
 $\rightarrow \text{lr-s-similar-temps} (\text{put-assoc} (\text{list} (t, value), name1, s-temps),$
 $\text{put-assoc} (addr, name2, lr-temps),$
 $data-seg)$

THEOREM: lr-s-similar-temps-put-assoc-put-assoc-helper
 $(\text{lr-s-similar-temps} (s-temps, lr-temps, data-seg))$
 $\wedge \text{lr-valp} (value, addr, data-seg)$
 $\wedge \text{lr-proper-heapp} (data-seg)$
 $\wedge \text{lr-proper-p-areasp} (data-seg)$
 $\wedge (\text{strip-cars} (temp-alist) = \text{strip-cars} (s-temps))$
 $\wedge (\text{strip-cdrs} (temp-alist) = \text{strip-cars} (lr-temps))$
 $\wedge \text{no-duplicatesp} (\text{strip-cars} (lr-temps))$
 $\wedge \text{definedp} (s-expr, s-temps)$
 $\rightarrow \text{lr-s-similar-temps} (\text{put-assoc} (\text{list} (t, value), s-expr, s-temps),$
 $\text{put-assoc} (addr,$
 $\text{cdr} (\text{assoc} (s-expr, temp-alist)),$
 $lr-temps),$
 $data-seg)$

THEOREM: disjointp-cons-arg2
 $(\text{disjointp} (list1, list2) \wedge (x \notin list1))$
 $\rightarrow \text{disjointp} (list1, \text{cons} (x, list2))$

THEOREM: disjointp-nlistp-arg2
 $(list2 \simeq \mathbf{nil}) \rightarrow \text{disjointp} (list1, list2)$

THEOREM: disjointp-lr-make-temp-name-alist-1
 $\text{disjointp} (formals,$
 $\text{strip-cdrs} (\text{lr-make-temp-name-alist-1} (initial,$
 $num-list,$
 $temp-list,$
 $formals)))$

THEOREM: lr-s-similar-statesp-s-change-temp-helper-2

```

let lr-eval be lr-eval(t, lr-set-pos(s->lr1(s, l, table), pos), c)
in
  (proper-p-statep(lr->p(s->lr1(s, l, table)))
   ^ good-posp1(s-pos(s), s-body(s-prog(s)))
   ^ good-posp1(pos, s-body(s-prog(s)))
   ^ lr-s-similar-statesp(s-params, s-temps(s-eval), lr-eval, table)
   ^ lr-programs-properp(s->lr1(s, l, table), table)
   ^ s-good-statep(s, c)
   ^ listp(s-expr(s))
   ^ ((car(s-expr(s)) = S-TEMP-EVAL)
      ∨ (car(s-expr(s)) = S-TEMP-TEST))
   ^ (p-psw(lr-eval) = 'run)
   ^ (lr-expr = caddr(lr-expr(s->lr1(s, l, table))))
  → (lr-s-similar-statesp(s-params,
                          s-temps(s-change-temp(s-eval,
                                                  s-expr,
                                                  value)),
                          lr-set-temp(lr-eval, addr, lr-expr),
                          table)
     = lr-s-similar-temps(put-assoc(list(t, value),
                                         s-expr,
                                         s-temps(s-eval)),
                          lr-temps(p-frame(put-assoc(addr,
                                                       lr-expr,
                                                       bindings(car(p-ctrl-stk(lr-eval)))),
                                   ret-pc(car(p-ctrl-stk(lr-eval))),
                                   s->lr1(s, l, table)),
                                   p-data-segment(lr-eval))) endlet

```

THEOREM: good-posp1-dv-1-temps-lr-expr

```

(((car(s-expr(s)) = S-TEMP-EVAL) ∨ (car(s-expr(s)) = S-TEMP-TEST))
 ^ listp(s-expr(s))
 ^ good-posp1(s-pos(s), s-body(assoc(s-pname(s), s-progs(s))))
 → good-posp1(dv(s-pos(s), 1), s-body(assoc(s-pname(s), s-progs(s))))

```

THEOREM: put-assoc-restn

```

(¬ definedp(name, firstn(n, alist)))
 → (put-assoc(val, name, restn(n, alist)))
   = restn(n, put-assoc(val, name, alist)))

```

THEOREM: disjointp-plist-arg-2

```

disjointp(x, plist(y)) = disjointp(x, y)

```

THEOREM: not-disjointp-member-arg1-cons-arg2

```

(v ∈ y) → (¬ disjointp(y, 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
 $\text{disjointp}(\text{strip-cdrs}(\text{lr-make-temp-name-alist-1}(\text{initial},$
 $\text{num-list},$
 $\text{temp-list},$
 $\text{formals})),$
 $\text{formals})$

THEOREM: proper-p-statep-lr->p-s->lr1-strip-cars-bindings-ctrl-stk
 $(\text{proper-p-statep}(\text{lr->p}(\text{s->lr1}(s, l, \text{table})))$
 $\wedge \text{definedp}(\text{s-pname}(s), \text{s-progs}(s)))$
 $\rightarrow (\text{strip-cars}(\text{bindings}(\text{car}(\text{p-ctrl-stk}(l))))$
 $= \text{append}(\text{s-formals}(\text{assoc}(\text{s-pname}(s), \text{s-progs}(s))),$
 $\text{strip-cdrs}(\text{lr-make-temp-name-alist}(\text{s-temp-list}(\text{assoc}(\text{s-pname}(s),$
 $\text{s-progs}(s))),$
 $\text{s-formals}(\text{assoc}(\text{s-pname}(s),$
 $\text{s-progs}(s))))))$

THEOREM: lr-programs-properp-lr->p-s->lr1-definedp-s-pname
 $(\neg \text{definedp}(\text{s-pname}(s), \text{s-progs}(s)))$
 $\rightarrow (\neg \text{lr-programs-properp}(\text{s->lr1}(s, l, \text{table}), \text{table}))$

THEOREM: lr-temps-p-frame-put-assoc
 $(\text{proper-p-statep}(\text{lr->p}(\text{s->lr1}(s, l, \text{table})))$
 $\wedge \text{lr-programs-properp}(\text{s->lr1}(s, l, \text{table}), \text{table})$
 $\wedge \text{definedp}(\text{s-pname}(s), \text{s-progs}(s))$
 $\wedge \text{listp}(\text{s-expr}(s))$
 $\wedge ((\text{car}(\text{s-expr}(s)) = \text{S-TEMP-EVAL})$
 $\vee (\text{car}(\text{s-expr}(s)) = \text{S-TEMP-TEST}))$
 $\wedge \text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s)))$
 $\wedge (\text{p-psw}(\text{lr-eval}(\mathbf{t}, l2, c)) = \text{'run})$
 $\wedge (\text{lr-expr} = \text{caddr}(\text{lr-expr}(\text{s->lr1}(s, l, \text{table}))))$
 $\wedge (l2 = \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), \text{dv}(\text{s-pos}(s), 1)))$
 $\rightarrow (\text{lr-temps}(\text{p-frame}(\text{put-assoc}(\text{val},$
 $\text{lr-expr},$
 $\text{bindings}(\text{car}(\text{p-ctrl-stk}(\text{lr-eval}(\mathbf{t}, l2, c))))),$
 $\text{ret-pc},$
 $\text{s->lr1}(s, l, \text{table}))$
 $= \text{put-assoc}(\text{val},$
 $\text{caddr}(\text{lr-expr}(\text{s->lr1}(s, l, \text{table}))),$

$$\text{lr-temps}(\text{car}(\text{p-ctrl-stk}(\text{lr-eval}(\mathbf{t}, l2, c))), \\ \text{s->lr1}(s, l, \text{table})))$$

THEOREM: strip-cars-lr-temps-strip-cars-temp-var-dcls

$$\begin{aligned} & (\text{s-good-statep}(s, c) \\ & \wedge \text{proper-p-statep}(\text{lr->p}(\text{s->lr1}(s, l, \text{table}))) \\ & \wedge (\text{frame} = \text{top}(\text{p-ctrl-stk}(\text{s->lr1}(s, l, \text{table})))) \\ \rightarrow & (\text{strip-cars}(\text{lr-temps}(\text{frame}, \text{s->lr1}(s, l, \text{table}))) \\ & = \text{strip-cdrs}(\text{lr-make-temp-name-alist}(\text{s-temp-list}(\text{assoc}(\text{s-pname}(s), \\ & \text{s-progs}(s))), \\ & \text{s-formals}(\text{assoc}(\text{s-pname}(s), \\ & \text{s-progs}(s))))) \end{aligned}$$

EVENT: Disable proper-p-statep-lr->p-s->lr1-strip-cars-bindings-ctrl-stk.

THEOREM: lr-s-similar-statesp-s-change-temp-helper-1

$$\begin{aligned} & (\text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s))) \\ & \wedge \text{listp}(\text{s-expr}(s)) \\ & \wedge ((\text{car}(\text{s-expr}(s)) = \text{S-TEMP-EVAL}) \\ & \vee (\text{car}(\text{s-expr}(s)) = \text{S-TEMP-TEST}) \\ & \vee (\text{car}(\text{s-expr}(s)) = \text{S-TEMP-FETCH})) \\ \rightarrow & (\text{caddr}(\text{lr-expr}(\text{s->lr1}(s, l, \text{table}))) \\ & = \text{cdr}(\text{assoc}(\text{cadr}(\text{s-expr}(s)), \\ & \text{lr-make-temp-name-alist}(\text{s-temp-list}(\text{s-prog}(s)), \\ & \text{s-formals}(\text{s-prog}(s))))) \end{aligned}$$

THEOREM: lr-s-similar-statesp-s->lr1-lr-similar-temps

$$\begin{aligned} & \text{lr-s-similar-statesp}(s\text{-params}, s\text{-temps}, l, \text{table}) \\ \rightarrow & \text{lr-s-similar-temps}(s\text{-temps}, \\ & \text{lr-temps}(\text{top}(\text{p-ctrl-stk}(l)), l), \\ & \text{p-data-segment}(l)) \end{aligned}$$

THEOREM: count-codelist1-cons

$$\text{count-codelist1}(\text{cons}(x, y)) = (x + (10 * \text{count-codelist1}(y)))$$

THEOREM: equal-append-initial

$$(\text{append}(x, y) = \text{append}(x, z)) = (y = z)$$

THEOREM: plist-listp-x-append-x-not-0

$$\text{plistp}(x) \rightarrow ((\text{append}(x, 0) = 0) = (x = \mathbf{nil}))$$

THEOREM: equal-append-final-0

$$(\text{append}(y, 0) = \text{append}(z, 0)) = (\text{plist}(y) = \text{plist}(z))$$

THEOREM: count-codelist1-append-non-listp

$$\begin{aligned} & (\neg \text{listp}(z)) \\ \rightarrow & \quad (\text{count-codelist1}(\text{append}(\text{num-list}, z)) = \text{count-codelist1}(\text{num-list})) \end{aligned}$$

THEOREM: not-equal-make-symbol-car-gensym

$$\begin{aligned} & (\text{count-codelist1}(\text{num-list1}) < \text{count-codelist1}(\text{num-list2})) \\ \rightarrow & \quad (\text{make-symbol}(\text{initial}, \text{num-list1}) \\ & \quad \neq \text{car}(\text{gensym}(\text{initial}, \text{num-list2}, \text{atom-list}))) \end{aligned}$$

THEOREM: count-codelist1-cdr-gensym

$$\begin{aligned} & (\text{count-codelist1}(\text{num-list1}) < \text{count-codelist1}(\text{num-list2})) \\ \rightarrow & \quad (\text{count-codelist1}(\text{num-list1}) \\ & \quad < \text{count-codelist1}(\text{cdr}(\text{gensym}(\text{initial}, \text{num-list2}, \text{atom-list})))) \end{aligned}$$

THEOREM: not-member-make-symbol-lr-make-temp-name-alist-1-incr

$$\begin{aligned} & (\text{count-codelist1}(\text{num-list1}) < \text{count-codelist1}(\text{num-list2})) \\ \rightarrow & \quad ((\text{make-symbol}(\text{initial}, \text{num-list1}) \\ & \quad \in \text{strip-cdrs}(\text{lr-make-temp-name-alist-1}(\text{initial}, \\ & \quad \quad \quad \text{num-list2}, \\ & \quad \quad \quad \text{temp-list}, \\ & \quad \quad \quad \text{formals})))) \\ & = \text{f}) \end{aligned}$$

THEOREM: not-member-car-gensym-lr-make-temp-name-alist-1-cdr

$$\begin{aligned} & (\text{car}(\text{gensym}(\text{initial}, \text{num-list}, \text{atoms}))) \\ \in & \quad \text{strip-cdrs}(\text{lr-make-temp-name-alist-1}(\text{initial}, \\ & \quad \quad \quad \text{cdr}(\text{gensym}(\text{initial}, \\ & \quad \quad \quad \text{num-list}, \\ & \quad \quad \quad \text{atoms})), \\ & \quad \quad \quad \text{temp-list}, \\ & \quad \quad \quad \text{formals}))) \\ & = \text{f} \end{aligned}$$

THEOREM: no-duplicatesp-strip-cdrs-lr-make-temp-name-alist-1

$$\begin{aligned} & \text{no-duplicatesp}(\text{strip-cdrs}(\text{lr-make-temp-name-alist-1}(\text{initial}, \\ & \quad \quad \quad \text{num-list}, \\ & \quad \quad \quad \text{temp-list}, \\ & \quad \quad \quad \text{formals})))) \end{aligned}$$

THEOREM: no-duplicatesp-strip-cdrs-lr-make-temp-name-alist

$$\text{no-duplicatesp}(\text{strip-cdrs}(\text{lr-make-temp-name-alist}(\text{temp-list}, \text{formals}))))$$

THEOREM: definedp-s-temps-s-eval

$$\begin{aligned} & (\text{s-err-flag}(\text{s-eval}(\text{flag}, s, c)) = \text{'run}) \\ \rightarrow & \quad (\text{definedp}(x, \text{s-temps}(\text{s-eval}(\text{flag}, s, c))) = \text{definedp}(x, \text{s-temps}(s))) \end{aligned}$$

THEOREM: strip-cars-lr-make-temp-name-alist-1
 strip-cars (lr-make-temp-name-alist-1 (*initial*, *num-list*, *temp-list*, *formals*))
 = plist (*temp-list*)

THEOREM: strip-cars-lr-make-temp-name-alist
 strip-cars (lr-make-temp-name-alist (*temp-list*, *formals*)) = plist (*temp-list*)

THEOREM: lr-eval-preserves-strip-cars-lr-temps-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 (lr-temps (car (p-ctrl-stk (lr-eval (*flag*, *l*, *c*))), *l2*))
 = strip-cars (lr-temps (car (p-ctrl-stk (*l*)), *l2*)))

THEOREM: lr-s-similar-statesp-s-change-temp
let *s-l* **be** s->lr1 (*s*, *l*, *table*),
 lr-eval **be** lr-eval (**t**, lr-set-pos (s->lr1 (*s*, *l*, *table*), *pos*), *c*)
in
 (proper-p-statep (lr->p (s->lr1 (*s*, *l*, *table*)))
 ∧ lr-s-similar-statesp (*s-params*, s-temps (*s-eval*), *lr-eval*, *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*)
 ∧ listp (p-temp-stk (*lr-eval*))
 ∧ lr-check-result (**t**,
 s-ans (*s-eval*),
 p-temp-stk (*lr-eval*),
 p-data-segment (*lr-eval*),
 orig-temp-stk)
 ∧ listp (s-expr (*s*))
 ∧ ((car (s-expr (*s*)) = S-TEMP-EVAL)
 ∨ (car (s-expr (*s*)) = S-TEMP-TEST))
 ∧ (p-psw (*lr-eval*) = 'run)
 ∧ (s-err-flag (*s-eval*) = 'run)
 ∧ (*s-eval* = s-eval (**t**, s-set-pos (*s*, *pos*), *c*))
 ∧ (*value* = caddr (lr-expr (*s-l*)))
 ∧ (*pos* = dv (s-pos (*s*), 1)))
 → lr-s-similar-statesp (*s-params*,
 s-temps (s-change-temp (*s-eval*,
 cadr (s-expr (*s*)),
 s-ans (*s-eval*))),
 lr-set-temp (*lr-eval*,
 car (p-temp-stk (*lr-eval*)),
 value),

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-similar-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*)
 $\wedge$ (strip-cdrs (*alist*) = *list*)
 $\wedge$ (*name* $\notin$ *list*)
 $\wedge$ definedp (*s-expr*, *alist*)
 $\rightarrow$ (cdr (assoc (*s-expr*, *alist*)) $\neq$ *name*)

THEOREM: not-member-no-duplicates-cdr-assoc
(no-duplicatesp (*list*)
 $\wedge$ (strip-cdrs (*alist1*) = *list*)
 $\wedge$ (*name* $\notin$ *list*)
 $\wedge$ definedp (*s-expr*, *alist2*)
 $\wedge$ (strip-cars (*alist2*) = strip-cars (*alist1*)))
 $\rightarrow$ (cdr (assoc (*s-expr*, *alist1*)) $\neq$ *name*)

THEOREM: not-equal-lr-s-eval-temp-setp-not-lr-s-similar-temps
((*lr-expr* = cdr (assoc (*s-expr*, *temp-alist*)))
 $\wedge$ lr-proper-heapp (*data-seg*)
 $\wedge$ lr-proper-p-areasp (*data-seg*)
 $\wedge$ lr-s-similar-temps (*s-temps*, *lr-temps*, *data-seg*)
 $\wedge$ (strip-cars (*temp-alist*) = strip-cars (*s-temps*))
 $\wedge$ (strip-cdrs (*temp-alist*) = strip-cdrs (*lr-temps*))
 $\wedge$ no-duplicatesp (strip-cars (*lr-temps*))
 $\wedge$ definedp (*s-expr*, *s-temps*)
 $\rightarrow$ ((cdr (assoc (*lr-expr*, *lr-temps*)) $\neq$ LR-UNDEF-ADDR)
 $\leftrightarrow$ cadr (assoc (*s-expr*, *s-temps*)))

THEOREM: definedp-strip-cars-append-member-x-2
(strip-cars (*x*) = append (*y*, *z*))
 $\rightarrow$ ((*e* $\in$ *z*) = definedp (*e*, restn (length (*y*), *x*)))

THEOREM: not-iff-lr-s-temp-setp-not-lr-s-similar-statesp-helper
 $((x \in \text{strip-cdrs}(\text{lr-make-temp-name-alist}(\text{s-temp-list}(\text{assoc}(\text{s-pname}(s), \text{s-progs}(s))), \text{s-formals}(\text{assoc}(\text{s-pname}(s), \text{s-progs}(s)))))$
 $\wedge \text{proper-p-statep}(\text{lr->p}(\text{s->lr1}(s, l, \text{table})))$
 $\wedge \text{definedp}(\text{s-pname}(s), \text{s-progs}(s)))$
 $\rightarrow (\text{cdr}(\text{assoc}(x, \text{lr-temps}(\text{car}(\text{p-ctrl-stk}(l)), \text{s->lr1}(s, l, \text{table}))))$
 $= \text{cdr}(\text{assoc}(x, \text{bindings}(\text{car}(\text{p-ctrl-stk}(l)))))$

THEOREM: lr-programs-properp-member-lr-expr-temps
 $(\text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{lr-programs-properp}(l, \text{table})$
 $\wedge ((\text{car}(\text{lr-expr}(l)) = \text{S-TEMP-FETCH})$
 $\vee (\text{car}(\text{lr-expr}(l)) = \text{S-TEMP-EVAL})$
 $\vee (\text{car}(\text{lr-expr}(l)) = \text{S-TEMP-TEST}))$
 $\rightarrow (\text{caddr}(\text{lr-expr}(l))$
 $\in \text{strip-cars}(\text{temp-var-dcls}(\text{p-current-program}(l))))$

THEOREM: not-iff-lr-s-temp-setp-not-lr-s-similar-statesp
 $(\text{proper-p-statep}(\text{lr->p}(\text{s->lr1}(s, l, \text{table})))$
 $\wedge \text{lr-proper-heapp}(\text{p-data-segment}(l))$
 $\wedge \text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s)))$
 $\wedge \text{lr-programs-properp}(\text{s->lr1}(s, l, \text{table}), \text{table})$
 $\wedge \text{listp}(\text{s-expr}(s))$
 $\wedge ((\text{car}(\text{s-expr}(s)) = \text{S-TEMP-TEST})$
 $\vee (\text{car}(\text{s-expr}(s)) = \text{S-TEMP-FETCH}))$
 $\wedge (\neg (\text{lr-eval-temp-setp}(\text{s->lr1}(s, l, \text{table}))$
 $\leftrightarrow \text{s-temp-setp}(\text{cadr}(\text{s-expr}(s)), \text{s-temps}(s))))$
 $\wedge \text{s-good-statep}(s, c))$
 $\rightarrow (\neg \text{lr-s-similar-statesp}(s\text{-params}, \text{s-temps}(s), \text{s->lr1}(s, l, \text{table}), \text{table}))$

THEOREM: lr-valp-lr-s-eval-lr-s-similar-temps
 $((\text{lr-expr} = \text{cdr}(\text{assoc}(s\text{-expr}, \text{temp-alist})))$
 $\wedge \text{lr-proper-heapp}(\text{data-seg})$
 $\wedge \text{lr-proper-p-areasp}(\text{data-seg})$
 $\wedge \text{lr-s-similar-temps}(s\text{-temps}, \text{lr-temps}, \text{data-seg})$
 $\wedge (\text{strip-cars}(\text{temp-alist}) = \text{strip-cars}(s\text{-temps}))$
 $\wedge (\text{strip-cdrs}(\text{temp-alist}) = \text{strip-cdrs}(\text{lr-temps}))$
 $\wedge \text{no-duplicatesp}(\text{strip-cars}(\text{lr-temps}))$
 $\wedge \text{definedp}(s\text{-expr}, s\text{-temps})$
 $\wedge (\text{cdr}(\text{assoc}(\text{lr-expr}, \text{lr-temps})) \neq \text{LR-UNDEF-ADDR}))$
 $\rightarrow \text{lr-valp}(\text{caddr}(\text{assoc}(s\text{-expr}, s\text{-temps})),$
 $\text{cdr}(\text{assoc}(\text{lr-expr}, \text{lr-temps})),$
 $\text{data-seg})$

THEOREM: member-cdr-assoc-strip-cdrs-definedp
 $\text{definedp}(x, \text{alist}) \rightarrow (\text{cdr}(\text{assoc}(x, \text{alist})) \in \text{strip-cdrs}(\text{alist}))$

THEOREM: definedp-pairlist
 $\text{definedp}(x, \text{pairlist}(\text{temp-list}, \text{anything})) = (x \in \text{temp-list})$

THEOREM: definedp-lr-make-temp-name-alist-1
 $\text{definedp}(x, \text{lr-make-temp-name-alist-1}(\text{initial}, \text{num-list}, \text{temp-list}, \text{formals}))$
 $= (x \in \text{temp-list})$

THEOREM: definedp-lr-make-temp-name-alist
 $\text{definedp}(x, \text{lr-make-temp-name-alist}(\text{temp-list}, \text{formals}))$
 $= (x \in \text{temp-list})$

THEOREM: p-temp-stk-lr-do-temp-fetch-p-psw-run
 $(\text{p-psw}(\text{lr-do-temp-fetch}(l)) = \text{'run})$
 $\rightarrow (\text{p-temp-stk}(\text{lr-do-temp-fetch}(l))$
 $= \text{push}(\text{local-var-value}(\text{caddr}(\text{lr-expr}(l)), \text{p-ctrl-stk}(l)),$
 $\text{p-temp-stk}(l)))$

THEOREM: lr-check-result-lr-do-temp-fetch
 $(\text{proper-p-statep}(\text{lr->p}(\text{s->lr1}(s, l, \text{table})))$
 $\wedge \text{good-pospl}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s)))$
 $\wedge \text{s-good-statep}(s, c)$
 $\wedge (c \neq 0)$
 $\wedge \text{listp}(\text{s-expr}(s))$
 $\wedge ((\text{car}(\text{s-expr}(s)) = \text{S-TEMP-TEST})$
 $\vee (\text{car}(\text{s-expr}(s)) = \text{S-TEMP-FETCH}))$
 $\wedge \text{lr-proper-heapp}(\text{p-data-segment}(l))$
 $\wedge \text{lr-s-similar-statesp}(\text{s-params}(s), \text{s-temps}(s), \text{s->lr1}(s, l, \text{table}), \text{table})$
 $\wedge \text{lr-eval-temp-setp}(\text{s->lr1}(s, l, \text{table}))$
 $\wedge (\text{value} = \text{caddr}(\text{lr-expr}(\text{s->lr1}(s, l, \text{table}))))$
 $\rightarrow \text{lr-check-result}(t,$
 $\text{caddr}(\text{assoc}(\text{cadr}(\text{s-expr}(s)), \text{s-temps}(s))),$
 $\text{cons}(\text{cdr}(\text{assoc}(\text{value}, \text{bindings}(\text{car}(\text{p-ctrl-stk}(l)))))),$
 $\text{p-temp-stk}(l),$
 $\text{p-data-segment}(l),$
 $\text{p-temp-stk}(l))$

THEOREM: lr-do-temp-fetch-run-lr-eval-temp-setp
 $(\text{p-psw}(\text{lr-do-temp-fetch}(l)) = \text{'run}) \rightarrow \text{lr-eval-temp-setp}(l)$

THEOREM: lr-s-similar-const-table-lr-valp-assoc
 $(\text{definedp}(\text{value}, \text{table}) \wedge \text{lr-s-similar-const-table}(\text{table}, \text{data-seg}))$
 $\rightarrow \text{lr-valp}(\text{value}, \text{cdr}(\text{assoc}(\text{value}, \text{table})), \text{data-seg})$

THEOREM: lr-proper-expr-list-quote-opener

```

(flag ≠ 'list)
→ (lr-proper-expr (flag,
                    list ('quote, addr),
                    program-names,
                    formals,
                    temps,
                    table)
    = ((type (addr) = 'addr) ∧ (addr ∈ strip-cdrs (table))))

```

THEOREM: lr-check-result-lr-push-tstk-quote

```

(good-pospl (s-pos (s), s-body (s-prog (s)))
 ∧ listp (s-expr (s))
 ∧ (car (s-expr (s)) = 'quote)
 ∧ 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)
 ∧ lr-proper-heapp (p-data-segment (l))
 ∧ (p-psw (lr-push-tstk (s->lr1 (s, l, table),
                        cadr (lr-expr (s->lr1 (s, l, table))))))
    = 'run))
→ lr-check-result (t,
                    cadr (s-expr (s)),
                    p-temp-stk (lr-push-tstk (s->lr1 (s, l, table),
                                                cadr (lr-expr (s->lr1 (s, l, table))))),
                    p-data-segment (l),
                    p-temp-stk (l))

```

THEOREM: lr-eval-subrp-user-funcall-opener

```

let lr-eval-list be lr-eval ('list,
                             lr-set-pos (s->lr1 (s, l, table), dv (s-pos (s), 1)),
                             c)

in
((flag ≠ 'list)
 ∧ (c ≠ 0)
 ∧ listp (s-expr (s))
 ∧ (car (s-expr (s)) ≠ 'if)
 ∧ (car (s-expr (s)) ≠ S-TEMP-EVAL)
 ∧ (car (s-expr (s)) ≠ S-TEMP-TEST)
 ∧ (car (s-expr (s)) ≠ S-TEMP-FETCH)
 ∧ (car (s-expr (s)) ≠ 'quote)
 ∧ good-pospl (s-pos (s), s-body (s-prog (s)))
 ∧ s-good-statep (s, c))
→ (lr-eval (flag, s->lr1 (s, l, table), c)

```



```

else lr-set-error (s->lr1 (s, l, table),
                    'bad-instruction) endif) endlet

```

$$\begin{aligned} & \wedge (\text{subrp}(\text{car}(\text{lr-expr}(l))) \vee \text{litatom}(\text{car}(\text{lr-expr}(l)))) \\ \rightarrow & (\text{length}(\text{cdr}(\text{lr-expr}(l))) = \text{arity}(\text{car}(\text{lr-expr}(l)))) \end{aligned}$$

```
else induct-hint-8( $n - 1$ , cdr(value), cdr(temp-stk)) endif
```

$$\rightarrow \text{lr-valp}(\text{get}(n, \text{values}), \text{get}(n, \text{temp-stk}), \text{data-seg})$$
$$\text{lr-valp}(value, addr, data-seg) \rightarrow \text{lr-good-pointerp}(addr, data-seg)$$
$$\begin{aligned} \rightarrow & ((\text{type}(\text{car}(\text{temp-stk})) = \text{'addr}) \\ & \wedge (\text{caddr}(\text{car}(\text{temp-stk})) = \text{nil}) \\ & \wedge \text{listp}(\text{car}(\text{temp-stk})) \\ & \wedge \text{adpp}(\text{untag}(\text{car}(\text{temp-stk})), \text{data-seg}) \\ & \wedge \text{lr-boundary-nodep}(\text{car}(\text{temp-stk})) \\ & \wedge (\text{area-name}(\text{car}(\text{temp-stk})) = \text{identity}(\text{LR-HEAP-NAME})) \\ & \wedge (\text{type}(\text{fetch}(\text{add-addr}(\text{car}(\text{temp-stk}), \\ & \qquad \qquad \qquad \text{identity}(\text{LR-REF-COUNT-OFFSET})), \\ & \qquad \qquad \qquad \text{data-seg})) \\ & = \text{'nat})) \end{aligned}$$

THEOREM: lr-check-result1-lr-good-pointerp-get-n-lessp-cadr
 (lr-check-result1 (*values*, *temp-stk*, *data-seg*) $\wedge$ (length(*values*) $\not\leq$ 2))
 $\rightarrow$ ((type(cadr(*temp-stk*)) = 'addr)
 $\wedge$ (caddr(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)

```



```

 $\wedge$  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-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)))
 $\wedge$  (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)))
 $\wedge$  (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)))
 $\wedge$  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-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)))
 $\wedge$  lr-programs-properp (l, table1)
 $\wedge$  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-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)))
 $\rightarrow$  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*)
 $\rightarrow$ (length (*temp-stk*) $\not\leq$ 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)))
 $\rightarrow$ (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*))
 $\wedge$ (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 (*new-l*) = 'run)
 $\wedge$ (p-psw (lr-apply-subr (*l*, *new-l*)) = 'run)
 $\wedge$ (*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*)
 $\wedge$ 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*)
 $\wedge$ lr-good-pointerp (cadr (*temp-stk*), *data-seg*)
 $\wedge$ lr-valp (car (*values*), cadr (*temp-stk*), *data-seg*)
 $\wedge$ lr-good-pointerp (car (*temp-stk*), *data-seg*)))

THEOREM: lr-valp-fetch-tag-cons-lr-valp-car-cdr
(lr-valp (*value*, *addr*, *data-seg*)
 $\wedge$ (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)
 $\wedge$ lr-valp (cdr (*value*),
fetch (add-addr (*addr*, identity (LR-CDR-OFFSET)), *data-seg*),
data-seg))

THEOREM: lr-good-pointerp-type-tag-nat

$(\text{lr-proper-heapp} (data-seg) \wedge \text{lr-good-pointerp} (addr, data-seg))$
 $\rightarrow (\text{type} (\text{fetch} (addr, data-seg)) = \text{'nat})$

THEOREM: lr-proper-heapp-lr-valp-f-helper
 $(\text{lr-proper-heapp-nodep} (\text{LR-F-ADDR}, data-seg)$
 $\wedge \text{lr-proper-heapp-nodep} (addr, data-seg)$
 $\wedge \text{lr-nodep} (addr, data-seg)$
 $\wedge \text{lr-minimum-heapp} (data-seg)$
 $\wedge \text{lr-proper-p-areasp} (data-seg))$
 $\rightarrow (\text{lr-valp} (\mathbf{f}, addr, data-seg) = (addr = \text{LR-F-ADDR}))$

THEOREM: lr-proper-heapp-lr-valp-f
 $(\text{lr-proper-heapp} (data-seg) \wedge \text{lr-proper-p-areasp} (data-seg))$
 $\rightarrow (\text{lr-valp} (\mathbf{f}, addr, data-seg) = (addr = \text{LR-F-ADDR}))$

THEOREM: lr-valp-equal-value-fact
 $(\text{lr-valp} (value1, addr, data-seg) \wedge \text{lr-valp} (value2, addr, data-seg))$
 $\rightarrow (value1 = value2)$

THEOREM: lr-proper-heapp-lr-valp-0
 $\text{lr-proper-heapp} (data-seg)$
 $\rightarrow (\text{lr-valp} (value, \text{identity} (\text{LR-0-ADDR}), data-seg) = (value = 0))$

THEOREM: lr-proper-heapp-lr-valp-lr-f-addr
 $(\text{lr-proper-heapp} (data-seg) \wedge \text{lr-proper-p-areasp} (data-seg))$
 $\rightarrow (\text{lr-valp} (value, \text{identity} (\text{LR-F-ADDR}), data-seg) = (value = \mathbf{f}))$

THEOREM: lr-proper-heapp-lr-valp-lr-t-addr
 $\text{lr-proper-heapp} (data-seg)$
 $\rightarrow (\text{lr-valp} (value, \text{identity} (\text{LR-T-ADDR}), data-seg) = (value = \mathbf{t}))$

THEOREM: lr-valp-fetch-tag-not-cons-lr-valp-car-cdr-0
 $(\text{proper-p-statep} (\text{lr->p} (l))$
 $\wedge \text{lr-proper-heapp} (\text{p-data-segment} (l))$
 $\wedge (\text{fetch} (\text{car} (\text{p-temp-stk} (l)), \text{p-data-segment} (l))$
 $\neq \text{tag} (\text{'nat}, \text{LR-CONS-TAG}))$
 $\wedge ((\text{car} (value) \neq 0) \vee (\text{cdr} (value) \neq 0)))$
 $\rightarrow (\neg \text{lr-valp} (value, \text{car} (\text{p-temp-stk} (l)), \text{p-data-segment} (l)))$

THEOREM: lr-valp-not-tag-cons-not-listp
 $((\neg \text{listp} (value)) \wedge (\text{fetch} (addr, data-seg) = \text{tag} (\text{'nat}, \text{LR-CONS-TAG})))$
 $\rightarrow (\neg \text{lr-valp} (value, addr, data-seg))$

THEOREM: lr-valp-fetch-tag-not-cons-lr-valp-listp
 $(\text{proper-p-statep} (\text{lr->p} (l))$
 $\wedge \text{lr-proper-heapp} (\text{p-data-segment} (l))$

$$\begin{aligned}
& \wedge \text{ (fetch (car (p-temp-stk } (l)), \text{ p-data-segment } (l))} \\
& \quad \neq \text{ tag ('nat, LR-CONS-TAG))} \\
& \wedge \text{ listp (value))} \\
\rightarrow & \text{ (}\neg \text{ lr-valp (value, car (p-temp-stk } (l)), \text{ p-data-segment } (l)))
\end{aligned}$$

THEOREM: lr-valp-cons

$$\begin{aligned}
& \text{lr-valp (cons (x, y), addr, data-seg)} \\
= & \text{ if lr-good-pointerp (addr, data-seg)} \\
& \text{ then (untag (fetch (addr, data-seg)) = LR-CONS-TAG)} \\
& \quad \wedge \text{ lr-valp (x,} \\
& \quad \quad \text{fetch (add-addr (addr, LR-CAR-OFFSET), data-seg),} \\
& \quad \quad \text{data-seg)} \\
& \quad \wedge \text{ lr-valp (y,} \\
& \quad \quad \text{fetch (add-addr (addr, LR-CDR-OFFSET), data-seg),} \\
& \quad \quad \text{data-seg)} \\
& \text{ else f endif}
\end{aligned}$$

THEOREM: lr-valp-deposit-a-list-cons

$$\begin{aligned}
& \text{(lr-proper-free-listp (data-seg))} \\
& \wedge \text{ lr-proper-p-areasp (data-seg)} \\
& \wedge \text{ lr-nodep (lr-max-node (data-seg), data-seg)} \\
& \wedge \text{ lr-minimum-heapp (data-seg)} \\
& \wedge \text{ lr-boundary-nodep (lr-max-node (data-seg))} \\
& \wedge \text{ lr-valp (value, addr, data-seg)} \\
& \wedge \text{ lr-proper-p-areasp (data-seg)} \\
& \wedge \text{ (fp-addr = fetch (identity (LR-FP-ADDR), data-seg))} \\
\rightarrow & \text{ lr-valp (value, addr, deposit-a-list (list (x0, x1, x2, x3), fp-addr, data-seg))}
\end{aligned}$$

THEOREM: lr-valp-car-p-temp-stk-p-run-subr-cons-helper

$$\begin{aligned}
& \text{(lr-proper-heapp (data-seg))} \\
& \wedge \text{ lr-valp (car, car-addr, data-seg)} \\
& \wedge \text{ lr-valp (cdr, cdr-addr, data-seg)} \\
& \wedge \text{ lr-proper-p-areasp (data-seg)} \\
& \wedge \text{ (type (ref-count) = 'nat)} \\
\rightarrow & \text{ lr-valp (cons (car, cdr),} \\
& \quad \text{fetch (identity (LR-FP-ADDR), data-seg),} \\
& \quad \text{deposit-a-list (list (identity (tag ('nat, LR-CONS-TAG)),} \\
& \quad \quad \text{ref-count,} \\
& \quad \quad \text{car-addr,} \\
& \quad \quad \text{cdr-addr),} \\
& \quad \text{fetch (identity (LR-FP-ADDR), data-seg),} \\
& \quad \text{data-seg))}
\end{aligned}$$

EVENT: Disable lr-valp-cons.

THEOREM: lr-valp-not-tag-true-not-listp
 $((\neg \text{truep}(\text{value})) \wedge (\text{fetch}(\text{addr}, \text{data-seg}) = \text{tag}(\text{'nat}, \text{LR-TRUE-TAG})))$
 $\rightarrow (\neg \text{lr-valp}(\text{value}, \text{addr}, \text{data-seg}))$

THEOREM: lr-valp-fetch-tag-not-true-lr-valp-listp
 $(\text{proper-p-statep}(\text{lr->p}(l))$
 $\wedge \text{lr-proper-heapp}(\text{p-data-segment}(l))$
 $\wedge (\text{fetch}(\text{car}(\text{p-temp-stk}(l)), \text{p-data-segment}(l))$
 $\neq \text{tag}(\text{'nat}, \text{LR-TRUE-TAG}))$
 $\rightarrow (\neg \text{lr-valp}(\text{t}, \text{car}(\text{p-temp-stk}(l)), \text{p-data-segment}(l)))$

THEOREM: lr-valp-car-p-temp-stk-p-run-subr
 $(\text{lr-proper-heapp}(\text{p-data-segment}(l))$
 $\wedge \text{lr-check-result1}(\text{reverse}(\text{values}), \text{p-temp-stk}(l), \text{p-data-segment}(l))$
 $\wedge (\text{length}(\text{values}) = \text{arity}(\text{subr}))$
 $\wedge \text{proper-p-statep}(\text{lr->p}(l))$
 $\wedge \text{lr-programs-properp}(l, \text{table})$
 $\wedge (\text{p-psw}(\text{p-run-subr}(\text{subr}, \text{p-set-pc}(\text{lr->p}(l), \text{pc}))) = \text{'run})$
 $\wedge (\text{p-psw}(l) = \text{'run})$
 $\wedge (\text{area-name}(\text{pc}) = \text{area-name}(\text{p-pc}(l)))$
 $\wedge (\text{unlabel}(\text{get}(\text{offset}(\text{pc}),$
 $\text{program-body}(\text{assoc}(\text{area-name}(\text{pc}),$
 $\text{p-prog-segment}(\text{lr->p}(l))))))$
 $= \text{list}(\text{'call}, \text{subr}))$
 $\rightarrow \text{lr-valp}(\text{apply-subr}(\text{subr}, \text{values}),$
 $\text{car}(\text{p-temp-stk}(\text{p-run-subr}(\text{subr}, \text{p-set-pc}(\text{lr->p}(l), \text{pc})))),$
 $\text{p-data-segment}(\text{p-run-subr}(\text{subr}, \text{p-set-pc}(\text{lr->p}(l), \text{pc}))))$

THEOREM: lr-programs-properp-not-definedp-subrp-runtime-support
 $(\text{subrp}(\text{car}(\text{s-expr}(s)))$
 $\wedge \text{listp}(\text{s-expr}(s))$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{'if})$
 $\wedge (\neg \text{definedp}(\text{car}(\text{s-expr}(s)), \text{P-RUNTIME-SUPPORT-PROGRAMS}))$
 $\wedge \text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s)))$
 $\rightarrow (\neg \text{lr-programs-properp}(\text{s->lr1}(s, l, \text{table}), \text{table}))$

THEOREM: lr-valp-apply-subr-lr-apply-subr
let *new-l* **be** $\text{lr-eval}(\text{'list}, \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), \text{pos}), c)$
in
 $(\text{listp}(\text{s-expr}(s))$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{'if})$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{'quote})$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{S-TEMP-EVAL})$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{S-TEMP-TEST})$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{S-TEMP-FETCH})$

```

^ 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)

```

$$\begin{aligned}
& \wedge \quad (\text{s-err-flag}(\text{s-eval}('list, \text{s-set-pos}(s, \text{dv}(\text{s-pos}(s), 1)), c)) \\
& \quad = 'run)) \\
\rightarrow & \quad \text{lr-check-result}(t, \\
& \quad \quad \text{apply-subr}(\text{car}(\text{s-expr}(s)), \\
& \quad \quad \quad \text{s-ans}(\text{s-eval}('list, \\
& \quad \quad \quad \quad \text{s-set-pos}(s, pos), \\
& \quad \quad \quad \quad c))), \\
& \quad \text{p-temp-stk}(\text{lr-apply-subr}(\text{s->lr1}(s, l, table), \\
& \quad \quad \quad new-l)), \\
& \quad \text{p-data-segment}(\text{lr-apply-subr}(\text{s->lr1}(s, l, table), \\
& \quad \quad \quad new-l)), \\
& \quad \text{p-temp-stk}(l)) \text{endlet}
\end{aligned}$$

THEOREM: s->lr1-lr-funcall-s-fun-call-state

$$\begin{aligned}
& (\text{listp}(\text{s-expr}(s))) \\
& \wedge \quad (\neg \text{subrp}(\text{car}(\text{s-expr}(s)))) \\
& \wedge \quad (\text{car}(\text{s-expr}(s)) \neq 'quote) \\
& \wedge \quad (\text{car}(\text{s-expr}(s)) \neq 'if) \\
& \wedge \quad \text{litatom}(\text{car}(\text{s-expr}(s))) \\
& \wedge \quad \text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s))) \\
& \wedge \quad (\text{p-psw}(\text{lr-funcall}(\text{s->lr1}(s, l, table), lr-eval)) = 'run) \\
& \wedge \quad (\text{s-progs}(lr-eval) = \text{s-progs}(s)) \\
& \wedge \quad (\text{p-prog-segment}(lr-eval) = \text{p-prog-segment}(\text{s->lr1}(s, l, table))) \\
\rightarrow & \quad (\text{s->lr1}(\text{s-fun-call-state}(lr-eval, \text{car}(\text{s-expr}(s))), \\
& \quad \quad \text{lr-funcall}(\text{s->lr1}(s, l, table), lr-eval), \\
& \quad \quad table) \\
& \quad = \quad \text{lr-funcall}(\text{s->lr1}(s, l, table), lr-eval))
\end{aligned}$$

THEOREM: lr-params-lr-funcall

$$\begin{aligned}
& ((\text{p-psw}(\text{lr-funcall}(l1, l2)) = 'run) \\
& \wedge \quad (\text{p-prog-segment}(l1) = \text{p-prog-segment}(l2))) \\
\rightarrow & \quad (\text{lr-params}(\text{car}(\text{p-ctrl-stk}(\text{lr-funcall}(l1, l2))), \text{lr-funcall}(l1, l2)) \\
& \quad = \quad \text{pair-formal-vars-with-actuals}(\text{formal-vars}(\text{assoc}(\text{user-fname}(\text{car}(\text{lr-expr}(l1))), \\
& \quad \quad \quad \text{p-prog-segment}(l1))), \\
& \quad \quad \text{p-temp-stk}(l2)))
\end{aligned}$$

THEOREM: lr-temps-lr-funcall

$$\begin{aligned}
& ((\text{p-psw}(\text{lr-funcall}(l1, l2)) = 'run) \\
& \wedge \quad (\text{p-prog-segment}(l1) = \text{p-prog-segment}(l2))) \\
\rightarrow & \quad (\text{lr-temps}(\text{car}(\text{p-ctrl-stk}(\text{lr-funcall}(l1, l2))), \text{lr-funcall}(l1, l2)) \\
& \quad = \quad \text{pair-temps-with-initial-values}(\text{temp-var-dcls}(\text{assoc}(\text{user-fname}(\text{car}(\text{lr-expr}(l1))), \\
& \quad \quad \quad \text{p-prog-segment}(l1))))))
\end{aligned}$$

THEOREM: listp-pairlist

$$\text{listp}(\text{pairlist}(x, y)) = \text{listp}(x)$$

THEOREM: car-reverse-last
 $\text{car}(\text{reverse}(list)) = \text{car}(\text{last}(list))$

THEOREM: get-sub1-length-car-last
 $(\text{listp}(list) \wedge (n = (\text{length}(list) - 1)))$
 $\rightarrow (\text{get}(n, list) = \text{car}(\text{last}(list)))$

THEOREM: car-last-first-n-add1-get
 $\text{car}(\text{last}(\text{first-n}(1 + n, list))) = \text{get}(n, list)$

THEOREM: length-butlast
 $\text{length}(\text{butlast}(x)) = (\text{length}(x) - 1)$

DEFINITION:
 $\text{induct-hint-1}(x, y, z)$
 $=$ **if** $\text{listp}(x)$
 then if $\text{listp}(y)$
 then if $\text{listp}(z)$
 then $\text{induct-hint-1}(\text{cdr}(x), \text{butlast}(y), \text{butlast}(z))$
 else t endif
 else t endif
 else t endif

THEOREM: lr-check-result1-append-2
 $(\text{length}(values) = \text{length}(temp-stk1))$
 $\rightarrow (\text{lr-check-result1}(values, \text{append}(temp-stk1, temp-stk2), data-seg)$
 $= \text{lr-check-result1}(values, temp-stk1, data-seg))$

THEOREM: lr-check-result1-butlast
 $(\text{lr-check-result1}(values, temp-stk, data-seg)$
 $\wedge (\text{length}(temp-stk) = \text{length}(values))$
 $\wedge \text{listp}(temp-stk)$
 $\wedge \text{listp}(values))$
 $\rightarrow \text{lr-check-result1}(\text{butlast}(values), \text{butlast}(temp-stk), data-seg)$

THEOREM: reverse-butlast
 $\text{listp}(x) \rightarrow (\text{reverse}(\text{butlast}(x)) = \text{cdr}(\text{reverse}(x)))$

THEOREM: lr-s-similar-params-lr-valp-get
 $((n < \text{length}(s-params))$
 $\wedge (\text{strip-cars}(s-params) = \text{strip-cars}(lr-params))$
 $\wedge \text{lr-s-similar-params}(s-params, lr-params, data-seg))$
 $\rightarrow \text{lr-valp}(\text{cdr}(\text{get}(n, s-params)), \text{cdr}(\text{get}(n, lr-params)), data-seg)$

THEOREM: lr-s-similar-params-lr-funcall-helper-1
 $(\text{lr-s-similar-params}(\text{pairlist}(\text{cdr}(\text{formals}), \text{cdr}(\text{reverse}(\text{values}))),$
 $\text{pairlist}(\text{cdr}(\text{formals}), \text{cdr}(\text{reverse}(\text{temp-stk}))),$
 $\text{data-seg})$
 $\wedge \text{listp}(\text{formals})$
 $\wedge \text{listp}(\text{values})$
 $\wedge \text{listp}(\text{temp-stk})$
 $\wedge \text{lr-check-result1}(\text{values}, \text{temp-stk}, \text{data-seg})$
 $\wedge ((1 + \text{length}(\text{cdr}(\text{formals}))) = \text{length}(\text{temp-stk}))$
 $\wedge (\text{length}(\text{temp-stk}) = \text{length}(\text{values}))$
 $\rightarrow \text{lr-valp}(\text{car}(\text{last}(\text{values})), \text{car}(\text{last}(\text{temp-stk})), \text{data-seg})$

THEOREM: lr-s-similar-params-lr-funcall
 $(\text{lr-check-result1}(\text{values}, \text{temp-stk}, \text{data-seg})$
 $\wedge (\text{length}(\text{temp-stk}) = \text{length}(\text{values}))$
 $\wedge (\text{length}(\text{temp-stk}) = \text{length}(\text{formals})))$
 $\rightarrow \text{lr-s-similar-params}(\text{pairlist}(\text{formals}, \text{reverse}(\text{values})),$
 $\text{pairlist}(\text{formals}, \text{reverse}(\text{temp-stk})),$
 $\text{data-seg})$

THEOREM: append-first-n-restn
 $(\text{length}(l) \not\leq i) \rightarrow (\text{append}(\text{first-n}(i, l), \text{restn}(i, l)) = l)$

THEOREM: lr-check-result1-first-n-temp-stk
 $(\text{length}(\text{p-temp-stk}(l)) \not\leq \text{length}(\text{values}))$
 $\rightarrow (\text{lr-check-result1}(\text{values}, \text{p-temp-stk}(l), \text{data-seg})$
 $= \text{lr-check-result1}(\text{values},$
 $\text{first-n}(\text{length}(\text{values}), \text{p-temp-stk}(l)),$
 $\text{data-seg}))$

THEOREM: lr-push-tstk-length
 $(\text{p-psw}(\text{lr-push-tstk}(l, \text{object})) = \text{'run})$
 $\rightarrow (\text{length}(\text{p-temp-stk}(\text{lr-push-tstk}(l, \text{object})))$
 $= (1 + \text{length}(\text{p-temp-stk}(l))))$

THEOREM: length-add1-add1-cddr-fact
 $(\text{length}(x) = (1 + (1 + \text{length}(y)))) \rightarrow (\text{length}(\text{cddr}(x)) = \text{length}(y))$

THEOREM: length-p-temp-stk-p-run-subr-helper-1
 $(\text{length}(\text{p-temp-stk}(\text{lr-eval}(\text{'list}, \text{lr-set-pos}(l, \text{pos}), c))))$
 $= (1 + (1 + \text{length}(\text{p-temp-stk}(l))))$
 $\rightarrow (\text{length}(\text{cddr}(\text{p-temp-stk}(\text{lr-eval}(\text{'list}, \text{lr-set-pos}(l, \text{pos}), c))))$
 $= \text{length}(\text{p-temp-stk}(l)))$

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))
  ∧ (car (lr-expr (l)) ≠ 'if)
  ∧ subrp (car (lr-expr (l)))
  ∧ (length (p-temp-stk (new-l))
    = (length (p-temp-stk (l)) + arity (car (lr-expr (l))))))
  ∧ 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 (p-run-subr (car (lr-expr (l)),
    p-set-pc (lr->p (new-l), lr-return-pc (l))))
    = 'run)
  ∧ (p-psw (new-l) = 'run)
  ∧ (pos = dv (offset (p-pc (l)), 1)))
→ (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))
  ∧ (car (lr-expr (l)) ≠ 'if)
  ∧ subrp (car (lr-expr (l)))
  ∧ (length (p-temp-stk (new-l))
    = (length (p-temp-stk (l)) + arity (car (lr-expr (l))))))
  ∧ 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 (new-l) = 'run)
  ∧ (p-psw (lr-apply-subr (l, new-l)) = 'run)
  ∧ (pos = dv (offset (p-pc (l)), 1)))
→ (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)
    ∨ (length (formal-vars (car (programs)))
      = arity (logic-fname (name (car (programs))))))
    ∧ lr-proper-formalsp (cdr (programs))
  else t endif

```


THEOREM: length-formal-vars-lr-proper-formalsp-arity
 (definedp (*name*, *programs*)
 $\wedge$ (logic-fname (*name*) $\neq$ 'quote)
 $\wedge$ lr-proper-formalsp (*programs*)
 $\rightarrow$ (length (formal-vars (assoc (*name*, *programs*)))
 $=$ arity (logic-fname (*name*)))

THEOREM: arity-formals-not-quote
 (formals (*name*) $\wedge$ (*name* $\neq$ 'quote))
 $\rightarrow$ (arity (*name*) = length (formals (*name*)))

THEOREM: lr-proper-formalsp-lr-compile-programs
 s-programs-okp (*programs*)
 $\rightarrow$ 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*))
 $\wedge$ (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*)))
 $\wedge$ listp (p-prog-segment (*l*))
 $\wedge$ (user-fname (car (lr-expr (*l*))) = caar (p-prog-segment (*l*)))
 $\wedge$ good-posp1 (offset (p-pc (*l*), program-body (p-current-program (*l*)))
 $\rightarrow$ ($\neg$ lr-programs-properp (*l*, *table*))

THEOREM: length-p-temp-stk-lr-funcall
 (listp (lr-expr (*l*))
 $\wedge$ (car (lr-expr (*l*)) $\neq$ 'if)
 $\wedge$ (car (lr-expr (*l*)) $\neq$ 'quote)
 $\wedge$ (p-psw (*new-l*) = 'run)
 $\wedge$ ($\neg$ subrp (car (lr-expr (*l*))))
 $\wedge$ litatom (car (lr-expr (*l*)))
 $\wedge$ (length (p-temp-stk (lr-eval (*t*, lr-funcall (*l*, *new-l*), *c* - 1)))
 $=$ (1 + length (p-temp-stk (lr-funcall (*l*, *new-l*))))
 $\wedge$ (length (p-temp-stk (*new-l*))
 $=$ (length (p-temp-stk (*l*)) + arity (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-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*))
 ∧ good-posp (*flag*, offset (p-pc (*l*)), program-body (p-current-program (*l*)))
 ∧ lr-programs-properp (*l*, *table1*)
 ∧ (p-psw (lr-eval (*flag*, *l*, *c*)) = 'run)
 ∧ lr-proper-heapp (p-data-segment (*l*))
 ∧ lr-s-similar-const-table (*table2*, p-data-segment (*l*)))
 → 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*))
 ∧ (¬ 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*)))
 ∧ proper-p-statep (lr->p (s->lr1 (*s*, *l*, *table*)))
 ∧ lr-programs-properp (s->lr1 (*s*, *l*, *table*), *table*)
 ∧ s-good-statep (*s*, *c*)
 ∧ (p-psw (lr-funcall (s->lr1 (*s*, *l*, *table*), *lr-eval*)) = 'run)
 ∧ (p-psw (*lr-eval*) = 'run)
 ∧ lr-check-result ('list,
 values,
 p-temp-stk (*lr-eval*),
 p-data-segment (*lr-eval*),
 p-temp-stk (*l*))
 ∧ lr-proper-heapp (p-data-segment (*l*))
 ∧ (s-err-flag (s-eval ('list, s-set-pos (*s*, *pos*), *c*)) = 'run)
 ∧ (*formals* = s-formals (assoc (user-fname (car (s-expr (*s*))),
 s-progs (*s*))))
 ∧ (*lr-eval* = lr-eval ('list,

$$\begin{aligned}
& \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), \text{pos}), \\
& \quad c)) \\
\wedge & \quad (\text{values} = \text{s-ans}(\text{s-eval}(' \text{list}, \text{s-set-pos}(s, \text{pos}), c))) \\
\wedge & \quad \text{lr-s-similar-statesp}(\text{s-params}(s), \\
& \quad \text{s-temps}(s), \\
& \quad \text{s->lr1}(s, l, \text{table}), \\
& \quad \text{table})) \\
\rightarrow & \quad \text{lr-s-similar-statesp}(\text{pairlist}(\text{formals}, \text{values}), \\
& \quad \text{make-temps-entries}(\text{s-temp-list}(\text{assoc}(\text{user-fname}(\text{car}(\text{s-expr}(s))), \\
& \quad \text{s-progs}(s)))), \\
& \quad \text{lr-funcall}(\text{s->lr1}(s, l, \text{table}), \text{lr-eval}), \\
& \quad \text{table}) \text{ endlet}
\end{aligned}$$

THEOREM: lr-params-lr-set-expr-lr-pop-cstk

$$\begin{aligned}
& ((\text{area-name}(\text{p-pc}(l)) = \text{area-name}(\text{p-pc}(\text{new-l}))) \\
& \wedge (\text{p-prog-segment}(l) = \text{p-prog-segment}(\text{new-l}))) \\
\rightarrow & \quad (\text{lr-params}(\text{frame}, \\
& \quad \text{lr-set-expr}(\text{lr-pop-cstk}(\text{lr-eval}(\mathbf{t}, \text{lr-funcall}(l, \text{new-l}), c)), \\
& \quad \quad l, \\
& \quad \quad \text{pos})) \\
& = \quad \text{lr-params}(\text{frame}, l))
\end{aligned}$$

THEOREM: lr-temps-lr-set-expr-lr-pop-cstk

$$\begin{aligned}
& ((\text{area-name}(\text{p-pc}(l)) = \text{area-name}(\text{p-pc}(\text{new-l}))) \\
& \wedge (\text{p-prog-segment}(l) = \text{p-prog-segment}(\text{new-l}))) \\
\rightarrow & \quad (\text{lr-temps}(\text{frame}, \\
& \quad \text{lr-set-expr}(\text{lr-pop-cstk}(\text{lr-eval}(\mathbf{t}, \text{lr-funcall}(l, \text{new-l}), c)), \\
& \quad \quad l, \\
& \quad \quad \text{pos})) \\
& = \quad \text{lr-temps}(\text{frame}, l))
\end{aligned}$$

THEOREM: lr-eval-preserves-lr-s-similar-params

$$\begin{aligned}
& (\text{proper-p-statep}(\text{lr->p}(l)) \\
& \wedge \quad \text{good-posp}(\text{flag}, \text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l))) \\
& \wedge \quad \text{lr-programs-properp}(l, \text{table}) \\
& \wedge \quad (\text{p-psw}(\text{lr-eval}(\text{flag}, l, c)) = ' \text{run}) \\
& \wedge \quad \text{lr-proper-heapp}(\text{p-data-segment}(l)) \\
& \wedge \quad \text{lr-s-similar-params}(s\text{-params}, \text{lr-params}, \text{p-data-segment}(l))) \\
\rightarrow & \quad \text{lr-s-similar-params}(s\text{-params}, \\
& \quad \text{lr-params}, \\
& \quad \text{p-data-segment}(\text{lr-eval}(\text{flag}, l, c)))
\end{aligned}$$

THEOREM: lr-eval-preserves-lr-s-similar-temps

$$\begin{aligned}
& (\text{proper-p-statep}(\text{lr->p}(l)) \\
& \wedge \quad \text{good-posp}(\text{flag}, \text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))
\end{aligned}$$

$\wedge$ lr-programs-properp (l , $table$)
 $\wedge$ (p-psw (lr-eval ($flag$, l , c)) = 'run)
 $\wedge$ lr-proper-heapp (p-data-segment (l))
 $\wedge$ 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*, $table$)
 $\wedge$ 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)))
 $\wedge$ lr-programs-properp (s->lr1 (s , l , $table$), $table$)
 $\wedge$ lr-proper-heapp (p-data-segment (*lr-eval*))
 $\wedge$ (pos = dv (s-pos (s), 1))
 $\wedge$ (p-psw (lr-eval (t , *funcall*, c - 1)) = 'run)
 $\wedge$ (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 ($list$) $\not\leq n$) $\rightarrow$ (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

```

(listp (s-expr (s))
  ^ (car (s-expr (s)) ≠ 'if)
  ^ (car (s-expr (s)) ≠ 'quote)
  ^ (¬ subrp (car (s-expr (s))))
  ^ litatom (car (s-expr (s)))
  ^ lr-check-result ('list,
                    s-ans (s-eval),
                    p-temp-stk (new-l),
                    p-data-segment (new-l),
                    p-temp-stk (l))
  ^ 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-funcall (s->lr1 (s, l, table), new-l)) = 'run)
  ^ (s-err-flag (s-eval) = 'run)
  ^ lr-check-result (t,
                    s-ans (s-eval (t,
                                s-fun-call-state (s-eval,
                                                    car (s-expr (s))),
                                c - 1)),
                    p-temp-stk (lr-eval (t,
                                lr-funcall (s->lr1 (s,
                                                    l,
                                                    table),
                                                    new-l),
                                c - 1)),
                    p-data-segment (lr-eval (t,
                                lr-funcall (s->lr1 (s,
                                                    l,
                                                    table),
                                                    new-l),
                                c - 1)),
                    p-temp-stk (lr-funcall (s->lr1 (s, l, table), new-l))))
→ lr-check-result (t,
                  s-ans (s-eval (t,
                                s-fun-call-state (s-eval,
                                                    car (s-expr (s))),
                                c - 1)),
                  p-temp-stk (lr-eval (t,
                                lr-funcall (s->lr1 (s, l, table),
                                                    new-l),
                                c - 1)),

```

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*)))
 ∧ listp (s-expr-list (*s*))
 ∧ listp (s-pos (*s*))
 ∧ (s-err-flag (*s*) = 'run))
 → (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*)))
 ∧ (¬ listp (s-expr-list (*s*)))
 ∧ listp (s-pos (*s*))
 ∧ (s-err-flag (*s*) = 'run))
 → (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*)
 → 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*))
 ∧ good-posp1 (*pos*, program-body (p-current-program (*l*)))
 ∧ lr-programs-properp (*l*, *table*)
 ∧ (p-psw (lr-if-ok (lr-eval (**t**, lr-set-pos (*l*, *pos*), *c*))) = 'run))
 → (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))
 $\wedge$ 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))
 $\rightarrow$ 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) $\rightarrow$ (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)))
 $\wedge$ 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)
 $\wedge$ 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))
 $\wedge$ (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)),
        caddr (list1) + caddr (list2))

```

EVENT: Disable max-r.

DEFINITION:

```

s-add-temp-r (list, n)
= list (n + car (list), cadr (list), caddr (list), caddr (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  $\simeq$  0 then list (0, 0, 0, 0)
elseif litatom (s-expr (s)) then list (1, 0, 0, 0)
elseif s-expr (s)  $\simeq$  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),
    caddr (arg-r))) endlet endlet endlet
else list (0, 0, 0, 0) endif

```

DEFINITION: $\text{s-eval-temp-r} (flag, s, c) = \text{car} (\text{s-eval-r} (flag, s, c))$

DEFINITION: $\text{s-eval-ctrl-r} (flag, s, c) = \text{cadr} (\text{s-eval-r} (flag, s, c))$

DEFINITION: $\text{s-eval-ws-r} (flag, s, c) = \text{caddr} (\text{s-eval-r} (flag, s, c))$

DEFINITION:

$\text{s-eval-heap-r} (flag, s, c) = \text{caddr} (\text{s-eval-r} (flag, s, c))$

DEFINITION:

S-MAX-SUBR-REQS

$= \max (\log (2, \text{LR-CONS-TAG}),$

$$\max(\log(2, \text{LR-TRUE-TAG}), \max(\log(2, \text{LR-CDR-OFFSET}), \log(2, \text{LR-CAR-OFFSET})))$$

EVENT: Disable s-max-subr-reqs.

THEOREM: numberp-car-cadr-caddr-cadddr-s-apply-subr-r

$$\begin{aligned} & (\text{car}(\text{s-apply-subr-r}(subr, s)) \in \mathbf{N}) \\ \wedge & \quad (\text{cadr}(\text{s-apply-subr-r}(subr, s)) \in \mathbf{N}) \\ \wedge & \quad (\text{caddr}(\text{s-apply-subr-r}(subr, s)) \in \mathbf{N}) \\ \wedge & \quad (\text{caddrd}(\text{s-apply-subr-r}(subr, s)) \in \mathbf{N}) \end{aligned}$$

EVENT: Disable s-apply-subr-r.

THEOREM: numberp-max-r

$$\begin{aligned} & (\text{car}(\text{max-r}(list1, list2)) \in \mathbf{N}) \\ \wedge & \quad (\text{cadr}(\text{max-r}(list1, list2)) \in \mathbf{N}) \\ \wedge & \quad (\text{caddr}(\text{max-r}(list1, list2)) \in \mathbf{N}) \\ \wedge & \quad (\text{caddr}(\text{max-r}(list1, list2)) \in \mathbf{N}) \end{aligned}$$

THEOREM: numberp-s-eval-temp-ctrl-ws-heap-r

$$\begin{aligned} & (\text{s-eval-temp-r}(flag, s, c) \in \mathbf{N}) \\ \wedge & \quad (\text{s-eval-ctrl-r}(flag, s, c) \in \mathbf{N}) \\ \wedge & \quad (\text{s-eval-ws-r}(flag, s, c) \in \mathbf{N}) \\ \wedge & \quad (\text{s-eval-heap-r}(flag, s, c) \in \mathbf{N}) \end{aligned}$$

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 ∈ 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:

$$\begin{aligned}
& \text{lr-check-resourcesp}(flag, s, l, c) \\
= & ((\text{p-max-temp-stk-size}(l) \\
& \quad \not\leq (\text{length}(\text{p-temp-stk}(l)) + \text{s-eval-temp-r}(flag, s, c))) \\
& \wedge (\text{p-max-ctrl-stk-size}(l) \\
& \quad \not\leq (\text{p-ctrl-stk-size}(\text{p-ctrl-stk}(l)) \\
& \quad \quad + \text{s-eval-ctrl-r}(flag, s, c))) \\
& \wedge (\text{p-word-size}(l) \not\leq \text{s-eval-ws-r}(flag, s, c)) \\
& \wedge (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{p-data-segment}(l)), \\
& \quad \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{p-data-segment}(l)), \\
& \quad \quad \quad \text{p-data-segment}(l)), \\
& \quad \quad \text{p-data-segment}(l)) \\
& \quad \not\leq \text{s-eval-heap-r}(flag, s, c)))
\end{aligned}$$

EVENT: Disable lr-check-resourcesp.

THEOREM: not-lessp-max-r-car

$$\begin{aligned}
& (\text{car}(\text{max-r}(list1, list2)) \not\leq \text{car}(list1)) \\
& \wedge (\text{car}(\text{max-r}(list1, list2)) \not\leq \text{car}(list2))
\end{aligned}$$

THEOREM: not-lessp-max-r-cadr

$$\begin{aligned}
& (\text{cadr}(\text{max-r}(list1, list2)) \not\leq \text{cadr}(list1)) \\
& \wedge (\text{cadr}(\text{max-r}(list1, list2)) \not\leq \text{cadr}(list2))
\end{aligned}$$

THEOREM: not-lessp-max-r-caddr

$$\begin{aligned}
& (\text{caddr}(\text{max-r}(list1, list2)) \not\leq \text{caddr}(list1)) \\
& \wedge (\text{caddr}(\text{max-r}(list1, list2)) \not\leq \text{caddr}(list2))
\end{aligned}$$

THEOREM: not-lessp-max-r-caddr

$$\begin{aligned}
& (\text{caddr}(\text{max-r}(list1, list2)) \not\leq \text{caddr}(list1)) \\
& \wedge (\text{caddr}(\text{max-r}(list1, list2)) \not\leq \text{caddr}(list2))
\end{aligned}$$

THEOREM: lr-check-resourcesp-listp-s-expr-list

$$\begin{aligned}
& ((\text{s-err-flag}(s) = \text{'run}) \\
& \wedge \text{listp}(\text{s-pos}(s)) \\
& \wedge \text{listp}(\text{s-expr-list}(s)) \\
& \wedge \text{good-posp}(\text{'list}, \text{s-pos}(s), \text{s-body}(\text{s-prog}(s))) \\
& \wedge \text{lr-check-resourcesp}(\text{'list}, s, l, c)) \\
& \rightarrow \text{lr-check-resourcesp}(\mathbf{t}, s, l, c)
\end{aligned}$$

THEOREM: lr-eval-preserves-lr-proper-heapp

$$\begin{aligned}
& (\text{proper-p-statep}(\text{lr->p}(s->\text{lr1}(s, l, \text{table}))) \\
& \wedge \text{lr-proper-heapp}(\text{p-data-segment}(l)) \\
& \wedge \text{good-posp}(flag, \text{s-pos}(s), \text{s-body}(\text{s-prog}(s))) \\
& \wedge \text{lr-programs-properp}(s->\text{lr1}(s, l, \text{table}), \text{table})
\end{aligned}$$

THEOREM: lr-eval-s->lr1-preserves-p-ctrl-stk-size
 (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))
 $\rightarrow$ (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*)))
 $\wedge$ good-posp (*flag*, 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 (*flag*, s->lr1 (s, l, *table*), c)) = 'run)
 $\wedge$ (*flag* $\neq$ 'list))
 $\rightarrow$ (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*)))
 $\wedge$ lr-proper-heapp (p-data-segment (l))
 $\wedge$ good-posp (*flag*, pos, 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*, lr-set-pos (s->lr1 (s, l, *table*), pos), c)) = 'run))
 $\rightarrow$ 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*)))
 $\wedge$ lr-proper-heapp (p-data-segment (l))
 $\wedge$ good-posp (*flag*, pos, 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*, lr-set-pos (s->lr1 (s, l, *table*), pos), c)) = 'run))
 $\rightarrow$ 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

$(\text{proper-p-statep} (\text{lr->p} (\text{s->lr1} (s, l, \text{table}))))$
 $\wedge \text{lr-proper-heapp} (\text{p-data-segment} (l))$
 $\wedge \text{good-posp1} (pos, \text{s-body} (\text{s-prog} (s)))$
 $\wedge \text{lr-programs-properp} (\text{s->lr1} (s, l, \text{table}), \text{table})$
 $\wedge \text{lr-s-similar-statesp} (\text{s-params} (s), \text{s-temps} (s), \text{s->lr1} (s, l, \text{table}), \text{table})$
 $\wedge \text{s-good-statep} (s, c)$
 $\wedge (\text{p-psw} (\text{lr-eval} (\mathbf{t}, \text{lr-set-pos} (\text{s->lr1} (s, l, \text{table}), pos), c)) = \text{'run})$
 $\wedge (\neg \text{s-ans} (\text{s-eval} (\mathbf{t}, \text{s-set-pos} (s, pos), c))))$
 $\rightarrow (\text{car} (\text{p-temp-stk} (\text{lr-eval} (\mathbf{t}, \text{lr-set-pos} (\text{s->lr1} (s, l, \text{table}), pos), c)))$
 $\quad = \text{LR-F-ADDR})$

THEOREM: $\text{lr-eval-s-eval-flag-t-s-ans-non-f-lr-set-pos}$

$(\text{proper-p-statep} (\text{lr->p} (\text{s->lr1} (s, l, \text{table}))))$
 $\wedge \text{lr-proper-heapp} (\text{p-data-segment} (l))$
 $\wedge \text{good-posp1} (pos, \text{s-body} (\text{s-prog} (s)))$
 $\wedge \text{lr-programs-properp} (\text{s->lr1} (s, l, \text{table}), \text{table})$
 $\wedge \text{lr-s-similar-statesp} (\text{s-params} (s), \text{s-temps} (s), \text{s->lr1} (s, l, \text{table}), \text{table})$
 $\wedge \text{s-good-statep} (s, c)$
 $\wedge (\text{p-psw} (\text{lr-eval} (\mathbf{t}, \text{lr-set-pos} (\text{s->lr1} (s, l, \text{table}), pos), c)) = \text{'run})$
 $\wedge \text{s-ans} (\text{s-eval} (\mathbf{t}, \text{s-set-pos} (s, pos), c)))$
 $\rightarrow (\text{car} (\text{p-temp-stk} (\text{lr-eval} (\mathbf{t}, \text{lr-set-pos} (\text{s->lr1} (s, l, \text{table}), pos), c)))$
 $\quad \neq \text{identity} (\text{LR-F-ADDR}))$

THEOREM: $\text{subsetp-not-member-both}$

$((\text{addr} \notin \text{set2}) \wedge \text{subsetp} (\text{set1}, \text{set2})) \rightarrow (\text{addr} \notin \text{set1})$

THEOREM: $\text{lr-count-free-nodes-deposit-free-ptr}$

$(\text{adpp} (\text{'(free-ptr . 0)}, \text{data-seg}) \wedge \text{lr-node-listp} (\text{node-list}, \text{data-seg}))$
 $\rightarrow (\text{lr-count-free-nodes} (\text{addr},$
 $\quad \text{node-list},$
 $\quad \text{deposit} (\text{anything}, \text{identity} (\text{LR-FP-ADDR}), \text{data-seg}))$
 $\quad = \text{lr-count-free-nodes} (\text{addr}, \text{node-list}, \text{data-seg}))$

THEOREM: $\text{lr-count-free-nodes-deposit-non-ref-count}$

$(\text{lr-noddep} (\text{addr2}, \text{data-seg}))$
 $\wedge (\text{offset} \neq \text{LR-REF-COUNT-OFFSET})$
 $\wedge (\text{offset} \in \mathbf{N})$
 $\wedge (\text{offset} < \text{LR-NODE-SIZE})$
 $\wedge \text{lr-node-listp} (\text{node-list}, \text{data-seg}))$
 $\rightarrow (\text{lr-count-free-nodes} (\text{addr1},$
 $\quad \text{node-list},$
 $\quad \text{deposit} (\text{anything}, \text{add-addr} (\text{addr2}, \text{offset}), \text{data-seg}))$
 $\quad = \text{lr-count-free-nodes} (\text{addr1}, \text{node-list}, \text{data-seg}))$

THEOREM: $\text{lr-count-free-nodes-deposit-lr-noddep}$

$$\begin{aligned}
& (\text{lr-nodep}(\text{addr2}, \text{data-seg}) \wedge \text{lr-node-listp}(\text{node-list}, \text{data-seg})) \\
\rightarrow & (\text{lr-count-free-nodes}(\text{addr1}, \text{node-list}, \text{deposit}(\text{anything}, \text{addr2}, \text{data-seg})) \\
& = \text{lr-count-free-nodes}(\text{addr1}, \text{node-list}, \text{data-seg}))
\end{aligned}$$

THEOREM: lr-count-free-nodes-delete-deposit

$$\begin{aligned}
& ((\text{addr1} \notin \text{node-list}) \\
& \wedge \text{lr-nodep}(\text{addr1}, \text{data-seg}) \\
& \wedge \text{lr-node-listp}(\text{node-list}, \text{data-seg}) \\
& \wedge (\text{type}(\text{ref-count}) = \text{'nat'}) \\
& \wedge (\text{untag}(\text{ref-count}) \in \mathbf{N})) \\
\rightarrow & (\text{lr-count-free-nodes}(\text{addr2}, \\
& \quad \text{node-list}, \\
& \quad \text{deposit}(\text{ref-count}, \\
& \quad \quad \text{add-addr}(\text{addr1}, \\
& \quad \quad \quad \text{identity}(\text{LR-REF-COUNT-OFFSET})), \\
& \quad \quad \text{data-seg})) \\
& = \text{lr-count-free-nodes}(\text{addr2}, \text{node-list}, \text{data-seg}))
\end{aligned}$$

THEOREM: lr-count-free-nodes-max-addr-lr-free-list-nodes

$$\begin{aligned}
& \text{lr-count-free-nodes}(\text{max-addr}, \\
& \quad \text{lr-free-list-nodes}(\text{max-addr}, \text{data-seg1}), \\
& \quad \text{data-seg2}) \\
& = 0
\end{aligned}$$

THEOREM: lr-count-lr-free-list-nodes-p-run-cons

$$\begin{aligned}
& \text{let } dds \text{ be } \text{deposit-a-list}(\text{list}(\text{identity}(\text{tag}(\text{'nat'}, 5)), \\
& \quad \text{ref-count}, \\
& \quad \text{any1}, \\
& \quad \text{any2}), \\
& \quad \text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \text{data-seg}), \\
& \quad \text{data-seg}) \\
& \text{in} \\
& (\text{lr-proper-heapp}(\text{data-seg}) \\
& \wedge (\text{max-addr} = \text{lr-max-node}(\text{data-seg})) \\
& \wedge (\text{type}(\text{ref-count}) = \text{'nat'}) \\
& \wedge (\text{untag}(\text{ref-count}) \in \mathbf{N}) \\
& \wedge (\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \text{data-seg}) \neq \text{max-addr})) \\
\rightarrow & ((1 + \text{lr-count-free-nodes}(\text{fetch}(\text{add-addr}(\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \\
& \quad \text{data-seg}), \\
& \quad \quad \text{identity}(\text{LR-REF-COUNT-OFFSET})), \\
& \quad \quad \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{max-addr}, dds), \\
& \quad dds)) \\
& = \text{lr-count-free-nodes}(\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \\
& \quad \text{data-seg}),
\end{aligned}$$

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)
 ∧ proper-p-statep (lr->p (*new-l*))
 ∧ proper-p-statep (p-set-pc (lr->p (*new-l*), *pc*))
 ∧ (*max-node* = lr-max-node (p-data-segment (*new-l*)))
 ∧ (p-psw (*new-l*) = 'run)
 ∧ lr-programs-properp (*new-l*, *table*)
 ∧ lr-proper-heapp (p-data-segment (*new-l*))
 ∧ (unlabel (get (offset (*pc*),
 program-body (assoc (area-name (*pc*),
 p-prog-segment (lr->p (*new-l*))))))
 = '(call cons)))
 → (fetch (identity (LR-FP-ADDR), p-data-segment (*new-l*)) ≠ *max-node*)

THEOREM: get-comp-body-lr-compile-programs
 (good-pospl (s-pos (*s*), s-body (s-prog (*s*)))
 ∧ lr-programs-properp (s->lr1 (*s*, *l*, *table*), *table*)
 ∧ (car (s-expr (*s*)) ≠ 'if)
 ∧ (car (s-expr (*s*)) ≠ S-TEMP-EVAL)
 ∧ (car (s-expr (*s*)) ≠ S-TEMP-TEST)
 ∧ (car (s-expr (*s*)) ≠ S-TEMP-FETCH)
 ∧ (car (s-expr (*s*)) ≠ 'quote)
 ∧ listp (s-expr (*s*)))
 → (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))))
   + caddr (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)

```

```

^ (p-psw (lr-apply-subr (s->lr1 (s, l, table), new-l)) = 'run)
^ s-good-statep (s, c)
^ lr-proper-heapp (p-data-segment (l))
^ lr-s-similar-statesp (s-params (s),
                        s-temps (s),
                        s->lr1 (s, l, table),
                        table)
^ (pos = dv (s-pos (s), 1))
^ (max-addr = lr-max-node (p-data-segment (l)))
^ (s-eval-size = s-eval-heap-r ('list, s-set-pos (s, pos), c))
^ ((lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment (new-l)),
                               lr-free-list-nodes (max-addr,
                                                    p-data-segment (new-l)),
                               p-data-segment (new-l))
  + s-eval-size)
 = lr-count-free-nodes (fetch (LR-FP-ADDR,
                               p-data-segment (l),
                               lr-free-list-nodes (max-addr,
                                                    p-data-segment (l)),
                               p-data-segment (l))))
→ ((s-eval-size
  + caddr (s-apply-subr-r (car (s-expr (s)),
                          s-eval ('list,
                                  s-set-pos (s, pos),
                                  c)))
  + lr-count-free-nodes (fetch (identity (LR-FP-ADDR),
                                p-data-segment (lr-apply-subr (s->lr1 (s,
                                                                    l,
                                                                    table),
                                                                    new-l))),
                          lr-free-list-nodes (max-addr,
                                              p-data-segment (lr-apply-subr (s->lr1 (s,
                                                                    l,
                                                                    table),
                                                                    new-l))),
                          p-data-segment (lr-apply-subr (s->lr1 (s,
                                                                    l,
                                                                    table),
                                                                    new-l))))))
 = 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))) endlet

```

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)))
   ^ 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)
   ^ lr-s-similar-statep (s-params (s),
                          s-temps (s),
                          s->lr1 (s, l, table),
                          table)
   ^ s-good-statep (s, c)
   ^ (p-psw (lr-eval) = 'run)
   ^ listp (s-expr (s))
   ^ (car (s-expr (s)) ≠ 'if)
   ^ (car (s-expr (s)) ≠ S-TEMP-EVAL)
   ^ (car (s-expr (s)) ≠ S-TEMP-TEST)
   ^ (car (s-expr (s)) ≠ S-TEMP-FETCH)
   ^ (car (s-expr (s)) ≠ 'quote))
  → 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: caddr-max-r

caddr (max-r (*list1*, *list2*)) = (caddr (*list1*) + caddr (*list2*))

THEOREM: lr-eval-s-eval-heap-r-lr-count-lr-free-list-nodes

```

  (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-statep (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)
   ^ (s-err-flag (s-eval (flag, s, c)) = 'run))
  → ((lr-count-free-nodes (fetch (LR-FP-ADDR,
                                   p-data-segment (lr-eval (flag,
                                                             s->lr1 (s, l, table),

```

$$\begin{aligned}
& \text{lr-free-list-nodes}(\text{lr-max-node}(\text{p-data-segment}(l)), \\
& \quad \text{p-data-segment}(\text{lr-eval}(flag, \\
& \quad \quad s \rightarrow \text{lr1}(s, \\
& \quad \quad \quad l, \\
& \quad \quad \quad table), \\
& \quad \quad \quad c))), \\
& \quad \text{p-data-segment}(\text{lr-eval}(flag, s \rightarrow \text{lr1}(s, l, table), c))) \\
& + \text{s-eval-heap-r}(flag, s, c)) \\
= & \text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{p-data-segment}(l)), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{p-data-segment}(l)), \\
& \quad \quad \text{p-data-segment}(l)), \\
& \quad \text{p-data-segment}(l)))
\end{aligned}$$

THEOREM: lr-check-resourcesp-list-set-expr-nx

$$\begin{aligned}
& (\text{listp}(\text{s-pos}(s))) \\
& \wedge \text{listp}(\text{s-expr-list}(s)) \\
& \wedge \text{good-posp}('list, \text{s-pos}(s), \text{s-body}(\text{s-prog}(s))) \\
& \wedge \text{proper-p-statep}(\text{lr} \rightarrow \text{p}(\text{s} \rightarrow \text{lr1}(s, l, table))) \\
& \wedge \text{lr-proper-heapp}(\text{p-data-segment}(l)) \\
& \wedge \text{lr-programs-properp}(\text{s} \rightarrow \text{lr1}(s, l, table), table) \\
& \wedge \text{lr-s-similar-statesp}(\text{s-params}(s), \text{s-temps}(s), \text{s} \rightarrow \text{lr1}(s, l, table), table) \\
& \wedge \text{s-good-statep}(s, c) \\
& \wedge (\text{p-psw}(\text{lr-eval}(\mathbf{t}, \text{s} \rightarrow \text{lr1}(s, l, table), c)) = 'run) \\
& \wedge (\text{s-err-flag}(\text{s-eval}(\mathbf{t}, s, c)) = 'run) \\
& \wedge \text{lr-check-resourcesp}('list, s, l, c)) \\
\rightarrow & \text{lr-check-resourcesp}('list, \\
& \quad \text{s-set-expr}(\text{s-eval}(\mathbf{t}, s, c), s, \text{nx}(\text{s-pos}(s))), \\
& \quad \text{lr-eval}(\mathbf{t}, \text{s} \rightarrow \text{lr1}(s, l, table), c), \\
& \quad c)
\end{aligned}$$

THEOREM: lr-check-resourcesp-lr-push-tstk-flag-run

$$\begin{aligned}
& (\text{lr-check-resourcesp}(flag, s, l, c)) \\
& \wedge (flag \neq 'list) \\
& \wedge \text{litatom}(\text{s-expr}(s)) \\
& \wedge (c \neq 0) \\
& \wedge (\text{s-err-flag}(s) = 'run) \\
\rightarrow & (\text{p-psw}(\text{lr-push-tstk}(\text{s} \rightarrow \text{lr1}(s, l, table), \\
& \quad \text{cdr}(\text{assoc}(\text{s-expr}(s), \text{bindings}(\text{car}(\text{p-ctrl-stk}(l))))))) \\
& = 'run)
\end{aligned}$$

THEOREM: lr-check-resourcesp-s-set-pos-if-cadr

$$\begin{aligned}
& (\text{lr-check-resourcesp}(flag, s, l, c)) \\
& \wedge \text{s-good-statep}(s, c) \\
& \wedge (flag \neq 'list)
\end{aligned}$$

$\wedge (c \neq 0)$
 $\wedge \text{listp}(\text{s-expr}(s))$
 $\wedge (\text{car}(\text{s-expr}(s)) = \text{'if})$
 $\rightarrow \text{lr-check-resourcesp}(\mathbf{t}, \text{s-set-pos}(s, \text{dv}(\text{s-pos}(s), 1)), l, c)$

THEOREM: s-eval-subsetp-s-collect-temp-alist-s-set-pos-if

$(\text{listp}(\text{s-expr}(s))$
 $\wedge (\text{car}(\text{s-expr}(s)) = \text{'if})$
 $\wedge \text{s-good-statep}(s, c)$
 $\wedge \text{good-posp1}(\text{dv}(\text{s-pos}(s), 1), \text{s-body}(\text{s-prog}(s)))$
 $\wedge \text{s-check-temps-setp}(\text{s-temps}(s))$
 $\wedge \text{s-all-temps-setp}(\mathbf{t}, \text{cadr}(\text{s-expr}(s)), \text{temp-alist-to-set}(\text{s-temps}(s)))$
 $\wedge \text{s-all-progs-temps-setp}(\text{s-progs}(s))$
 $\wedge (\text{s-err-flag}(\text{s-eval}(\mathbf{t}, \text{s-set-pos}(s, \text{dv}(\text{s-pos}(s), 1)), c)) = \text{'run}))$
 $\rightarrow \text{subsetp}(\text{s-collect-all-temps}(\mathbf{t}, \text{cadr}(\text{s-expr}(s))),$
 $\text{temp-alist-to-set}(\text{s-temps}(\text{s-eval}(\mathbf{t},$
 $\text{s-set-pos}(s, \text{dv}(\text{s-pos}(s), 1)),$
 $c))))$

THEOREM: length-p-temp-stk-lr-pop-tstk-lr-eval-flag-t

$(\text{proper-p-statep}(\text{lr->p}(\text{s->lr1}(s, l, \text{table}))))$
 $\wedge \text{good-posp1}(pos, \text{s-body}(\text{s-prog}(s)))$
 $\wedge \text{lr-programs-properp}(\text{s->lr1}(s, l, \text{table}), \text{table})$
 $\wedge \text{s-good-statep}(s, c)$
 $\wedge (\text{p-psw}(\text{lr-if-ok}(\text{lr-eval}(\mathbf{t}, \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), pos), c)))$
 $= \text{'run}))$
 $\rightarrow (\text{length}(\text{p-temp-stk}(\text{lr-pop-tstk}(\text{lr-if-ok}(\text{lr-eval}(\mathbf{t},$
 $\text{lr-set-pos}(\text{s->lr1}(s,$
 $l,$
 $\text{table}),$
 $pos),$
 $c))))))$
 $= \text{length}(\text{p-temp-stk}(l)))$

THEOREM: lr-eval-s->lr1-preserves-p-ctrl-stk-size-lr-set-pos

$(\text{proper-p-statep}(\text{lr->p}(l)))$
 $\wedge \text{good-posp}(\text{flag}, pos, \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{lr-programs-properp}(l, \text{table})$
 $\wedge (\text{p-psw}(\text{lr-eval}(\text{flag}, \text{lr-set-pos}(l, pos), c)) = \text{'run}))$
 $\rightarrow (\text{p-ctrl-stk-size}(\text{p-ctrl-stk}(\text{lr-eval}(\text{flag}, \text{lr-set-pos}(l, pos), c)))$
 $= \text{p-ctrl-stk-size}(\text{p-ctrl-stk}(l)))$

THEOREM: lr-check-resourcesp-lr-pop-tstk-lr-eval-1

let *lr-eval* **be** $\text{lr-if-ok}(\text{lr-eval}(\mathbf{t}, \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), pos), c))$
in


```

((c ≠ 0)
 ∧ listp (s-expr (s))
 ∧ (car (s-expr (s)) = 'if)
 ∧ proper-p-statep (lr->p (s->lr1 (s, l, table)))
 ∧ good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧ 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) = 'run)
 ∧ (pos = dv (s-pos (s), 1))
 ∧ (s-err-flag (s-eval (t, s-set-pos (s, pos), c)) = 'run)
 ∧ s-ans (s-eval (t, s-set-pos (s, dv (s-pos (s), 1)), c))
 ∧ lr-check-resourcesp (flag, s, l, c)
 ∧ (flag ≠ 'list))
→ lr-check-resourcesp (t,
                      s-set-expr (s-eval (t, s-set-pos (s, pos), c),
                                   s,
                                   dv (s-pos (s), 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 ≠ 0)
 ∧ listp (s-expr (s))
 ∧ (car (s-expr (s)) = 'if)
 ∧ proper-p-statep (lr->p (s->lr1 (s, l, table)))
 ∧ good-posp1 (s-pos (s), s-body (s-prog (s)))
 ∧ 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) = 'run)
 ∧ (pos = dv (s-pos (s), 1))
 ∧ (s-err-flag (s-eval (t, s-set-pos (s, pos), c)) = 'run)
 ∧ (¬ s-ans (s-eval (t, s-set-pos (s, dv (s-pos (s), 1)), c)))

```

$\wedge$ lr-check-resourcesp ($flag, s, l, c$)
 $\wedge$ ($flag \neq \text{'list'}$)
 $\rightarrow$ lr-check-resourcesp ($\mathbf{t}$,
 $\quad$ s-set-expr (s-eval ($\mathbf{t}$, s-set-pos (s, pos), c),
 $\quad$ s ,
 $\quad$ dv (s-pos (s), $\mathbf{3}$)),
 $\quad$ lr-pop-tstk ($lr\text{-eval}$),
 $\quad$ c) **endlet**

THEOREM: lr-check-resourcesp-s-temp-eval

$((c \neq 0)$
 $\wedge$ listp (s-expr (s))
 $\wedge$ (car (s-expr (s)) = S-TEMP-EVAL)
 $\wedge$ good-posp1 (s-pos (s), s-body (s-prog (s)))
 $\wedge$ ($pos = \text{dv (s-pos (s), 1)}$)
 $\wedge$ lr-check-resourcesp ($flag, s, l, c$)
 $\wedge$ ($flag \neq \text{'list'}$)
 $\rightarrow$ lr-check-resourcesp ($\mathbf{t}$, s-set-pos (s, pos), l, c)

THEOREM: lr-check-resourcesp-s-temp-test

$((c \neq 0)$
 $\wedge$ 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)))
 $\wedge$ ($pos = \text{dv (s-pos (s), 1)}$)
 $\wedge$ lr-check-resourcesp ($flag, s, l, c$)
 $\wedge$ ($flag \neq \text{'list'}$)
 $\rightarrow$ lr-check-resourcesp ($\mathbf{t}$, s-set-pos (s, pos), l, c)

THEOREM: lr-do-temp-fetch-lr-check-resourcesp-temp-test

$(\text{lr-check-resourcesp (flag, s, l, c)}$
 $\wedge$ 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)
 $\quad \vee$ (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 ($s, l, table$), $table$)
 $\wedge$ ($c \neq 0$)
 $\wedge$ s-temp-setp (cadr (s-expr (s)), s-temps (s))
 $\wedge$ ($flag \neq \text{'list'}$)
 $\rightarrow$ (p-psw (lr-do-temp-fetch (s->lr1 ($s, l, table$)))) = 'run)

THEOREM: lr-push-tstk-lr-check-resourcesp-quote
 $(\text{lr-check-resourcesp}(flag, s, l, c)$
 $\wedge \text{good-pospl}(s\text{-pos}(s), s\text{-body}(s\text{-prog}(s)))$
 $\wedge \text{listp}(s\text{-expr}(s))$
 $\wedge (\text{car}(s\text{-expr}(s)) = \text{'quote})$
 $\wedge s\text{-good-statep}(s, c)$
 $\wedge (c \neq 0)$
 $\wedge (flag \neq \text{'list}))$
 $\rightarrow (\text{p-psw}(\text{lr-push-tstk}(s\text{->lr1}(s, l, table),$
 $\text{cadr}(\text{lr-expr}(s\text{->lr1}(s, l, table))))))$
 $= \text{'run})$

THEOREM: lr-check-resourcesp-funcall
 $((c \neq 0)$
 $\wedge \text{listp}(s\text{-expr}(s))$
 $\wedge (\text{car}(s\text{-expr}(s)) \neq \text{'if})$
 $\wedge (\text{car}(s\text{-expr}(s)) \neq \text{S-TEMP-EVAL})$
 $\wedge (\text{car}(s\text{-expr}(s)) \neq \text{S-TEMP-TEST})$
 $\wedge (\text{car}(s\text{-expr}(s)) \neq \text{S-TEMP-FETCH})$
 $\wedge (\text{car}(s\text{-expr}(s)) \neq \text{'quote})$
 $\wedge (s\text{-err-flag}(s\text{-eval}(\text{'list}, s\text{-set-pos}(s, \text{dv}(s\text{-pos}(s), 1)), c)) = \text{'run})$
 $\wedge \text{proper-p-statep}(\text{lr->p}(s\text{->lr1}(s, l, table)))$
 $\wedge \text{lr-proper-heapp}(\text{p-data-segment}(l))$
 $\wedge \text{good-pospl}(s\text{-pos}(s), s\text{-body}(s\text{-prog}(s)))$
 $\wedge \text{lr-programs-properp}(s\text{->lr1}(s, l, table), table)$
 $\wedge s\text{-good-statep}(s, c)$
 $\wedge \text{lr-check-resourcesp}(flag, s, l, c)$
 $\wedge (flag \neq \text{'list}))$
 $\rightarrow \text{lr-check-resourcesp}(\text{'list}, s\text{-set-pos}(s, \text{dv}(s\text{-pos}(s), 1)), l, c)$

THEOREM: numberp-s-eval-temp-ctrl-ws-heap-r-opened
 $(\text{car}(s\text{-eval-r}(flag, s, c)) \in \mathbf{N})$
 $\wedge (\text{cadr}(s\text{-eval-r}(flag, s, c)) \in \mathbf{N})$
 $\wedge (\text{caddr}(s\text{-eval-r}(flag, s, c)) \in \mathbf{N})$
 $\wedge (\text{cadddr}(s\text{-eval-r}(flag, s, c)) \in \mathbf{N})$

THEOREM: lessp-1-not-zerop-exp
 $((m \neq 0) \wedge (1 < n)) \rightarrow (1 < \exp(n, m))$

THEOREM: lessp-1-not-zerop-log
 $((1 < c) \wedge (n \in \mathbf{N})) \rightarrow ((\log(c, n) < 1) = (n < 1))$

DEFINITION:
 $\text{induct-hint-18}(c, n, m)$
 $= \text{if } c < 2 \text{ then } t$

```

elseif  $n \simeq 0$  then t
elseif  $m \simeq 0$  then t
else  $\text{induct-hint-18}(c, n \div c, m - 1)$  endif

```

THEOREM: times-quotient-lessp-fact-1
 $((c \not\simeq 0) \wedge (n \in \mathbf{N}) \wedge (m \in \mathbf{N}))$
 $\rightarrow ((n < (c * m)) = ((n \div c) < m))$

THEOREM: exp-log-lessp-fact-1
 $((1 < c) \wedge (n \in \mathbf{N}) \wedge (m \in \mathbf{N}))$
 $\rightarrow ((n < \exp(c, m)) = (\log(c, n) < (1 + m)))$

EVENT: Disable times-quotient-lessp-fact-1.

THEOREM: adpp-untag-add-addr-offset-car
 $(\text{lr-good-pointerp}(addr, data-seg)$
 $\wedge \text{lr-proper-p-areasp}(data-seg)$
 $\wedge (\text{untag}(\text{fetch}(addr, data-seg)) = \text{LR-CONS-TAG})$
 $\wedge \text{lr-proper-heapp}(data-seg))$
 $\rightarrow \text{adpp}(\text{untag}(\text{add-addr}(addr, \text{identity}(\text{LR-CAR-OFFSET}))), data-seg)$

THEOREM: adpp-untag-add-addr-offset-cdr
 $(\text{lr-good-pointerp}(addr, data-seg)$
 $\wedge \text{lr-proper-p-areasp}(data-seg)$
 $\wedge (\text{untag}(\text{fetch}(addr, data-seg)) = \text{LR-CONS-TAG})$
 $\wedge \text{lr-proper-heapp}(data-seg))$
 $\rightarrow \text{adpp}(\text{untag}(\text{add-addr}(addr, \text{identity}(\text{LR-CDR-OFFSET}))), data-seg)$

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
 $\text{p-test-bool-and-jump-okp}(\text{list}(ins-name, 't, label),$
 $\text{p-state}(pc,$
 $\text{ctrl-stk},$
 $\text{cons}('(\text{bool } t), temp-stk),$
 $prog-seg,$
 $data-seg,$
 $max-ctrl,$
 $max-temp,$
 $word-size,$
 $psw))$
 $= t$

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))
    < caddr (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))

```

$$\begin{aligned}
& + \text{car}(\text{s-apply-subr-r}(' \mathbf{cdr}, \text{new-s}))) \\
\wedge & (\text{p-max-ctrl-stk-size}(\text{new-l}) \\
& \not\leq (\text{p-ctrl-stk-size}(\text{p-ctrl-stk}(\text{new-l})) \\
& + \text{cadr}(\text{s-apply-subr-r}(' \mathbf{cdr}, \text{new-s})))) \\
\wedge & (\text{p-word-size}(\text{new-l}) \\
& \not\leq \max(\text{S-MAX-SUBR-REQS}, \text{caddr}(\text{s-apply-subr-r}(' \mathbf{cdr}, \text{new-s})))) \\
\wedge & (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{p-data-segment}(\text{new-l})), \\
& \text{lr-free-list-nodes}(\text{lr-max-node}(\text{p-data-segment}(\text{new-l})), \\
& \text{p-data-segment}(\text{new-l})), \\
& \text{caddr}(\text{s-apply-subr-r}(' \mathbf{cdr}, \text{new-s})))) \\
\wedge & (\text{length}(\text{p-temp-stk}(\text{new-l})) \not\leq \text{arity}(' \mathbf{cdr})) \\
\wedge & (\text{length}(\text{s-ans}(\text{new-s})) = \text{arity}(' \mathbf{cdr})) \\
\rightarrow & (\text{p-psw}(\text{p}(\text{p-set-pc}(\text{lr->p}(\text{new-l}), \text{pc}), \\
& \text{p-cdr-clock}(\text{p-set-pc}(\text{lr->p}(\text{new-l}), \text{pc})))) \\
& = ' \mathbf{run})
\end{aligned}$$

THEOREM: lessp-plus-remainder-0-fact

$$\begin{aligned}
& (((\text{offset1} \bmod \text{max}) = 0) \\
& \wedge ((\text{offset2} \bmod \text{max}) = 0) \\
& \wedge (n < \text{max}) \\
& \wedge (\text{offset1} \in \mathbf{N}) \\
& \wedge (\text{offset2} \in \mathbf{N})) \\
\rightarrow & (((n + \text{offset1}) < \text{offset2}) = (\text{offset1} < \text{offset2}))
\end{aligned}$$

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

$$\begin{aligned}
& (\text{lr-boundary-nodep}(\text{addr1}) \\
& \wedge \text{lr-boundary-nodep}(\text{addr2}) \\
& \wedge (n < \text{LR-NODE-SIZE}) \\
& \wedge (\text{offset}(\text{addr1}) \in \mathbf{N}) \\
& \wedge (\text{offset}(\text{addr2}) \in \mathbf{N})) \\
\rightarrow & (((n + \text{offset}(\text{addr1})) < \text{offset}(\text{addr2})) \\
& = (\text{offset}(\text{addr1}) < \text{offset}(\text{addr2})))
\end{aligned}$$

THEOREM: adpp-untag-add-addr-lr-nodep-not-max-addr

$$\begin{aligned}
& (\text{adpp}(\text{untag}(\text{addr}), \text{data-seg}) \\
& \wedge \text{lr-boundary-nodep}(\text{addr}) \\
& \wedge (\text{area-name}(\text{addr}) = ' \mathbf{heap}) \\
& \wedge \text{definedp}(\text{LR-HEAP-NAME}, \text{data-seg}) \\
& \wedge \text{lr-boundary-nodep}(\text{lr-max-node}(\text{data-seg})) \\
& \wedge (\text{offset}(\text{addr}) < \text{offset}(\text{lr-max-node}(\text{data-seg}))) \\
& \wedge (n < \text{LR-NODE-SIZE})) \\
\rightarrow & \text{adpp}(\text{untag}(\text{add-addr}(\text{addr}, n)), \text{data-seg})
\end{aligned}$$

THEOREM: adpp-untag-add-addr-offset-on-free-listp

```

(lr-proper-p-areasp (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)
    ≠ 1)
  ∧ lr-proper-heapp (data-seg)
  ∧ (n < LR-NODE-SIZE))
→ 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)
  ∧ 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 cons))
  ∧ (p-max-temp-stk-size (new-l)
    ≠ (length (p-temp-stk (new-l))
      + car (s-apply-subr-r ('cons, new-s))))
  ∧ (p-max-ctrl-stk-size (new-l)
    ≠ (p-ctrl-stk-size (p-ctrl-stk (new-l))
      + cadr (s-apply-subr-r ('cons, new-s))))
  ∧ (p-word-size (new-l)
    ≠ max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r ('cons, 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))
    ≠ caddr (s-apply-subr-r ('cons, new-s))))
  ∧ (length (p-temp-stk (new-l)) ≠ arity ('cons))
  ∧ (length (s-ans (new-s)) = arity ('cons)))
→ (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))
    < caddr (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))
    ↯ caddr (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))
    ↯ caddr (s-apply-subr-r ('listp, new-s)))

```

$\wedge$ (length (p-temp-stk (*new-l*)) $\not\leq$ arity ('listp))
 $\wedge$ (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)
 $\wedge$ 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 nlistp))
 $\wedge$ (p-max-temp-stk-size (*new-l*)
 $\not\leq$ (length (p-temp-stk (*new-l*))
+ car (s-apply-subr-r ('nlistp, *new-s*))))
 $\wedge$ (p-max-ctrl-stk-size (*new-l*)
 $\not\leq$ (p-ctrl-stk-size (p-ctrl-stk (*new-l*))
+ cadr (s-apply-subr-r ('nlistp, *new-s*))))
 $\wedge$ (p-word-size (*new-l*)
 $\not\leq$ max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r ('nlistp, *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*))
 $\not\leq$ caddr (s-apply-subr-r ('nlistp, *new-s*)))
 $\wedge$ (length (p-temp-stk (*new-l*)) $\not\leq$ arity ('nlistp))
 $\wedge$ (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)

```

 $\wedge$  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 true))
 $\wedge$  (p-max-temp-stk-size (new-l)
 $\not\leq$  (length (p-temp-stk (new-l))
    + car (s-apply-subr-r ('true, new-s))))
 $\wedge$  (p-max-ctrl-stk-size (new-l)
 $\not\leq$  (p-ctrl-stk-size (p-ctrl-stk (new-l))
    + cadr (s-apply-subr-r ('true, new-s))))
 $\wedge$  (p-word-size (new-l)
 $\not\leq$  max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r ('true, 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))
 $\not\leq$  caddr (s-apply-subr-r ('true, new-s)))
 $\wedge$  (length (p-temp-stk (new-l))  $\not\leq$  arity ('true))
 $\wedge$  (length (s-ans (new-s)) = arity ('true)))
 $\rightarrow$  (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)
 $\wedge$  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 truep))
 $\wedge$  (p-max-temp-stk-size (new-l)
 $\not\leq$  (length (p-temp-stk (new-l))
    + car (s-apply-subr-r ('truep, new-s))))
 $\wedge$  (p-max-ctrl-stk-size (new-l)
 $\not\leq$  (p-ctrl-stk-size (p-ctrl-stk (new-l))

```

```

      + cadr (s-apply-subr-r ('truep, new-s))))
 $\wedge$  (p-word-size (new-l)
 $\not\leq$  max (S-MAX-SUBR-REQS, caddr (s-apply-subr-r ('truep, 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))
 $\not\leq$  caddr (s-apply-subr-r ('truep, new-s)))
 $\wedge$  (length (p-temp-stk (new-l))  $\not\leq$  arity ('truep))
 $\wedge$  (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
 $\text{listp}(l) \rightarrow (\text{length}(\text{last}(l)) = 1)$

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)))
 $\wedge$ 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)
 $\wedge$ (p-psw (new-l) = 'run)
 $\wedge$ (s-err-flag (s-eval ('list, s-set-pos (s, pos), c)) = 'run)
 $\wedge$ 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)
 $\wedge$ (pos = dv (s-pos (s), 1))
 $\wedge$ (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
 $((\neg \text{definedp}(\text{car}(\text{lr-expr}(l)), \text{P-RUNTIME-SUPPORT-PROGRAMS}))$
 $\wedge (\text{car}(\text{lr-expr}(l)) \neq \text{'if})$
 $\wedge \text{subrp}(\text{car}(\text{lr-expr}(l)))$
 $\wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l), \text{program-body}(\text{p-current-program}(l))))$
 $\rightarrow (\neg \text{lr-programs-properp}(l, \text{table}))$

THEOREM: p-psw-run-p-run-subr-lr-check-resourcesp
let *new-l* **be** $\text{lr-eval}(\text{'list}, \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), \text{pos}), c),$
new-s **be** $\text{s-eval}(\text{'list}, \text{s-set-pos}(s, \text{pos}), c),$
pc **be** $\text{lr-return-pc}(\text{s->lr1}(s, l, \text{table})),$
r **be** $\text{s-apply-subr-r}(\text{car}(\text{s-expr}(s)), \text{s-eval}(\text{'list}, \text{s-set-pos}(s, \text{pos}), c))$
in
 $(\text{proper-p-statep}(\text{lr->p}(\text{s->lr1}(s, l, \text{table})))$
 $\wedge (c \neq 0)$
 $\wedge \text{listp}(\text{s-expr}(s))$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{'if})$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{'quote})$
 $\wedge (\text{s-err-flag}(\text{new-s}) = \text{'run})$
 $\wedge \text{subrp}(\text{car}(\text{s-expr}(s)))$
 $\wedge (\text{p-psw}(\text{new-l}) = \text{'run})$
 $\wedge \text{lr-proper-heapp}(\text{p-data-segment}(l))$
 $\wedge \text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s)))$
 $\wedge \text{lr-programs-properp}(\text{s->lr1}(s, l, \text{table}), \text{table})$
 $\wedge \text{lr-s-similar-statesp}(\text{s-params}(s),$
 $\text{s-temps}(s),$
 $\text{s->lr1}(s, l, \text{table}),$
 $\text{table})$
 $\wedge \text{s-good-statep}(s, c)$

$$\begin{aligned}
& \wedge \text{ (p-max-temp-stk-size } (l) \\
& \quad \not\leq (\text{length (p-temp-stk } (l)) \\
& \quad \quad + \text{ arity (car (s-expr } (s))) \\
& \quad \quad + \text{ car } (r))) \\
& \wedge \text{ (p-max-ctrl-stk-size } (l) \\
& \quad \not\leq (\text{p-ctrl-stk-size (p-ctrl-stk } (l)) + \text{ cadr } (r))) \\
& \wedge \text{ (p-word-size } (l) \not\leq \text{max (S-MAX-SUBR-REQS, cadr } (r))) \\
& \wedge \text{ (lr-count-free-nodes (fetch (LR-FP-ADDR, p-data-segment } (l)), \\
& \quad \quad \quad \text{lr-free-list-nodes (lr-max-node (p-data-segment } (l)), \\
& \quad \quad \quad \text{p-data-segment } (l)), \\
& \quad \quad \quad \text{p-data-segment } (l)) \\
& \quad \not\leq (\text{caddr } (r) \\
& \quad \quad + \text{ s-eval-heap-r ('list, s-set-pos (s, pos), c))) \\
& \wedge \text{ (pos = dv (s-pos (s), 1))} \\
& \rightarrow \text{ (p-psw (p-run-subr (car (s-expr } (s)), \text{ p-set-pc (lr->p (new-l), pc))} \\
& \quad = \text{ 'run) endlet}
\end{aligned}$$

EVENT: Disable lr-programs-properp-definedp-subrp-runtime-support.

THEOREM: not-lessp-help-fact

$$((x \not\leq y) \wedge (x \not\leq z)) \rightarrow ((x < \max(y, z)) = \mathbf{f})$$

THEOREM: p-psw-run-lr-apply-subr-lr-check-resourcesp

$$\begin{aligned}
& ((c \neq 0) \\
& \wedge \text{ listp (s-expr } (s)) \\
& \wedge \text{ (car (s-expr } (s)) \neq \text{ 'if)} \\
& \wedge \text{ (car (s-expr } (s)) \neq \text{ 'quote)} \\
& \wedge \text{ (s-err-flag (s-eval ('list, s-set-pos (s, dv (s-pos (s), 1)), c)) = 'run)} \\
& \wedge \text{ subrp (car (s-expr } (s))) \\
& \wedge \text{ (p-psw (lr-eval ('list,} \\
& \quad \quad \text{lr-set-pos (s->lr1 (s, l, table), dv (s-pos (s), 1)),} \\
& \quad \quad \text{c))} \\
& \quad = \text{ 'run)} \\
& \wedge \text{ proper-p-statep (lr->p (s->lr1 (s, l, table)))} \\
& \wedge \text{ lr-proper-heapp (p-data-segment } (l)) \\
& \wedge \text{ good-posp1 (s-pos (s), s-body (s-prog (s)))} \\
& \wedge \text{ lr-programs-properp (s->lr1 (s, l, table), table)} \\
& \wedge \text{ lr-s-similar-statesp (s-params (s), s-temps (s), s->lr1 (s, l, table), table)} \\
& \wedge \text{ s-good-statep (s, c)} \\
& \wedge \text{ lr-check-resourcesp (flag, s, l, c)} \\
& \wedge \text{ (p-word-size } (l) \not\leq \text{S-MAX-SUBR-REQS)} \\
& \wedge \text{ (flag \neq 'list)} \\
& \rightarrow \text{ (p-psw (lr-apply-subr (s->lr1 (s, l, table),} \\
& \quad \quad \text{lr-eval ('list,}
\end{aligned}$$

$$\begin{aligned}
& \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), \\
& \quad \text{dv}(\text{s-pos}(s), 1)), \\
& \quad c))) \\
= & \text{'run})
\end{aligned}$$

THEOREM: strip-logic-fnames-lr-compile-programs
 $\text{strip-logic-fnames}(\text{lr-compile-programs}(\text{programs}, \text{const-table}))$
 $= \text{strip-logic-fnames}(\text{programs})$

THEOREM: strip-logic-fnames-cdr-lr-compile-programs
 $\text{strip-logic-fnames}(\text{cdr}(\text{lr-compile-programs}(\text{programs}, \text{const-table})))$
 $= \text{strip-logic-fnames}(\text{cdr}(\text{programs}))$

THEOREM: lr-programs-properp-s->lr1-definedp-cdr-s-progs
 $(\text{lr-programs-properp}(\text{s->lr1}(s, l, \text{table}), \text{table})$
 $\wedge \text{listp}(\text{s-expr}(s))$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{'if})$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{'quote})$
 $\wedge (\neg \text{subrp}(\text{car}(\text{s-expr}(s))))$
 $\wedge \text{litatom}(\text{car}(\text{s-expr}(s)))$
 $\wedge \text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s)))$
 $\wedge \text{s-programs-okp}(\text{cdr}(\text{s-progs}(s)))$
 $\rightarrow \text{definedp}(\text{user-fname}(\text{car}(\text{s-expr}(s))), \text{cdr}(\text{s-progs}(s)))$

THEOREM: s-programs-okp-formals-not-f
 $(\text{s-programs-okp}(\text{progs}) \wedge (\text{prog} \in \text{progs})) \rightarrow (\text{s-formals}(\text{prog}) \neq \mathbf{f})$

THEOREM: not-lessp-plus-arity-length-formals
 $(\text{listp}(\text{s-expr}(s))$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{'if})$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq \text{'quote})$
 $\wedge (\neg \text{subrp}(\text{car}(\text{s-expr}(s))))$
 $\wedge \text{litatom}(\text{car}(\text{s-expr}(s)))$
 $\wedge \text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s)))$
 $\wedge \text{s-good-statep}(s, c)$
 $\rightarrow (((\text{arity}(\text{car}(\text{s-expr}(s))) + x) < \text{length}(\text{formals}(\text{car}(\text{s-expr}(s)))))$
 $= \mathbf{f})$

THEOREM: length-lr-make-temp-var-dcls
 $\text{length}(\text{lr-make-temp-var-dcls}(\text{temp-alist})) = \text{length}(\text{temp-alist})$

THEOREM: length-lr-make-temp-name-alist-1
 $\text{length}(\text{lr-make-temp-name-alist-1}(\text{initial}, \text{num-list}, \text{temp-list}, \text{formals}))$
 $= \text{length}(\text{temp-list})$

THEOREM: length-lr-make-temp-name-alist
 $\text{length}(\text{lr-make-temp-name-alist}(\text{temp-list}, \text{formals})) = \text{length}(\text{temp-list})$

THEOREM: p-ctrl-stk-size-0
 $(\text{p-ctrl-stk-size}(\text{ctrl-stk}) = 0) = (\neg \text{listp}(\text{ctrl-stk}))$

THEOREM: length-make-temps-entries
 $\text{length}(\text{make-temps-entries}(\text{list})) = \text{length}(\text{list})$

THEOREM: s-eval-ctrl-r-funcall-opener
let *arg-s* **be** $\text{s-eval}(' \text{list}, \text{s-set-pos}(s, \text{dv}(\text{s-pos}(s), 1)), c)$
in
 $((c \neq 0)$
 $\wedge \text{s-good-statep}(s, c)$
 $\wedge \text{listp}(\text{s-expr}(s))$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq ' \text{if})$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq ' \text{quote})$
 $\wedge (\neg \text{subrp}(\text{car}(\text{s-expr}(s))))$
 $\wedge \text{litatom}(\text{car}(\text{s-expr}(s)))$
 $\wedge (\text{flag} \neq ' \text{list})$
 $\wedge \text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s)))$
 $\wedge (\text{s-err-flag}(\text{arg-s}) = ' \text{run})$
 $\rightarrow (\text{s-eval-ctrl-r}(\text{flag}, s, c)$
 $= \text{max}(\text{s-eval-ctrl-r}(' \text{list},$
 $\text{s-set-pos}(s, \text{dv}(\text{s-pos}(s), 1)),$
 $c),$
 $1 + (1 + (\text{length}(\text{formals}(\text{car}(\text{s-expr}(s))))$
 $+ \text{length}(\text{s-temp-list}(\text{assoc}(\text{user-fname}(\text{car}(\text{s-expr}(s))),$
 $\text{s-progs}(s))))$
 $+ \text{s-eval-ctrl-r}(\text{t},$
 $\text{s-fun-call-state}(\text{arg-s},$
 $\text{car}(\text{s-expr}(s))),$
 $c - 1)))) \text{endlet}$

THEOREM: s-good-statep-formals-assoc-cdr-s-progs
 $(\text{s-good-statep}(s, c)$
 $\wedge \text{listp}(\text{s-expr}(s))$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq ' \text{if})$
 $\wedge (\text{car}(\text{s-expr}(s)) \neq ' \text{quote})$
 $\wedge (\neg \text{subrp}(\text{car}(\text{s-expr}(s))))$
 $\wedge \text{good-posp1}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s)))$
 $\wedge \text{litatom}(\text{car}(\text{s-expr}(s)))$
 $\wedge (\text{progs} = \text{cdr}(\text{s-progs}(s)))$
 $\rightarrow (\text{s-formals}(\text{assoc}(\text{user-fname}(\text{car}(\text{s-expr}(s))), \text{progs}))$
 $= \text{formals}(\text{car}(\text{s-expr}(s))))$

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))
 ∧ (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)
 ∧ (c ≠ 0)
 ∧ proper-p-statep (lr->p (s->lr1 (s, l, table)))
 ∧ lr-programs-properp (s->lr1 (s, l, table), table)
 ∧ (length (temp-list) = length (s-temp-list (s-prog)))
 ∧ (p-psw (lr-eval) = 'run)
 ∧ (s-err-flag (s-eval ('list, s-set-pos (s, dv (s-pos (s), 1)), c))
    = 'run)
 ∧ (p-max-ctrl-stk-size (l)
    ≠ (p-ctrl-stk-size (p-ctrl-stk (l))
      + s-eval-ctrl-r (flag, s, c)))
 ∧ (flag ≠ 'list))
→ (p-max-ctrl-stk-size (l)
   ≠ 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
¬ definedp (user-fname (*name*), P-RUNTIME-SUPPORT-PROGRAMS)

THEOREM: comp-programs-assoc-cons-opener
(user-fname (*name*) ≠ *prog1-name*)
→ (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* ≠ 0)

$$\begin{aligned}
& \wedge \text{listp}(\text{s-expr}(s)) \\
& \wedge (\text{car}(\text{s-expr}(s)) \neq \text{'if}) \\
& \wedge (\text{car}(\text{s-expr}(s)) \neq \text{'quote}) \\
& \wedge (\neg \text{subrp}(\text{car}(\text{s-expr}(s)))) \\
& \wedge \text{litatom}(\text{car}(\text{s-expr}(s))) \\
& \wedge \text{good-pospl}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s))) \\
& \wedge \text{lr-programs-properp}(\text{s->lr1}(s, l, \text{table}), \text{table}) \\
& \wedge (\text{p-psw}(\text{lr-eval}(\text{'list}, \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), \text{pos}), c)) \\
& \quad = \text{'run}) \\
& \wedge (\text{s-err-flag}(\text{s-eval}(\text{'list}, \text{s-set-pos}(s, \text{pos}), c)) = \text{'run}) \\
& \wedge (\text{pos} = \text{dv}(\text{s-pos}(s), 1)) \\
& \wedge \text{proper-p-statep}(\text{lr->p}(\text{s->lr1}(s, l, \text{table}))) \\
& \wedge \text{s-good-statep}(s, c) \\
& \wedge \text{lr-proper-heapp}(\text{p-data-segment}(l)) \\
& \wedge \text{lr-check-resourcesp}(\text{flag}, s, l, c) \\
& \wedge (\text{flag} \neq \text{'list}) \\
\rightarrow & (\text{p-psw}(\text{lr-funcall}(\text{s->lr1}(s, l, \text{table}), \\
& \quad \text{lr-eval}(\text{'list}, \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), \text{pos}), c))) \\
& \quad = \text{'run})
\end{aligned}$$

THEOREM: lessp-max-arg2

$\max(x, y) \not\prec y$

THEOREM: not-lessp-plus-arity-length-formals-alt

$$\begin{aligned}
& (\text{listp}(\text{s-expr}(s)) \\
& \wedge (\text{car}(\text{s-expr}(s)) \neq \text{'if}) \\
& \wedge (\text{car}(\text{s-expr}(s)) \neq \text{'quote}) \\
& \wedge (\neg \text{subrp}(\text{car}(\text{s-expr}(s)))) \\
& \wedge \text{litatom}(\text{car}(\text{s-expr}(s))) \\
& \wedge \text{good-pospl}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s))) \\
& \wedge \text{s-good-statep}(s, c)) \\
\rightarrow & (((x + \text{arity}(\text{car}(\text{s-expr}(s)))) < \text{length}(\text{formals}(\text{car}(\text{s-expr}(s)))))) \\
& = \text{f})
\end{aligned}$$

THEOREM: listp-lr-compile-programs

$\text{listp}(\text{lr-compile-programs}(\text{progs}, \text{table})) = \text{listp}(\text{progs})$

THEOREM: caar-lr-compile-programs

$\text{listp}(\text{progs}) \rightarrow (\text{caar}(\text{lr-compile-programs}(\text{progs}, \text{table})) = \text{caar}(\text{progs}))$

THEOREM: length-p-temp-stk-lr-eval-lr-funcall

let *lr-eval* **be** $\text{lr-eval}(\text{'list}, \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), \text{pos}), c)$
in
 $(\text{listp}(\text{s-expr}(s))$
 $\wedge (\neg \text{subrp}(\text{car}(\text{s-expr}(s))))$

```

 $\wedge$  (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  $\neq$  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
 $((c \neq 0)$
 $\wedge$ listp (s-expr (*s*))
 $\wedge$ (car (s-expr (*s*)) $\neq$ 'if)

$$\begin{aligned}
& \wedge (\text{car}(\text{s-expr}(s)) \neq \text{S-TEMP-EVAL}) \\
& \wedge (\text{car}(\text{s-expr}(s)) \neq \text{S-TEMP-TEST}) \\
& \wedge (\text{car}(\text{s-expr}(s)) \neq \text{S-TEMP-FETCH}) \\
& \wedge (\text{car}(\text{s-expr}(s)) \neq \text{'quote}) \\
& \wedge (\neg \text{subrp}(\text{car}(\text{s-expr}(s)))) \\
& \wedge \text{good-pospl}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s))) \\
& \wedge \text{litatom}(\text{car}(\text{s-expr}(s))) \\
& \wedge \text{lr-programs-properp}(\text{s->lr1}(s, l, \text{table}), \text{table}) \\
& \wedge (\text{p-psw}(\text{lr-eval}(\text{'list}, \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), \text{pos}), c)) \\
& \quad = \text{'run}) \\
& \wedge (\text{s-err-flag}(\text{s-eval}(\text{'list}, \text{s-set-pos}(s, \text{pos}), c)) = \text{'run}) \\
& \wedge (\text{pos} = \text{dv}(\text{s-pos}(s), 1)) \\
& \wedge \text{proper-p-statep}(\text{lr->p}(\text{s->lr1}(s, l, \text{table}))) \\
& \wedge \text{s-good-statep}(s, c) \\
& \wedge \text{lr-proper-heapp}(\text{p-data-segment}(l)) \\
& \wedge \text{lr-check-resourcesp}(\text{flag}, s, l, c) \\
& \wedge (\text{flag} \neq \text{'list}) \\
& \wedge \text{lr-s-similar-statesp}(\text{s-params}(s), \text{s-temps}(s), \text{s->lr1}(s, l, \text{table}), \text{table})) \\
\rightarrow & \text{lr-check-resourcesp}(\mathbf{t}, \\
& \quad \text{s-fun-call-state}(\text{s-eval}(\text{'list}, \\
& \quad \quad \text{s-set-pos}(s, \text{pos}), \\
& \quad \quad c), \\
& \quad \quad \text{car}(\text{s-expr}(s))), \\
& \quad \text{lr-funcall}(\text{s->lr1}(s, l, \text{table}), \\
& \quad \quad \text{lr-eval}(\text{'list}, \\
& \quad \quad \quad \text{lr-set-pos}(\text{s->lr1}(s, l, \text{table}), \\
& \quad \quad \quad \text{pos}), \\
& \quad \quad c)), \\
& \quad c - 1)
\end{aligned}$$

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

$$\begin{aligned}
& (\text{listp}(\text{s-expr}(s))) \\
& \wedge (\text{car}(\text{s-expr}(s)) \neq \text{'if}) \\
& \wedge (\text{car}(\text{s-expr}(s)) \neq \text{'quote}) \\
& \wedge (\text{s-err-flag}(\text{s-eval}(\text{'list}, \text{s-set-pos}(s, \text{dv}(\text{s-pos}(s), 1)), c)) = \text{'run}) \\
& \wedge \text{subrp}(\text{car}(\text{s-expr}(s))) \\
& \wedge (\text{p-psw}(\text{lr-apply-subr}(\text{s->lr1}(s, l, \text{table}), \text{new-l})) = \text{'run}) \\
& \wedge \text{good-pospl}(\text{s-pos}(s), \text{s-body}(\text{s-prog}(s))) \\
& \wedge \text{s-good-statep}(s, c) \\
\rightarrow & (\text{car}(\text{s-apply-subr-r}(\text{car}(\text{s-expr}(s)), \\
& \quad \text{s-eval}(\text{'list}, \text{s-set-pos}(s, \text{dv}(\text{s-pos}(s), 1)), c))) \\
& \quad \not\leq 1)
\end{aligned}$$

THEOREM: length-p-temp-stk-lr-eval-lr-set-pos-flag-t
 (proper-p-statep (lr->p (s->lr1 (s, l, table)))
 $\wedge$ good-pospl (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-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)
 $\wedge$ (s-err-flag (s-eval (flag, s, c)) = 'run)
 $\wedge$ good-posp (flag, s-pos (s), s-body (s-prog (s)))
 $\wedge$ s-good-statep (s, c)
 $\wedge$ (flag $\neq$ 'list))
 $\rightarrow$ (s-eval-temp-r (flag, s, c) $\not\prec$ 1)

THEOREM: p-psw-run-p-psw-lr-if-ok-not-run-check-resourcesp
 ((flag $\neq$ 'list)
 $\wedge$ (c $\neq$ 0)
 $\wedge$ listp (s-expr (s))
 $\wedge$ (car (s-expr (s)) = 'if)
 $\wedge$ s-good-statep (s, c)
 $\wedge$ lr-programs-properp (s->lr1 (s, l, table), table)
 $\wedge$ proper-p-statep (lr->p (s->lr1 (s, l, table)))
 $\wedge$ (s-err-flag (s-eval (t, s-set-pos (s, dv (s-pos (s), 1)), c)) = 'run)
 $\wedge$ (p-psw (lr-if-ok (lr-eval (t,
 lr-set-pos (s->lr1 (s, l, table), dv (s-pos (s), 1)),
 c)))
 $\neq$ 'run)
 $\wedge$ (p-psw (lr-eval (t, lr-set-pos (s->lr1 (s, l, table), dv (s-pos (s), 1)), c))
 $=$ 'run)
 $\wedge$ good-pospl (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)
 $\wedge$ (c $\neq$ 0)
 $\wedge$ (c $\in \mathbf{N}$)
 $\wedge$ listp (s-expr (s))
 $\wedge$ (car (s-expr (s)) = S-TEMP-TEST)
 $\wedge$ (p-max-temp-stk-size (l) < (2 + length (p-temp-stk (l))))
 $\wedge$ 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) < 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 (pname)

```

= if listp (pname)
  then l-restrict-subrps (t, body (car (pname)))
    ∧ l-restrict-subrps-progs (cdr (pname))
  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)
   ∧ 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 ∈ N)
 → (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) ∈ N) ∧ (offset (addr2) ∈ N) ∧ listp (list))
 → (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) ≠ offset (addr2))
 ∧ (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)

$$\begin{aligned}
&= (\text{adpp}(\text{identity}(\text{untag}(\text{LR-0-ADDR})), \text{data-seg}) \\
&\quad \wedge (\text{type}(\text{fetch}(\text{identity}(\text{add-addr}(\text{LR-0-ADDR}, \text{LR-REF-COUNT-OFFSET})), \\
&\quad \quad \text{data-seg})) \\
&\quad = \text{'nat'}) \\
&\quad \wedge (\text{untag}(\text{fetch}(\text{identity}(\text{LR-0-ADDR}), \text{data-seg})) = \text{LR-ADD1-TAG}) \\
&\quad \wedge (\text{untag}(\text{fetch}(\text{identity}(\text{add-addr}(\text{LR-0-ADDR}, \text{LR-UNBOX-NAT-OFFSET})), \\
&\quad \quad \text{data-seg})) \\
&\quad = 0))
\end{aligned}$$

EVENT: Disable lr-valp-0-lr-0-addr-opener.

$$\begin{aligned}
&\text{THEOREM: lr-valp-t-lr-t-addr-opener} \\
&\text{lr-valp}(\mathbf{t}, \text{identity}(\text{LR-T-ADDR}), \text{data-seg}) \\
&= (\text{adpp}(\text{identity}(\text{untag}(\text{LR-T-ADDR})), \text{data-seg}) \\
&\quad \wedge (\text{type}(\text{fetch}(\text{identity}(\text{add-addr}(\text{LR-T-ADDR}, \text{LR-REF-COUNT-OFFSET})), \\
&\quad \quad \text{data-seg})) \\
&\quad = \text{'nat'}) \\
&\quad \wedge (\text{untag}(\text{fetch}(\text{identity}(\text{LR-T-ADDR}), \text{data-seg})) = \text{LR-TRUE-TAG}))
\end{aligned}$$

EVENT: Disable lr-valp-t-lr-t-addr-opener.

$$\begin{aligned}
&\text{THEOREM: lr-valp-f-lr-f-addr-opener} \\
&\text{lr-valp}(\mathbf{f}, \text{identity}(\text{LR-F-ADDR}), \text{data-seg}) \\
&= (\text{adpp}(\text{identity}(\text{untag}(\text{LR-F-ADDR})), \text{data-seg}) \\
&\quad \wedge (\text{type}(\text{fetch}(\text{identity}(\text{add-addr}(\text{LR-F-ADDR}, \text{LR-REF-COUNT-OFFSET})), \\
&\quad \quad \text{data-seg})) \\
&\quad = \text{'nat'}) \\
&\quad \wedge (\text{untag}(\text{fetch}(\text{identity}(\text{LR-F-ADDR}), \text{data-seg})) = \text{LR-FALSE-TAG}))
\end{aligned}$$

EVENT: Disable lr-valp-f-lr-f-addr-opener.

$$\begin{aligned}
&\text{THEOREM: definedp-table-definedp-cdr-lr-compile-quote} \\
&\text{definedp}(x, \text{table}) \\
&\rightarrow \text{definedp}(x, \text{cdr}(\text{lr-compile-quote}(\text{flag}, \text{object}, \text{data-seg}, \text{table})))
\end{aligned}$$

$$\begin{aligned}
&\text{THEOREM: definedp-car-lr-compile-quote} \\
&\text{definedp}(x, \text{car}(\text{lr-compile-quote}(\text{flag}, \text{object}, \text{data-seg}, \text{table}))) \\
&= \text{definedp}(x, \text{data-seg})
\end{aligned}$$

$$\begin{aligned}
&\text{THEOREM: lr-proper-p-areasp-car-lr-compile-quote} \\
&\text{lr-proper-p-areasp}(\text{data-seg}) \\
&\rightarrow \text{lr-proper-p-areasp}(\text{car}(\text{lr-compile-quote}(\text{flag}, \text{object}, \text{data-seg}, \text{table})))
\end{aligned}$$

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)
→ 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)
→ 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)
→ 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)))
→ (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)))
→ (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))
 ∧ same-signature (data-seg, car (pair))
 ∧ lr-proper-p-areasp (data-seg)
 ∧ definedp (LR-HEAP-NAME, data-seg)
 ∧ lr-boundary-nodep (lr-max-node (data-seg))
 ∧ ((length (cdr (assoc (LR-HEAP-NAME, data-seg))) - 1)
    ≠ (offset (fetch (LR-FP-ADDR, car (pair))) + length (list))))
→ 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)
 ∧ 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,

)))

THEOREM: same-signature-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*)))
 $\rightarrow$ 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*)
 $\wedge$ 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-data-seg-table-body (*flag*, *expr*, *data-seg*, *table*)))
 $\wedge$ lr-proper-free-listp (car (lr-data-seg-table-body (*flag*,
expr,
data-seg,

))))

EVENT: Disable same-signature-car-lr-compile-quote.

THEOREM: lr-max-node-car-lr-data-seg-table-body
 (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-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
 $\text{adpp}(\text{identity}(\text{untag}(\text{LR-FP-ADDR})), \text{lr-init-data-seg}(\text{heap-size}))$

THEOREM: lr-max-node-lr-init-data-seg
 $(\text{heap-size} \not\leq 2)$
 $\rightarrow (\text{lr-max-node}(\text{lr-init-data-seg}(\text{heap-size}))$
 $\quad = \text{tag}(\text{'addr},$
 $\quad \quad \text{cons}(\text{identity}(\text{LR-HEAP-NAME}),$
 $\quad \quad \text{identity}(\text{LR-NODE-SIZE}) * \text{heap-size}))$

THEOREM: fetch-lr-fp-addr-lr-init-data-seg
 $\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \text{lr-init-data-seg}(\text{heap-size}))$
 $= \text{identity}(\text{add-addr}(\text{LR-F-ADDR}, \text{LR-NODE-SIZE}))$

THEOREM: lr-boundary-nodep-not-lessp-fact-helper
 $((x < (y * z)) \wedge ((x \bmod y) = 0) \wedge (x \in \mathbf{N}))$
 $\rightarrow ((x < (y * (z - 1))) = (x \neq (y * (z - 1))))$

THEOREM: lessp-times-difference-fact
 $((z \neq 0) \wedge (x \neq 0) \wedge ((x \bmod y) = 0))$
 $\rightarrow (((x - y) < (y * (z - 1))) = (x < (y * z)))$

THEOREM: lessp-times-difference-node-on-boundaryp-fact
 $((\text{heap-size} \neq 0) \wedge (\text{offset}(\text{addr}) \neq 0) \wedge \text{lr-boundary-nodep}(\text{addr}))$
 $\rightarrow ((((((\text{offset}(\text{addr}) - 1) - 1) - 1) - 1)$
 $\quad < (\text{identity}(\text{LR-NODE-SIZE}) * (\text{heap-size} - 1)))$
 $\quad = (\text{offset}(\text{addr}) < (\text{identity}(\text{LR-NODE-SIZE}) * \text{heap-size})))$

THEOREM: lr-boundary-nodep-lessp-lr-node-size-0
 $\text{lr-boundary-nodep}(\text{addr})$
 $\rightarrow (((((\text{offset}(\text{addr}) - 1) = 1) = \mathbf{f})$
 $\quad \wedge (((((\text{offset}(\text{addr}) - 1) - 1) = 1) = \mathbf{f}))$

THEOREM: lr-boundary-nodep-lessp-lr-node-size-1
 $((\text{offset}(\text{addr}) \in \mathbf{N}) \wedge \text{lr-boundary-nodep}(\text{addr}) \wedge (n < \text{LR-NODE-SIZE}))$
 $\rightarrow ((n < \text{offset}(\text{addr})) = (\text{offset}(\text{addr}) \neq 0))$

THEOREM: lr-boundary-nodep-lessp-lr-node-size-2
 $\text{lr-boundary-nodep}(\text{addr}) \rightarrow ((\text{offset}(\text{addr}) = 1) = \mathbf{f})$

DEFINITION:
 $\text{induct-hint-17}(\text{addr1}, \text{size}, \text{addr2})$
 $= \text{if } \text{size} \simeq 0 \text{ then t}$
 $\quad \text{elseif } \text{offset}(\text{addr2}) \simeq 0 \text{ then t}$
 $\quad \text{else } \text{induct-hint-17}(\text{add-addr}(\text{addr1}, \text{LR-NODE-SIZE}),$
 $\quad \quad \text{size} - 1,$
 $\quad \quad \text{sub-addr}(\text{addr2}, \text{LR-NODE-SIZE})) \text{ endif}$

THEOREM: get-cdr-lr-init-heap-contents
 $((\text{offset}(\text{addr2}) < (\text{LR-NODE-SIZE} * \text{heap-size}))$
 $\wedge \text{lr-boundary-nodep}(\text{addr2})$
 $\wedge (\text{offset}(\text{addr2}) \in \mathbf{N})$
 $\wedge (\text{offset}(\text{addr1}) \in \mathbf{N}))$
 $\rightarrow (\text{get}(\text{offset}(\text{addr2}), \text{cdr}(\text{lr-init-heap-contents}(\text{addr1}, \text{heap-size})))$
 $= \text{add-addr}(\text{add-addr}(\text{addr1}, \text{offset}(\text{addr2})), \text{LR-NODE-SIZE}))$

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
 $(\text{heap-size} \not\leq 2)$
 $\rightarrow (\text{length}(\text{cdr}(\text{assoc}(\text{identity}(\text{LR-HEAP-NAME}), \text{lr-init-data-seg}(\text{heap-size}))))$
 $= (1 + (\text{heap-size} * \text{identity}(\text{LR-NODE-SIZE}))))$

THEOREM: fetch-add-addr-ref-count-offset-lr-init-data-seg-help-1
 $((\text{offset}(\text{addr}) = 0)$
 $\wedge (\text{type}(\text{addr}) = \text{'addr})$
 $\wedge (\text{area-name}(\text{addr}) = \text{'heap}))$
 $\rightarrow (\text{add-addr}(\text{addr}, 4) = \text{'(addr (heap . 4))})$

THEOREM: equal-add-addr-fact
 $(\text{type}(\text{addr1}) = \text{type}(\text{addr2}))$
 $\rightarrow ((\text{add-addr}(\text{addr1}, n1) = \text{add-addr}(\text{addr2}, n2))$
 $= ((\text{area-name}(\text{addr1}) = \text{area-name}(\text{addr2}))$
 $\wedge ((\text{offset}(\text{addr1}) + n1) = (\text{offset}(\text{addr2}) + n2))))$

DEFINITION:
 $\text{lr-all-nodes}(\text{min-offset}, \text{max-addr})$
 $= \text{if } \text{offset}(\text{max-addr}) \simeq 0 \text{ then nil}$
 $\quad \text{elseif } \text{min-offset} \not\leq \text{offset}(\text{max-addr}) \text{ then nil}$
 $\quad \text{else cons}(\text{sub-addr}(\text{max-addr}, \text{LR-NODE-SIZE}),$
 $\quad \quad \text{lr-all-nodes}(\text{min-offset},$
 $\quad \quad \text{sub-addr}(\text{max-addr}, \text{LR-NODE-SIZE}))) \text{ endif}$

DEFINITION:
 $\text{induct-hint-19}(\text{addr}, \text{max-addr})$
 $= \text{if } \text{offset}(\text{addr}) < \text{offset}(\text{max-addr})$
 $\quad \text{then induct-hint-19}(\text{add-addr}(\text{addr}, \text{LR-NODE-SIZE}), \text{max-addr})$
 $\quad \text{else t endif}$

THEOREM: lessp-times-plus-fact

$$(n \neq 0) \rightarrow (((n * v) < (n + (n * w))) = (v < (1 + w)))$$

THEOREM: lessp-sub1-lessp-fact

$$\begin{aligned} & ((x \in \mathbf{N}) \wedge (y \in \mathbf{N}) \wedge (x \neq 0) \wedge (x \neq y)) \\ \rightarrow & (((x - 1) < y) = (x < y)) \end{aligned}$$

THEOREM: remainder-difference-not-equal-lessp-fact

$$\begin{aligned} & (((x \bmod n) = 0) \\ & \wedge ((y \bmod n) = 0) \\ & \wedge (x \neq (y - n)) \\ & \wedge (y \not< n) \\ & \wedge (x \in \mathbf{N}) \\ & \wedge (y \in \mathbf{N})) \\ \rightarrow & ((x < (y - n)) = (x < y)) \end{aligned}$$

EVENT: Disable lessp-sub1-lessp-fact.

THEOREM: lr-boundaryp-nodep-difference-node-size

$$\begin{aligned} & \text{lr-boundary-offsetp}(\text{offset}) \\ \rightarrow & \text{lr-boundary-offsetp}(\text{offset} - \text{LR-NODE-SIZE}) \end{aligned}$$

THEOREM: lr-boundary-offsetp-difference-not-equal-lessp-fact-1

$$\begin{aligned} & (\text{lr-boundary-offsetp}(x) \\ & \wedge \text{lr-boundary-offsetp}(y) \\ & \wedge (x \neq (y - \text{LR-NODE-SIZE})) \\ & \wedge (y \not< \text{LR-NODE-SIZE}) \\ & \wedge (x \in \mathbf{N}) \\ & \wedge (y \in \mathbf{N})) \\ \rightarrow & ((x < (y - \text{LR-NODE-SIZE})) = (x < y)) \end{aligned}$$

THEOREM: member-lr-all-nodes-helper

$$\begin{aligned} & ((\text{offset}(\text{max-addr}) \neq 0) \\ & \wedge (\text{offset}(\text{addr}) \in \mathbf{N}) \\ & \wedge (\text{caddr}(\text{addr}) = \mathbf{nil}) \\ & \wedge \text{listp}(\text{addr}) \\ & \wedge \text{listp}(\text{untag}(\text{addr})) \\ & \wedge \text{lr-boundary-nodep}(\text{addr}) \\ & \wedge \text{lr-boundary-nodep}(\text{max-addr}) \\ & \wedge (\text{area-name}(\text{addr}) = \text{area-name}(\text{max-addr})) \\ & \wedge (\text{type}(\text{addr}) = \text{type}(\text{max-addr})) \\ & \wedge (\text{addr} \neq \text{sub-addr}(\text{max-addr}, \text{LR-NODE-SIZE}))) \\ \rightarrow & ((\text{offset}(\text{addr}) < (((\text{offset}(\text{max-addr}) - 1) - 1) - 1) - 1)) \\ & = (\text{offset}(\text{addr}) < \text{offset}(\text{max-addr}))) \end{aligned}$$

THEOREM: member-lr-all-nodes

$$\begin{aligned}
& ((\text{type}(addr) = \text{'addr}) \\
& \quad \wedge \quad (\text{caddr}(addr) = \mathbf{nil}) \\
& \quad \wedge \quad \text{listp}(addr) \\
& \quad \wedge \quad \text{lr-boundary-nodep}(addr) \\
& \quad \wedge \quad (\text{area-name}(addr) = \text{LR-HEAP-NAME}) \\
& \quad \wedge \quad \text{listp}(\text{untag}(addr)) \\
& \quad \wedge \quad (\text{offset}(addr) \in \mathbf{N}) \\
& \quad \wedge \quad (\text{type}(max\text{-}addr) = \text{'addr}) \\
& \quad \wedge \quad (\text{caddr}(max\text{-}addr) = \mathbf{nil}) \\
& \quad \wedge \quad \text{listp}(max\text{-}addr) \\
& \quad \wedge \quad \text{lr-boundary-nodep}(max\text{-}addr) \\
& \quad \wedge \quad (\text{area-name}(max\text{-}addr) = \text{LR-HEAP-NAME}) \\
& \quad \wedge \quad \text{listp}(\text{untag}(max\text{-}addr)) \\
& \quad \wedge \quad \text{lr-boundary-offsetp}(min\text{-}offset) \\
& \quad \wedge \quad (\text{offset}(addr) \not\leq min\text{-}offset)) \\
\rightarrow & ((addr \in \text{lr-all-nodes}(min\text{-}offset, max\text{-}addr)) \\
& \quad = \quad (\text{offset}(addr) < \text{offset}(max\text{-}addr)))
\end{aligned}$$

EVENT: Disable member-lr-all-nodes-helper.

THEOREM: lr-all-nodes-nil

$$\begin{aligned}
& (\text{lr-all-nodes}(min\text{-}offset, max\text{-}addr) = \mathbf{nil}) \\
= & ((\text{offset}(max\text{-}addr) \simeq 0) \vee (min\text{-}offset \not\leq \text{offset}(max\text{-}addr)))
\end{aligned}$$

THEOREM: delete-append

$$\begin{aligned}
& \text{delete}(e, \text{append}(x, y)) \\
= & \quad \mathbf{if} \ e \in x \ \mathbf{then} \ \text{append}(\text{delete}(e, x), y) \\
& \quad \mathbf{else} \ \text{append}(x, \text{delete}(e, y)) \ \mathbf{endif}
\end{aligned}$$

THEOREM: lessp-difference-node-size-sub-addr-2

$$\begin{aligned}
& ((offset < \text{offset}(addr)) \\
& \quad \wedge \quad \text{lr-boundary-nodep}(addr) \\
& \quad \wedge \quad (\text{offset}(addr) \in \mathbf{N}) \\
& \quad \wedge \quad \text{lr-boundary-offsetp}(offset)) \\
\rightarrow & (((\text{offset}(addr) - \text{identity}(\text{LR-NODE-SIZE})) < offset) = \mathbf{f})
\end{aligned}$$

THEOREM: not-member-lr-all-nodes-too-small-addr

$$\begin{aligned}
& (\text{lr-boundary-nodep}(addr) \\
& \quad \wedge \quad \text{lr-boundary-nodep}(max\text{-}addr) \\
& \quad \wedge \quad \text{lr-boundary-offsetp}(min\text{-}offset) \\
& \quad \wedge \quad (\text{offset}(addr) < min\text{-}offset) \\
& \quad \wedge \quad (min\text{-}offset \in \mathbf{N})) \\
\rightarrow & (addr \notin \text{lr-all-nodes}(min\text{-}offset, max\text{-}addr))
\end{aligned}$$

THEOREM: plist-delete

$$\text{plist}(\text{delete}(e, x)) = \text{delete}(e, \text{plist}(x))$$

THEOREM: lr-check-free-nodes-plist-node-list

$$\begin{aligned} &\text{lr-check-free-nodes}(\text{addr}, \text{plist}(\text{node-list}), \text{data-seg}, \text{max-addr}) \\ &= \text{lr-check-free-nodes}(\text{addr}, \text{node-list}, \text{data-seg}, \text{max-addr}) \end{aligned}$$

THEOREM: lr-all-nodes-offset-same-max

$$\text{lr-all-nodes}(\text{offset}(\text{addr}), \text{addr}) = \mathbf{nil}$$

THEOREM: lr-all-nodes-offset-max-addr-opener-helper

$$\begin{aligned} &((\text{offset}(\text{addr}) \neq 0) \\ &\wedge \text{lr-boundary-nodep}(\text{addr}) \\ &\wedge (\text{offset} \in \mathbf{N}) \\ &\wedge \text{lr-boundary-offsetp}(\text{offset}) \\ &\wedge (\text{offset} < \text{offset}(\text{addr}))) \\ \rightarrow &((\text{offset} < (((\text{offset}(\text{addr}) - 1) - 1) - 1) - 1)) \\ &= (\text{offset} \neq (((\text{offset}(\text{addr}) - 1) - 1) - 1) - 1)) \end{aligned}$$

THEOREM: lr-all-nodes-lessp-max-addr-opener

$$\begin{aligned} &((\text{type}(\text{max-addr}) = \mathbf{'addr}) \\ &\wedge \text{listp}(\text{max-addr}) \\ &\wedge (\text{caddr}(\text{max-addr}) = \mathbf{nil}) \\ &\wedge \text{listp}(\text{untag}(\text{max-addr})) \\ &\wedge (\text{offset}(\text{max-addr}) \in \mathbf{N}) \\ &\wedge \text{lr-boundary-nodep}(\text{max-addr}) \\ &\wedge (\text{area-name}(\text{max-addr}) = \text{LR-HEAP-NAME}) \\ &\wedge (\text{min-offset} < \text{offset}(\text{max-addr})) \\ &\wedge (\text{min-offset} \in \mathbf{N}) \\ &\wedge \text{lr-boundary-offsetp}(\text{min-offset})) \\ \rightarrow &(\text{lr-all-nodes}(\text{min-offset}, \text{max-addr}) \\ &= \text{append}(\text{lr-all-nodes}(\text{min-offset} + \text{identity}(\text{LR-NODE-SIZE}), \\ &\quad \text{max-addr}), \\ &\quad \text{list}(\text{tag}(\mathbf{'addr}, \\ &\quad \text{cons}(\text{identity}(\text{LR-HEAP-NAME}), \text{min-offset})))))) \end{aligned}$$

THEOREM: fetch-init-init-data-seg-generalized

$$\begin{aligned} &((\text{offset}(\text{addr}) \in \mathbf{N}) \\ &\wedge (\text{type}(\text{addr}) = \mathbf{'addr}) \\ &\wedge (\text{caddr}(\text{addr}) = \mathbf{nil}) \\ &\wedge \text{listp}(\text{addr}) \\ &\wedge \text{listp}(\text{untag}(\text{addr})) \\ &\wedge \text{lr-boundary-nodep}(\text{addr}) \\ &\wedge (\text{area-name}(\text{addr}) = \text{LR-HEAP-NAME}) \\ &\wedge (\text{offset}(\text{addr}) < (\text{identity}(\text{LR-NODE-SIZE}) * \text{heap-size})) \end{aligned}$$

$$\begin{aligned}
& \wedge \quad (\text{cdr}(\text{assoc}(\text{LR-HEAP-NAME}, \text{data-seg})) \\
& \quad = \quad \text{lr-init-heap-contents}(\text{identity}(\text{tag}(' \mathbf{addr}, \\
& \quad \quad \quad \text{cons}(\text{LR-HEAP-NAME}, 0))), \\
& \quad \quad \quad \text{heap-size})) \\
& \rightarrow \quad (\text{fetch}(\text{add-addr}(\text{addr}, \text{identity}(\text{LR-REF-COUNT-OFFSET})), \text{data-seg}) \\
& \quad = \quad \text{add-addr}(\text{addr}, 4))
\end{aligned}$$

THEOREM: lessp-difference-node-size-sub-addr-3

$$\begin{aligned}
& ((\text{offset}(\text{addr}) < (\text{LR-NODE-SIZE} * \text{heap-size})) \\
& \wedge \quad \text{lr-boundary-nodep}(\text{addr}) \\
& \wedge \quad (\text{offset}(\text{addr}) \in \mathbf{N})) \\
& \rightarrow \quad ((((((\text{identity}(\text{LR-NODE-SIZE}) * \text{heap-size}) - 1) - 1) - 1) - 1) \\
& \quad < \quad \text{offset}(\text{addr})) \\
& \quad = \quad \mathbf{f})
\end{aligned}$$

THEOREM: lr-boundary-nodep-tag-cons-times-lr-node-size

$$\text{lr-boundary-nodep}(\text{tag}(x, \text{cons}(\text{name}, \text{identity}(\text{LR-NODE-SIZE}) * \text{heap-size})))$$

THEOREM: tag-type-name-offset-equal-same

$$\begin{aligned}
& ((\text{type}(\text{addr}) = x) \\
& \wedge \quad (\text{caddr}(\text{addr}) = \mathbf{nil}) \\
& \wedge \quad \text{listp}(\text{untag}(\text{addr})) \\
& \wedge \quad (\text{offset}(\text{addr}) \in \mathbf{N}) \\
& \wedge \quad (\text{area-name}(\text{addr}) = \text{name})) \\
& \rightarrow \quad (\text{tag}(x, \text{cons}(\text{name}, \text{offset}(\text{addr}))) = \text{addr})
\end{aligned}$$

THEOREM: lr-check-free-nodes-lr-free-list-nodes-init-data-seg

$$\begin{aligned}
& \mathbf{let} \ \text{init-data-seg} \ \mathbf{be} \ \text{list}(\text{cons}(\text{area-name}(\text{LR-FP-ADDR}), \text{any1}), \\
& \quad \text{cons}(\text{area-name}(\text{LR-ANSWER-ADDR}), \text{any2}), \\
& \quad \text{cons}(\text{LR-HEAP-NAME}, \\
& \quad \quad \text{lr-init-heap-contents}(\text{tag}(' \mathbf{addr}, \\
& \quad \quad \quad \text{cons}(\text{LR-HEAP-NAME}, \\
& \quad \quad \quad \quad 0)), \\
& \quad \quad \quad \text{heap-size}))) \\
& \mathbf{in}
\end{aligned}$$

$$\begin{aligned}
& ((\text{offset}(\text{max-addr}) \not< \text{offset}(\text{addr})) \\
& \wedge \quad (\text{type}(\text{addr}) = ' \mathbf{addr}) \\
& \wedge \quad (\text{caddr}(\text{addr}) = \mathbf{nil}) \\
& \wedge \quad \text{listp}(\text{addr}) \\
& \wedge \quad \text{listp}(\text{untag}(\text{addr})) \\
& \wedge \quad (\text{offset}(\text{addr}) \in \mathbf{N}) \\
& \wedge \quad \text{lr-boundary-nodep}(\text{addr}) \\
& \wedge \quad (\text{area-name}(\text{addr}) = \text{LR-HEAP-NAME}) \\
& \wedge \quad (\text{max-addr} = \text{lr-max-node}(\text{init-data-seg}))) \\
& \rightarrow \quad \text{lr-check-free-nodes}(\text{addr},
\end{aligned}$$

```

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)
 ^ (caddr (addr) = nil)
 ^ listp (addr)
 ^ adpp (untag (addr), data-seg)
 ^ lr-boundary-nodep (addr)
 ^ (area-name (addr) = LR-HEAP-NAME)
 ^ (type (max-addr) = 'addr)
 ^ (caddr (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)
 ^ (caddr (addr) = nil)
 ^ listp (addr)
 ^ adpp (untag (addr), data-seg)
 ^ lr-boundary-nodep (addr)
 ^ (area-name (addr) = LR-HEAP-NAME)
 ^ (type (max-addr) = 'addr)
 ^ (caddr (max-addr) = nil)

```

$$\begin{aligned}
& \wedge \text{listp}(max\text{-}addr) \\
& \wedge \text{adpp}(\text{untag}(max\text{-}addr), data\text{-}seg) \\
& \wedge \text{lr-boundary-nodep}(max\text{-}addr) \\
& \wedge (\text{area-name}(max\text{-}addr) = \text{LR-HEAP-NAME}) \\
& \wedge \text{lr-node-listp}(node\text{-}list, data\text{-}seg) \\
\rightarrow & (\text{lr-check-free-nodes}(addr1, \\
& \quad node\text{-}list, \\
& \quad \text{deposit-a-list}(\text{list}(a, b, c, d), addr, data\text{-}seg), \\
& \quad max\text{-}addr) \\
& = \text{lr-check-free-nodes}(addr1, \\
& \quad node\text{-}list, \\
& \quad \text{deposit}(b, \\
& \quad \quad \text{add-addr}(addr, \\
& \quad \quad \quad \text{identity}(\text{LR-REF-COUNT-OFFSET})), \\
& \quad \quad data\text{-}seg), \\
& \quad max\text{-}addr))
\end{aligned}$$

THEOREM: lr-all-nodes-not-lessp-min-offset-max-addr
 $(min\text{-}offset \not\leq \text{offset}(max\text{-}addr))$
 $\rightarrow (\text{lr-all-nodes}(min\text{-}offset, max\text{-}addr) = \mathbf{nil})$

THEOREM: fetch-init-init-data-seg-sub-addr
 $((\text{identity}(\text{LR-NODE-SIZE}) * heap\text{-}size) \not\leq \text{offset}(addr))$
 $\wedge (\text{offset}(addr) \in \mathbf{N})$
 $\wedge (\text{caddr}(addr) = \mathbf{nil})$
 $\wedge \text{listp}(addr)$
 $\wedge \text{listp}(\text{untag}(addr))$
 $\wedge (\text{type}(addr) = \mathbf{'addr})$
 $\wedge \text{lr-boundary-nodep}(addr)$
 $\wedge (\text{area-name}(addr) = \text{LR-HEAP-NAME})$
 $\wedge ((\text{offset}(addr) - \text{identity}(\text{LR-NODE-SIZE}))$
 $\quad < (\text{LR-NODE-SIZE} * heap\text{-}size))$
 $\wedge (\text{offset}(addr) \neq 0)$
 $\wedge (\text{cdr}(\text{assoc}(\text{LR-HEAP-NAME}, data\text{-}seg))$
 $\quad = \text{lr-init-heap-contents}(\text{tag}(\mathbf{'addr}, \text{cons}(\text{LR-HEAP-NAME}, 0)),$
 $\quad \quad heap\text{-}size)))$
 $\rightarrow (\text{fetch}(\text{add-addr}(\text{sub-addr}(addr, \text{identity}(\text{LR-NODE-SIZE})),$
 $\quad \quad \text{identity}(\text{LR-REF-COUNT-OFFSET})),$
 $\quad \quad data\text{-}seg)$
 $\quad = addr)$

THEOREM: lr-free-list-nodes-lr-init-heap-contents-generalized
 $(\text{lr-boundary-nodep}(max\text{-}addr))$
 $\wedge (\text{area-name}(max\text{-}addr) = \text{LR-HEAP-NAME})$
 $\wedge (\text{type}(max\text{-}addr) = \mathbf{'addr})$

$$\begin{aligned}
& \wedge \text{ (caddr (max-addr) = nil)} \\
& \wedge \text{ listp (max-addr)} \\
& \wedge \text{ listp (untag (max-addr))} \\
& \wedge \text{ ((LR-NODE-SIZE * heap-size) \not\leq \text{offset (max-addr)})} \\
& \wedge \text{ (cdr (assoc (LR-HEAP-NAME, data-seg))} \\
& \quad = \text{ lr-init-heap-contents (identity (tag ('addr,} \\
& \quad \quad \text{cons (LR-HEAP-NAME, 0))),} \\
& \quad \quad \text{heap-size))}) \\
\rightarrow & \text{ (lr-free-list-nodes (max-addr, data-seg) = lr-all-nodes (0, max-addr))}
\end{aligned}$$

THEOREM: lr-free-list-nodes-lr-init-heap-contents

$$\begin{aligned}
& \text{(lr-boundary-nodep (max-addr)} \\
& \wedge \text{ (area-name (max-addr) = LR-HEAP-NAME)} \\
& \wedge \text{ (type (max-addr) = 'addr)} \\
& \wedge \text{ (caddr (max-addr) = nil)} \\
& \wedge \text{ listp (max-addr)} \\
& \wedge \text{ listp (untag (max-addr))} \\
& \wedge \text{ ((LR-NODE-SIZE * heap-size) \not\leq \text{offset (max-addr)})} \\
\rightarrow & \text{ (lr-free-list-nodes (max-addr,} \\
& \quad \text{list (cons (identity (area-name (LR-FP-ADDR)), any1),} \\
& \quad \text{cons (identity (area-name (LR-ANSWER-ADDR)),} \\
& \quad \quad \text{any2),} \\
& \quad \text{cons (identity (LR-HEAP-NAME),} \\
& \quad \quad \text{lr-init-heap-contents (identity (tag ('addr,} \\
& \quad \quad \quad \text{cons (LR-HEAP-NAME,} \\
& \quad \quad \quad \quad \text{0))),} \\
& \quad \quad \text{heap-size))))) \\
& = \text{ lr-all-nodes (0, max-addr))}
\end{aligned}$$

THEOREM: lr-node-listp-lr-all-nodes

$$\begin{aligned}
& \text{(lr-boundary-nodep (addr)} \\
& \wedge \text{ (area-name (addr) = LR-HEAP-NAME)} \\
& \wedge \text{ adpp (untag (addr), data-seg)} \\
& \wedge \text{ (type (addr) = 'addr)} \\
\rightarrow & \text{ lr-node-listp (lr-all-nodes (min-offset, addr), data-seg)}
\end{aligned}$$

THEOREM: plistp-lr-all-nodes

$$\text{plistp (lr-all-nodes (min-offset, max-addr))}$$

THEOREM: lr-free-list-nodes-lr-init-data-seg

$$\begin{aligned}
& \text{(heap-size \not\leq 2)} \\
\rightarrow & \text{ (lr-free-list-nodes (tag ('addr,} \\
& \quad \text{cons (identity (LR-HEAP-NAME),} \\
& \quad \text{identity (LR-NODE-SIZE) * heap-size)),} \\
& \quad \text{lr-init-data-seg (heap-size))}
\end{aligned}$$

$$= \text{lr-all-nodes}(\text{identity}(\text{offset}(\text{add-addr}(\text{LR-F-ADDR}, \text{LR-NODE-SIZE}))), \\ \text{tag}(' \text{addr}, \\ \text{cons}(\text{identity}(\text{LR-HEAP-NAME}), \\ \text{identity}(\text{LR-NODE-SIZE}) * \text{heap-size}))))$$

THEOREM: lr-proper-free-listp-lr-init-data-seg-helper

$(\text{heap-size} \not\leq 2)$

$$\rightarrow \text{lr-check-free-nodes}(\text{identity}(\text{add-addr}(\text{LR-F-ADDR}, \text{LR-NODE-SIZE})), \\ \text{lr-all-nodes}(\text{identity}(\text{offset}(\text{add-addr}(\text{LR-F-ADDR}, \\ \text{LR-NODE-SIZE}))), \\ \text{tag}(' \text{addr}, \\ \text{cons}(\text{identity}(\text{LR-HEAP-NAME}), \\ \text{identity}(\text{LR-NODE-SIZE}) \\ * \text{heap-size}))), \\ \text{lr-init-data-seg}(\text{heap-size}), \\ \text{tag}(' \text{addr}, \\ \text{cons}(\text{identity}(\text{LR-HEAP-NAME}), \\ \text{identity}(\text{LR-NODE-SIZE}) * \text{heap-size}))))$$

THEOREM: lr-proper-free-listp-lr-init-data-seg

$(\text{heap-size} \not\leq 2) \rightarrow \text{lr-proper-free-listp}(\text{lr-init-data-seg}(\text{heap-size}))$

THEOREM: definedp-lr-heap-name-lr-init-data-seg

$\text{definedp}(\text{identity}(\text{LR-HEAP-NAME}), \text{lr-init-data-seg}(\text{heap-size}))$

THEOREM: lr-proper-p-areasp-lr-heap-name-lr-init-data-seg

$\text{lr-proper-p-areasp}(\text{lr-init-data-seg}(\text{heap-size}))$

THEOREM: lr-proper-p-areasp-car-lr-init-data-seg-table

$\text{lr-proper-p-areasp}(\text{data-seg})$

$\rightarrow \text{lr-proper-p-areasp}(\text{car}(\text{lr-init-data-seg-table}(\text{params}, \text{data-seg}, \text{table})))$

THEOREM: lr-proper-p-areasp-car-lr-data-seg-table-list

$\text{lr-proper-p-areasp}(\text{data-seg})$

$\rightarrow \text{lr-proper-p-areasp}(\text{car}(\text{lr-data-seg-table-list}(\text{progs}, \text{data-seg}, \text{table})))$

THEOREM: definedp-table-definedp-car-lr-init-data-seg-table

$\text{definedp}(\text{name}, \text{car}(\text{lr-init-data-seg-table}(\text{params}, \text{data-seg}, \text{table})))$

$= \text{definedp}(\text{name}, \text{data-seg})$

THEOREM: all-p-objects-lookup-cons-table

$(\text{all-p-objects-lookup}(\text{list}, \text{table}, p) \wedge \text{p-objectp}(y, p))$

$\rightarrow \text{all-p-objects-lookup}(\text{list}, \text{cons}(\text{cons}(x, y), \text{table}), p)$

THEOREM: p-objectp-opener-alt-lr-proper-free-listp
 (lr-proper-free-listp (p-data-segment (*p*))
 $\wedge$ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
 $\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
 $\wedge$ (*data-seg* = p-data-segment (*p*)))
 $\rightarrow$ p-objectp (fetch (identity (LR-FP-ADDR), *data-seg*), *p*)

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))
 → 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))
 → 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)
→ (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)))
∧ 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-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))
 $\wedge$ 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*)))
 $\rightarrow$ 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)
       $\wedge$  (caddr (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))

```


→ 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*)
 ∧ lr-proper-p-areasp (*data-seg*)
 ∧ definedp (LR-HEAP-NAME, *data-seg*)
 ∧ lr-boundary-nodep (lr-max-node (*data-seg*)))
 → adpp (untag (fetch (identity (LR-FP-ADDR),
 car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*))))),
 data-seg)

DEFINITION:
 lr-good-pointer-tablep (*table*, *data-seg*)
 = **if** listp (*table*)
 then lr-good-pointerp (cdar (*table*), *data-seg*)
 ∧ lr-good-pointer-tablep (cdr (*table*), *data-seg*)
 else t endif

THEOREM: lr-good-pointer-tablep-definedp-table
 (lr-good-pointer-tablep (*table*, *data-seg*) ∧ definedp (*object*, *table*))
 → 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*)
 ∧ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
 ∧ lr-boundary-nodep (lr-max-node (*data-seg*)))
 → (car (untag (fetch (identity (LR-FP-ADDR), *data-seg*)))) = LR-HEAP-NAME)

THEOREM: lr-good-pointer-tablep-deposit-free-ptr
 lr-good-pointer-tablep (*table*,
 deposit (*anything*, identity (LR-FP-ADDR), *data-seg*))
 = lr-good-pointer-tablep (*table*, *data-seg*)

THEOREM: add1-lr-boundary-nodep
 (lr-boundary-nodep (*addr1*) ∧ lr-boundary-nodep (*addr2*))
 → ((offset (*addr1*) = (1 + offset (*addr2*))) = **f**)

THEOREM: lr-boundary-offsetp-plus
 lr-boundary-offsetp (*n*)
 → (lr-boundary-offsetp (*m* + *n*) = lr-boundary-offsetp (*m*))

THEOREM: add1-add1-lr-boundary-nodep
 (lr-boundary-nodep (*addr1*) ∧ lr-boundary-nodep (*addr2*))
 → ((offset (*addr1*) = (1 + (1 + offset (*addr2*)))) = **f**)

THEOREM: lr-good-pointerp-tablep-deposit-a-list
 (lr-good-pointerp-tablep (*table*, *data-seg*)
 $\wedge$ (offset (*addr*) $\in \mathbf{N}$)
 $\wedge$ lr-boundary-nodep (*addr*)
 $\wedge$ (type (*ref-count*) = 'nat)
 $\wedge$ (untag (*ref-count*) $\in \mathbf{N}$)
 $\wedge$ (area-name (*addr*) = LR-HEAP-NAME)
 $\wedge$ (type (*tag*) = 'nat))
 $\rightarrow$ lr-good-pointerp-tablep (*table*,
 deposit-a-list (list (*tag*, *ref-count*, *x*, *y*),
 addr,
 data-seg))

THEOREM: lr-good-pointerp-table-cons
 ((type (*addr*) = 'addr)
 $\wedge$ (caddr (*addr*) = nil)
 $\wedge$ listp (*addr*)
 $\wedge$ adpp (untag (*addr*), *data-seg*)
 $\wedge$ lr-boundary-nodep (*addr*)
 $\wedge$ (area-name (*addr*) = LR-HEAP-NAME))
 $\rightarrow$ (lr-good-pointerp-tablep (cons (cons (*object*, *addr*), *table*), *data-seg*)
 = ((type (fetch (add-addr (*addr*, identity (LR-REF-COUNT-OFFSET)),
 data-seg))
 = 'nat)
 $\wedge$ lr-good-pointerp-tablep (*table*, *data-seg*)))

THEOREM: lr-proper-free-listp-length-sub1-not-lessp
 (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*)))
 $\rightarrow$ ((length (cdr (assoc (identity (LR-HEAP-NAME), *data-seg*))) - 1)
 $\not\leq$ offset (fetch (identity (LR-FP-ADDR),
 car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*))))))

THEOREM: lr-minimum-heapp-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-minimum-heapp (car (lr-compile-quote (*flag*, *object*, *data-seg*, *table*)))
 = lr-minimum-heapp (*data-seg*))

DEFINITION:
 s-heap-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
            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),
                              data-seg, table))

    else 0 endif

```

```

                                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* $\in \mathbf{N}$ **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*) $\not\prec$ *lr-count-free-nodes* (*addr*, *node-list*, *data-seg*)

THEOREM: lr-proper-free-listp-lr-count-free-nodes-max-addr
 (lr-proper-free-listp (*data-seg*)
 $\wedge$ definedp (LR-HEAP-NAME, *data-seg*)
 $\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
 $\wedge$ lr-proper-p-areasp (*data-seg*)
 $\wedge$ (length (cdr (assoc (LR-HEAP-NAME, *data-seg*)))
 $=$ (1 + offset (fetch (LR-FP-ADDR, *data-seg*))))
 $\wedge$ (*max-addr* = lr-max-node (*data-seg*))
 $\rightarrow$ (lr-count-free-nodes (fetch (identity (LR-FP-ADDR), *data-seg*),
 lr-free-list-nodes (*max-addr*, *data-seg*),
data-seg)
 $=$ 0)

EVENT: Disable lr-count-free-nodes-at-most.

THEOREM: lr-count-free-nodes-deposit-a-list-lr-nodep
 ((type (*addr1*) = 'addr)
 $\wedge$ (caddr (*addr1*) = nil)
 $\wedge$ listp (*addr1*)
 $\wedge$ adpp (untag (*addr1*), *data-seg*)
 $\wedge$ lr-boundary-nodep (*addr1*)
 $\wedge$ (area-name (*addr1*) = LR-HEAP-NAME)
 $\wedge$ (type (*ref-count*) = 'nat)
 $\wedge$ (untag (*ref-count*) $\in \mathbf{N}$)
 $\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
 $\wedge$ adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
 $\wedge$ lr-node-listp (*node-list*, *data-seg*)
 $\wedge$ (*addr1* $\notin$ *node-list*)
 $\rightarrow$ (lr-count-free-nodes (*addr2*,
node-list,
 deposit-a-list (list (*x*, *ref-count*, *y*, *z*),
addr1,
data-seg))
 $=$ lr-count-free-nodes (*addr2*, delete (*addr1*, *node-list*), *data-seg*))

THEOREM: lr-proper-free-listp-member-free-addr-lr-free-list-nodes
 (adpp (untag (lr-max-node (*data-seg*)), *data-seg*)
 $\wedge$ lr-boundary-nodep (lr-max-node (*data-seg*))
 $\wedge$ (*max-addr* = lr-max-node (*data-seg*))
 $\wedge$ (offset (fetch (LR-FP-ADDR, *data-seg*))
 $<$ (length (cdr (assoc (LR-HEAP-NAME, *data-seg*))) - 1))
 $\wedge$ lr-proper-free-listp (*data-seg*)
 $\rightarrow$ (fetch (identity (LR-FP-ADDR), *data-seg*)
 $\in$ lr-free-list-nodes (*max-addr*, *data-seg*))

THEOREM: lr-count-free-nodes-lr-compile-quote-s-heap-reqs-help1

$$\begin{aligned}
& (\text{adpp} (\text{untag} (\text{lr-max-node} (data-seg)), data-seg) \\
& \wedge \text{lr-boundary-nodep} (\text{lr-max-node} (data-seg)) \\
& \wedge (\text{offset} (\text{fetch} (\text{LR-FP-ADDR}, data-seg)) \\
& \quad < (\text{length} (\text{cdr} (\text{assoc} (\text{LR-HEAP-NAME}, data-seg))) - 1)) \\
& \wedge \text{lr-proper-free-listp} (data-seg) \\
& \wedge \text{lr-proper-p-areasp} (data-seg) \\
& \wedge (max-addr = \text{lr-max-node} (data-seg))) \\
\rightarrow & (\text{lr-count-free-nodes} (\text{fetch} (\text{identity} (\text{LR-FP-ADDR}), data-seg), \\
& \quad \text{lr-free-list-nodes} (max-addr, data-seg), \\
& \quad data-seg) \\
& = (1 + \text{lr-count-free-nodes} (\text{fetch} (\text{add-addr} (\text{fetch} (\text{identity} (\text{LR-FP-ADDR}), \\
& \quad data-seg), \\
& \quad \text{identity} (\text{LR-REF-COUNT-OFFSET})), \\
& \quad data-seg), \\
& \quad \text{delete} (\text{fetch} (\text{identity} (\text{LR-FP-ADDR}), \\
& \quad data-seg), \\
& \quad \text{lr-free-list-nodes} (max-addr, \\
& \quad data-seg)), \\
& \quad data-seg)))
\end{aligned}$$

THEOREM: lr-proper-free-listp-lr-count-free-nodes-max-addr-alt

$$\begin{aligned}
& (\text{lr-proper-free-listp} (data-seg) \\
& \wedge \text{definedp} (\text{LR-HEAP-NAME}, data-seg) \\
& \wedge \text{lr-boundary-nodep} (\text{lr-max-node} (data-seg)) \\
& \wedge \text{lr-proper-p-areasp} (data-seg) \\
& \wedge (\text{offset} (\text{fetch} (\text{LR-FP-ADDR}, data-seg)) \\
& \quad \not< (\text{length} (\text{cdr} (\text{assoc} (\text{LR-HEAP-NAME}, data-seg))) - 1)) \\
& \wedge (max-addr = \text{lr-max-node} (data-seg))) \\
\rightarrow & (\text{lr-count-free-nodes} (\text{fetch} (\text{identity} (\text{LR-FP-ADDR}), data-seg), \\
& \quad \text{lr-free-list-nodes} (max-addr, data-seg), \\
& \quad data-seg) \\
& = 0)
\end{aligned}$$

THEOREM: s-heap-reqs-object-t

$$\begin{aligned}
& ((flag \neq \text{'list}) \wedge (\neg \text{definedp} (t, table))) \\
\rightarrow & (\text{s-heap-reqs} (flag, t, data-seg, table) = 1)
\end{aligned}$$

THEOREM: lessp-lr-boundary-offsetp-nodep-plus-node-size-fact-1

$$\begin{aligned}
& (\text{lr-boundary-offsetp} (offset) \wedge \text{lr-boundary-nodep} (addr)) \\
\rightarrow & ((offset < (\text{identity} (\text{LR-NODE-SIZE}) + \text{offset} (addr))) \\
& = (\text{offset} (addr) \not< offset))
\end{aligned}$$

THEOREM: lr-count-free-nodes-lr-compile-quote-s-heap-reqs

$$(\text{lr-proper-free-listp} (data-seg))$$

```

 $\wedge$  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)
                              $\not\leq$  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))),
                                                         data-seg),
                                                         data-seg))
                                + 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)
                                 $\not\leq$  (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))
                                 $\not\leq$  x) endlet

```

THEOREM: s-heap-reqs-flag-list-nil-opener
 $\text{s-heap-reqs}('list, \mathbf{nil}, data-seg, table) = 0$

THEOREM: lr-compile-quote-flag-list-nil-opener
 $\text{lr-compile-quote}('list, \mathbf{nil}, data-seg, table) = \text{cons}(data-seg, table)$

THEOREM: s-heap-reqs-flag-list-cons-opener
 $\text{s-heap-reqs}('list, \text{cons}(x, y), data-seg, table)$
 $= (\text{s-heap-reqs}(\mathbf{t}, x, data-seg, table)$
 $\quad + \text{s-heap-reqs}('list,$
 $\quad \quad y,$
 $\quad \quad \text{car}(\text{lr-compile-quote}(\mathbf{t}, x, data-seg, table)),$
 $\quad \quad \text{cdr}(\text{lr-compile-quote}(\mathbf{t}, x, data-seg, table))))$

THEOREM: lr-compile-quote-flag-list-cons-opener
 $\text{lr-compile-quote}('list, \text{cons}(x, y), data-seg, table)$
 $= \text{lr-compile-quote}('list,$
 $\quad y,$
 $\quad \text{car}(\text{lr-compile-quote}(\mathbf{t}, x, data-seg, table)),$
 $\quad \text{cdr}(\text{lr-compile-quote}(\mathbf{t}, x, data-seg, table)))$

THEOREM: lr-compile-quote-lr-good-pointer-tablep-help-2
 $(\text{lr-proper-free-listp}(data-seg)$
 $\quad \wedge \text{lr-proper-p-areasp}(data-seg)$
 $\quad \wedge \text{definedp}(\text{LR-HEAP-NAME}, data-seg)$
 $\quad \wedge \text{lr-boundary-nodep}(\text{lr-max-node}(data-seg))$
 $\quad \wedge \text{lr-minimum-heapp}(data-seg)$
 $\quad \wedge (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, data-seg),$
 $\quad \quad \text{lr-free-list-nodes}(\text{lr-max-node}(data-seg),$
 $\quad \quad \quad data-seg),$
 $\quad \quad \quad data-seg)$
 $\quad \neq 0)$
 $\quad \wedge (\text{lr-count-free-nodes}(\text{fetch}(\text{add-addr}(\text{fetch}(\text{LR-FP-ADDR}, data-seg),$
 $\quad \quad \quad \text{LR-REF-COUNT-OFFSET}),$
 $\quad \quad \quad data-seg),$
 $\quad \quad \text{delete}(\text{fetch}(\text{LR-FP-ADDR}, data-seg),$
 $\quad \quad \quad \text{lr-free-list-nodes}(\text{lr-max-node}(data-seg),$
 $\quad \quad \quad \quad data-seg)),$
 $\quad \quad \quad data-seg)$
 $\quad \not\leq \text{s-heap-reqs}('list,$
 $\quad \quad \text{list}(\text{car}(object), \text{cdr}(object)),$
 $\quad \quad \quad data-seg,$
 $\quad \quad \quad table))$
 $\quad \wedge \text{listp}(object))$
 $\rightarrow ((\text{length}(\text{cdr}(\text{assoc}(\text{identity}(\text{LR-HEAP-NAME}), data-seg)))) - 1)$

```

✗   (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

$$\begin{aligned} & (\text{lr-proper-free-listp } (data-seg) \\ & \wedge \text{ lr-proper-p-areasp } (data-seg) \\ & \wedge \text{ definedp } (\text{LR-HEAP-NAME}, data-seg) \\ & \wedge \text{ lr-boundary-noddep } (\text{lr-max-node } (data-seg)) \\ & \wedge \text{ lr-minimum-heapp } (data-seg) \\ & \wedge (\text{lr-count-free-nodes } (\text{fetch } (\text{LR-FP-ADDR}, data-seg), \\ & \qquad \qquad \qquad \text{lr-free-list-nodes } (\text{lr-max-node } (data-seg), \\ & \qquad \qquad \qquad \qquad \qquad \qquad data-seg), \\ & \qquad \qquad \qquad \qquad \qquad \qquad data-seg) \\ & \quad \not\leq 1)) \\ \rightarrow & ((\text{length } (\text{cdr } (\text{assoc } (\text{identity } (\text{LR-HEAP-NAME}), data-seg))) - 1) \\ & \quad \not\leq (\text{offset } (\text{fetch } (\text{identity } (\text{LR-FP-ADDR}), data-seg)) \\ & \quad \quad + \text{identity } (\text{LR-NODE-SIZE}))) \end{aligned}$$

THEOREM: lr-compile-quote-lr-good-pointerp-tablep

[illegible]

data-seg,
table)),
car (lr-compile-quote (*flag,*
object,
data-seg,
table)))

THEOREM: lr-noddep-car-lr-compile-quote
lr-noddep (*addr, data-seg*)
→ lr-noddep (*addr, car (lr-compile-quote (flag, object, data-seg, table)))*)

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

THEOREM: lr-noddep-deposit-a-list
lr-noddep (*addr1, data-seg*)
→ lr-noddep (*addr1, deposit-a-list (list, addr2, data-seg)*)

DEFINITION:
induct-hint-16 (*object, list, data-seg, table*)
= **if** *list* $\simeq$ **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* $\in$ *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*)

```

^ lr-boundary-nodep (lr-max-node (data-seg))
^ lr-minimum-heapp (data-seg)
^ lr-good-pointerp-tablep (table, data-seg)
^ s-restricted-objectp ('list, object-list)
^ (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 (LR-FP-ADDR, data-seg),
                             lr-free-list-nodes (lr-max-node (data-seg),
                                                    data-seg),
                             data-seg) - 1)
   ✗ s-heap-reqs ('list, object-list, data-seg, table))
^ definedp (f, table)
^ (object ∈ object-list))
→ 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)
^ adpp (untag (addr), data-seg)
^ (area-name (addr) = 'heap)
^ (type (addr) = 'addr)
^ (caddr (addr) = nil)
^ listp (addr)
^ (n < LR-NODE-SIZE)
^ (n ≠ LR-REF-COUNT-OFFSET)
^ definedp (LR-HEAP-NAME, data-seg)
^ lr-good-pointerp (good-pointer, data-seg)
^ (type (fetch (add-addr (addr, LR-REF-COUNT-OFFSET), data-seg)) ≠ 'nat))
→ 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)
^ adpp (untag (addr), data-seg)
^ (area-name (addr) = 'heap)

```

$\wedge$ (type(*addr*) = 'addr)
 $\wedge$ (caddr(*addr*) = nil)
 $\wedge$ listp(*addr*)
 $\wedge$ (type(*x*) = 'nat)
 $\wedge$ 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*, identity(LR-REF-COUNT-OFFSET)),
data-seg))

THEOREM: lr-good-pointerp-deposit-non-add-addr-not-good-pointerp

(lr-boundary-nodep(*addr*)
 $\wedge$ adpp(untag(*addr*), *data-seg*)
 $\wedge$ (area-name(*addr*) = 'heap)
 $\wedge$ (type(*addr*) = 'addr)
 $\wedge$ (caddr(*addr*) = nil)
 $\wedge$ listp(*addr*)
 $\wedge$ 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*, *addr*, *data-seg*))

THEOREM: lr-check-numberp-addrp-deposit-a-list-cons-same-addr

((type(*addr*) = 'addr)
 $\wedge$ (caddr(*addr*) = nil)
 $\wedge$ listp(*addr*)
 $\wedge$ adpp(untag(*addr*), *data-seg*)
 $\wedge$ lr-boundary-nodep(*addr*)
 $\wedge$ (area-name(*addr*) = 'heap)
 $\wedge$ (offset(*addr*) $\not\leq$ (LR-NODE-SIZE + offset(LR-F-ADDR)))
 $\wedge$ (type(fetch(add-addr(*addr*, LR-REF-COUNT-OFFSET), *data-seg*))
= 'addr)
 $\wedge$ lr-good-pointerp(*good-pointer*, *data-seg*)
 $\wedge$ definedp('heap, *data-seg*)
 $\wedge$ lr-proper-p-areasp(*data-seg*)
 $\wedge$ (type(*ref-count*) = 'nat)
 $\wedge$ (untag(*ref-count*) $\in \mathbf{N}$)
 $\wedge$ (type(*tagged-number*) = 'nat)
 $\wedge$ (untag(*tagged-number*) $\in \mathbf{N}$))
 $\rightarrow$ lr-check-numberp-addrp(*addr*,
deposit-a-list(list(identity(tag('nat,
LR-ADD1-TAG)),

$ref\text{-}count,$
 $tagged\text{-}number,$
 $good\text{-}pointer),$
 $addr,$
 $data\text{-}seg))$

THEOREM: lr-proper-heapp-nodep-deposit-a-list-numberp

$(lr\text{-}nodep\ (max\text{-}addr,\ data\text{-}seg)$
 $\wedge\ lr\text{-}nodep\ (addr,\ data\text{-}seg)$
 $\wedge\ lr\text{-}proper\text{-}heapp\text{-}nodep\ (max\text{-}addr,\ data\text{-}seg)$
 $\wedge\ (type\ (fetch\ (add\text{-}addr\ (addr,\ LR\text{-}REF\text{-}COUNT\text{-}OFFSET),\ data\text{-}seg)) \neq 'nat)$
 $\wedge\ lr\text{-}good\text{-}pointerp\ (good\text{-}pointer,\ data\text{-}seg)$
 $\wedge\ adpp\ (untag\ (lr\text{-}max\text{-}node\ (data\text{-}seg)),\ data\text{-}seg)$
 $\wedge\ lr\text{-}proper\text{-}p\text{-}areasp\ (data\text{-}seg)$
 $\wedge\ (type\ (ref\text{-}count) = 'nat)$
 $\wedge\ (untag\ (ref\text{-}count) \in \mathbf{N})$
 $\wedge\ (type\ (tagged\text{-}number) = 'nat)$
 $\wedge\ (untag\ (tagged\text{-}number) \in \mathbf{N}))$
 $\rightarrow\ lr\text{-}proper\text{-}heapp\text{-}nodep\ (max\text{-}addr,$
 $\quad\quad\quad deposit\text{-}a\text{-}list\ (list\ (identity\ (tag\ ('nat,$
 $\quad\quad\quad\quad\quad\quad LR\text{-}ADD1\text{-}TAG)),$
 $\quad\quad\quad\quad\quad\quad ref\text{-}count,$
 $\quad\quad\quad\quad\quad\quad tagged\text{-}number,$
 $\quad\quad\quad\quad\quad\quad good\text{-}pointer),$
 $\quad\quad\quad addr,$
 $\quad\quad\quad data\text{-}seg))$

THEOREM: lr-proper-heapp2-deposit-a-list-numberp

$(lr\text{-}nodep\ (max\text{-}addr,\ data\text{-}seg)$
 $\wedge\ lr\text{-}nodep\ (addr,\ data\text{-}seg)$
 $\wedge\ lr\text{-}proper\text{-}heapp2\ (max\text{-}addr,\ data\text{-}seg)$
 $\wedge\ (type\ (fetch\ (add\text{-}addr\ (addr,\ LR\text{-}REF\text{-}COUNT\text{-}OFFSET),\ data\text{-}seg)) \neq 'nat)$
 $\wedge\ lr\text{-}good\text{-}pointerp\ (good\text{-}pointer,\ data\text{-}seg)$
 $\wedge\ adpp\ (untag\ (lr\text{-}max\text{-}node\ (data\text{-}seg)),\ data\text{-}seg)$
 $\wedge\ lr\text{-}proper\text{-}p\text{-}areasp\ (data\text{-}seg)$
 $\wedge\ (type\ (tag) = 'nat)$
 $\wedge\ (untag\ (tag) \in \mathbf{N})$
 $\wedge\ (type\ (tagged\text{-}number) = 'nat)$
 $\wedge\ (untag\ (tagged\text{-}number) \in \mathbf{N}))$
 $\rightarrow\ lr\text{-}proper\text{-}heapp2\ (max\text{-}addr,$
 $\quad\quad\quad deposit\text{-}a\text{-}list\ (list\ (identity\ (tag\ ('nat,\ LR\text{-}ADD1\text{-}TAG)),$
 $\quad\quad\quad\quad\quad\quad tag,$
 $\quad\quad\quad\quad\quad\quad tagged\text{-}number,$
 $\quad\quad\quad\quad\quad\quad good\text{-}pointer),$

$$addr,$$

$$data-seg))$$
$$\begin{aligned} & \text{THEOREM: lr-good-pointerp-lr-undef-addr} \\ & (\text{lr-minimum-heapp} (data-seg) \\ & \wedge \text{lr-proper-p-areasp} (data-seg) \\ & \wedge \text{lr-proper-heapp2} (\text{lr-max-node} (data-seg), data-seg) \\ & \wedge \text{lr-nodep} (\text{lr-max-node} (data-seg), data-seg)) \\ & \rightarrow \text{lr-good-pointerp} (\text{identity} (\text{LR-UNDEF-ADDR}), data-seg) \end{aligned}$$
$$\begin{array}{l}
\text{THEOREM: lr-proper-heapp-nodep-deposit-a-list-truep} \\
(\text{lr-nodep } (max\text{-}addr, data\text{-}seg) \\
\wedge \text{ lr-nodep } (addr, data\text{-}seg) \\
\wedge \text{ lr-proper-heapp-nodep } (max\text{-}addr, data\text{-}seg) \\
\wedge (\text{type } (\text{fetch } (\text{add-addr } (addr, \text{LR-REF-COUNT-OFFSET}), data\text{-}seg)) \neq \text{'nat}) \\
\wedge \text{ lr-good-pointerp } (good\text{-}pointer1, data\text{-}seg) \\
\wedge \text{ lr-good-pointerp } (good\text{-}pointer2, data\text{-}seg) \\
\wedge \text{ adpp } (\text{untag } (\text{lr-max-node } (data\text{-}seg)), data\text{-}seg) \\
\wedge \text{ lr-proper-p-areaasp } (data\text{-}seg) \\
\wedge (\text{type } (ref\text{-}count) = \text{'nat}) \\
\wedge (\text{untag } (ref\text{-}count) \in \mathbf{N})) \\
\rightarrow \text{lr-proper-heapp-nodep } (max\text{-}addr, \\
\hspace{15em} \text{deposit-a-list } (\text{list } (\text{identity } (\text{tag } (\text{'nat}, \\
\hspace{15em} \text{LR-TRUE-TAG})), \\
\hspace{15em} ref\text{-}count, \\
\hspace{15em} good\text{-}pointer1, \\
\hspace{15em} good\text{-}pointer2), \\
\hspace{15em} addr, \\
\hspace{15em} data\text{-}seg)))
\end{array}$$
$$\begin{aligned}
& \text{THEOREM: lr-proper-heapp2-deposit-a-list-truep} \\
& (\text{lr-nodep}(\text{max-addr}, \text{data-seg}) \\
& \quad \wedge \text{lr-nodep}(\text{addr}, \text{data-seg}) \\
& \quad \wedge \text{lr-proper-heapp2}(\text{max-addr}, \text{data-seg}) \\
& \quad \wedge (\text{type}(\text{fetch}(\text{add-addr}(\text{addr}, \text{LR-REF-COUNT-OFFSET}), \text{data-seg})) \neq \text{'nat'}) \\
& \quad \wedge \text{lr-good-pointerp}(\text{good-pointer1}, \text{data-seg}) \\
& \quad \wedge \text{lr-good-pointerp}(\text{good-pointer2}, \text{data-seg}) \\
& \quad \wedge \text{adpp}(\text{untag}(\text{lr-max-node}(\text{data-seg})), \text{data-seg}) \\
& \quad \wedge \text{lr-proper-p-areasp}(\text{data-seg}) \\
& \quad \wedge (\text{type}(\text{tag}) = \text{'nat'}) \\
& \quad \wedge (\text{untag}(\text{tag}) \in \mathbf{N})) \\
& \rightarrow \text{lr-proper-heapp2}(\text{max-addr}, \\
& \quad \text{deposit-a-list}(\text{list}(\text{identity}(\text{tag}(\text{'nat'}, \text{LR-TRUE-TAG})), \\
& \quad \quad \text{tag}, \\
& \quad \quad \text{good-pointer1},
\end{aligned}$$

$good_pointer2),$
 $addr,$
 $data-seg))$

THEOREM: lr-compile-quote-preserves-lr-proper-heapp2

$(lr_proper_free_listp\ (data-seg)$
 $\wedge\ lr_proper_p_areasp\ (data-seg)$
 $\wedge\ lr_nodep\ (lr_max_node\ (data-seg),\ data-seg)$
 $\wedge\ definedp\ (LR-HEAP-NAME,\ data-seg)$
 $\wedge\ lr_boundary_nodep\ (lr_max_node\ (data-seg))$
 $\wedge\ lr_good_pointerp_tablep\ (table,\ data-seg)$
 $\wedge\ s_restricted_objectp\ (flag,\ object)$
 $\wedge\ lr_minimum_heapp\ (data-seg)$
 $\wedge\ lr_proper_heapp2\ (lr_max_node\ (data-seg),\ data-seg)$
 $\wedge\ definedp\ (f,\ table)$
 $\wedge\ (lr_count_free_nodes\ (fetch\ (LR-FP-ADDR,\ data-seg),$
 $\quad\quad\quad lr_free_list_nodes\ (lr_max_node\ (data-seg),$
 $\quad\quad\quad data-seg),$
 $\quad\quad\quad data-seg)$
 $\quad\quad\quad \not\wedge\ s_heap_reqs\ (flag,\ object,\ data-seg,\ table)))$
 $\rightarrow\ lr_proper_heapp2\ (lr_max_node\ (data-seg),$
 $\quad\quad\quad car\ (lr_compile_quote\ (flag,\ object,\ data-seg,\ table)))$

THEOREM: plist-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

$lr_s_similar_const_table\ (table,\ data-seg)$
 $\rightarrow\ lr_good_pointerp_tablep\ (table,\ data-seg)$

THEOREM: lr-s-similar-const-table-deposit-cons

$(lr_proper_heapp\ (data-seg)$
 $\wedge\ lr_s_similar_const_table\ (table,\ data-seg)$
 $\wedge\ lr_proper_p_areasp\ (data-seg))$
 $\rightarrow\ lr_s_similar_const_table\ (table,$
 $\quad\quad\quad deposit_a_list\ (list\ (x0,\ x1,\ x2,\ x3),$
 $\quad\quad\quad fetch\ (identity\ (LR-FP-ADDR),$
 $\quad\quad\quad data-seg),$
 $\quad\quad\quad data-seg))$

THEOREM: lr-valp-deposit-a-list-cons-cons

```

(listp (object)
  ∧ lr-proper-p-areasp (data-seg)
  ∧ lr-proper-heapp (data-seg)
  ∧ lr-s-similar-const-table (table, data-seg)
  ∧ (type (ref-count) = 'nat)
  ∧ (untag (ref-count) ∈ ℕ)
  ∧ definedp (car (object), table)
  ∧ definedp (cdr (object), table)
  ∧ definedp (LR-HEAP-NAME, data-seg)
  ∧ (addr = fetch (identity (LR-FP-ADDR), data-seg)))
→ lr-valp (object,
  addr,
  deposit-a-list (list (identity (tag ('nat, LR-CONS-TAG)),
    ref-count,
    cdr (assoc (car (object), table)),
    cdr (assoc (cdr (object), table))),
  addr,
  data-seg))

```

THEOREM: lr-valp-deposit-a-list-cons-numberp

```

((object ∈ ℕ)
  ∧ lr-proper-p-areasp (data-seg)
  ∧ lr-proper-heapp (data-seg)
  ∧ (type (ref-count) = 'nat)
  ∧ (untag (ref-count) ∈ ℕ)
  ∧ definedp (LR-HEAP-NAME, data-seg)
  ∧ (addr = fetch (identity (LR-FP-ADDR), data-seg)))
→ lr-valp (object,
  addr,
  deposit-a-list (list (identity (tag ('nat, LR-ADD1-TAG)),
    ref-count,
    tag ('nat, object),
    identity (LR-UNDEF-ADDR)),
  addr,
  data-seg))

```

THEOREM: lr-compile-quote-preserves-lr-valp

```

(lr-proper-p-areasp (data-seg)
  ∧ lr-good-pointer-tablep (table, data-seg)
  ∧ s-restricted-objectp (flag, object)
  ∧ lr-minimum-heapp (data-seg)
  ∧ lr-nodep (lr-max-node (data-seg), data-seg)
  ∧ lr-proper-free-listp (data-seg)

```

$$\begin{aligned}
& \wedge \text{lr-proper-heapp2}(\text{lr-max-node}(\text{data-seg}), \text{data-seg}) \\
& \wedge \text{definedp}(\mathbf{f}, \text{table}) \\
& \wedge (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{data-seg}), \\
& \quad \quad \text{data-seg}) \\
& \quad \not\wedge \text{s-heap-reqs}(\text{flag}, \text{object}, \text{data-seg}, \text{table})) \\
& \wedge \text{lr-valp}(\text{value}, \text{addr}, \text{data-seg})) \\
& \rightarrow \text{lr-valp}(\text{value}, \text{addr}, \text{car}(\text{lr-compile-quote}(\text{flag}, \text{object}, \text{data-seg}, \text{table})))
\end{aligned}$$

THEOREM: lr-compile-quote-preserves-lr-proper-heapp

$$\begin{aligned}
& (\text{lr-proper-p-areasp}(\text{data-seg}) \\
& \wedge \text{lr-good-pointer-tablep}(\text{table}, \text{data-seg}) \\
& \wedge \text{lr-proper-heapp}(\text{data-seg}) \\
& \wedge \text{s-restricted-objectp}(\text{flag}, \text{object}) \\
& \wedge \text{definedp}(\mathbf{f}, \text{table}) \\
& \wedge (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{data-seg}), \\
& \quad \quad \text{data-seg}) \\
& \quad \not\wedge \text{s-heap-reqs}(\text{flag}, \text{object}, \text{data-seg}, \text{table}))) \\
& \rightarrow \text{lr-proper-heapp}(\text{car}(\text{lr-compile-quote}(\text{flag}, \text{object}, \text{data-seg}, \text{table})))
\end{aligned}$$

THEOREM: lr-valp-deposit-a-list-cons-truep

$$\begin{aligned}
& (\text{lr-proper-p-areasp}(\text{data-seg}) \\
& \wedge \text{lr-proper-heapp}(\text{data-seg}) \\
& \wedge (\text{type}(\text{ref-count}) = \text{'nat}) \\
& \wedge (\text{untag}(\text{ref-count}) \in \mathbf{N}) \\
& \wedge \text{definedp}(\text{LR-HEAP-NAME}, \text{data-seg}) \\
& \wedge (\text{addr} = \text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \text{data-seg}))) \\
& \rightarrow \text{lr-valp}(\mathbf{t}, \\
& \quad \text{addr}, \\
& \quad \text{deposit-a-list}(\text{list}(\text{identity}(\text{tag}(\text{'nat}, \text{LR-TRUE-TAG})), \\
& \quad \quad \text{ref-count}, \\
& \quad \quad \text{identity}(\text{LR-UNDEF-ADDR}), \\
& \quad \quad \text{identity}(\text{LR-UNDEF-ADDR})), \\
& \quad \text{addr}, \\
& \quad \text{data-seg}))
\end{aligned}$$

THEOREM: lr-s-similar-const-table-lr-compile-quote

$$\begin{aligned}
& (\text{lr-proper-p-areasp}(\text{data-seg}) \\
& \wedge \text{lr-proper-heapp}(\text{data-seg}) \\
& \wedge \text{lr-s-similar-const-table}(\text{table}, \text{data-seg}) \\
& \wedge \text{s-restricted-objectp}(\text{flag}, \text{object}) \\
& \wedge \text{definedp}(\mathbf{f}, \text{table})
\end{aligned}$$

$$\begin{aligned}
& \wedge \quad (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{data-seg}), \\
& \quad \quad \text{data-seg}) \\
& \quad \not\leq \text{s-heap-reqs}(\text{flag}, \text{object}, \text{data-seg}, \text{table})) \\
\rightarrow & \quad \text{lr-s-similar-const-table}(\text{cdr}(\text{lr-compile-quote}(\text{flag}, \\
& \quad \text{object}, \\
& \quad \text{data-seg}, \\
& \quad \text{table})), \\
& \quad \text{car}(\text{lr-compile-quote}(\text{flag}, \\
& \quad \text{object}, \\
& \quad \text{data-seg}, \\
& \quad \text{table})))
\end{aligned}$$

THEOREM: p-objectp-cdr-assoc-car-lr-compile-quote

$$\begin{aligned}
& (\text{lr-proper-p-areasp}(\text{data-seg}) \\
& \wedge \quad \text{lr-proper-heapp}(\text{data-seg}) \\
& \wedge \quad \text{lr-s-similar-const-table}(\text{table}, \text{data-seg}) \\
& \wedge \quad \text{s-restricted-objectp}(\text{flag}, \text{object}) \\
& \wedge \quad (\text{word-size} \not\leq \text{s-ws-reqs}(\text{flag}, \text{object}, \text{data-seg}, \text{table})) \\
& \wedge \quad \text{definedp}(\text{f}, \text{table}) \\
& \wedge \quad (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{data-seg}), \\
& \quad \quad \text{data-seg}) \\
& \quad \not\leq \text{s-heap-reqs}(\text{flag}, \text{object}, \text{data-seg}, \text{table})) \\
& \wedge \quad (\text{flag} \neq \text{'list})) \\
\rightarrow & \quad \text{p-objectp}(\text{cdr}(\text{assoc}(\text{object}, \\
& \quad \text{cdr}(\text{lr-compile-quote}(\text{flag}, \text{object}, \text{data-seg}, \text{table})))), \\
& \quad \text{p-state}(\text{pc}, \\
& \quad \quad \text{ctrl-stk}, \\
& \quad \quad \text{temp-stk}, \\
& \quad \quad \text{prog-seg}, \\
& \quad \quad \text{car}(\text{lr-compile-quote}(\text{flag}, \text{object}, \text{data-seg}, \text{table})), \\
& \quad \quad \text{max-ctrl-stk-size}, \\
& \quad \quad \text{max-temp-stk-size}, \\
& \quad \quad \text{word-size}, \\
& \quad \quad \text{psw}))
\end{aligned}$$

THEOREM: lr-count-free-nodes-s-init-heap-reqs

let *ccar* **be** $\text{lr-compile-quote}(\mathbf{t}, \text{object}, \text{data-seg}, \text{table})$

in

$$\begin{aligned}
& (\text{lr-proper-p-areasp}(\text{data-seg}) \\
& \wedge \quad \text{lr-proper-heapp}(\text{data-seg})
\end{aligned}$$

```

 $\wedge$  lr-s-similar-const-table (table, 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)
     $\not\leq$  (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))
     $\not\leq$  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*)
 $\wedge$ lr-proper-heapp (*data-seg*)
 $\wedge$ lr-s-similar-const-table (*table*, *data-seg*)
 $\wedge$ definedp (*f*, *table*)
 $\wedge$ definedp (*t*, *table*)
 $\wedge$ s-init-data-seg-restrictedp (*params*)
 $\wedge$ (*word-size* $\not\leq$ 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)
 $\not\leq$ 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* $\not\leq$ 4) $\rightarrow$ lr-minimum-heapp (lr-init-data-seg (*heap-size*))

THEOREM: adpp-cons-heap-name-node-size-lr-init-data-seg
 (*heap-size* $\not\leq$ 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)
 $\wedge$ (caddr (*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)
 $\wedge$ (caddr (*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)
 $\not\leq$ (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)
 $\not\leq$ (offset (fetch (LR-FP-ADDR, *data-seg*)) + LR-NODE-SIZE))

$$\begin{aligned}
& \wedge \text{lr-proper-p-areasp} (data-seg)) \\
\rightarrow & (\text{fetch} (\text{identity} (\text{LR-FP-ADDR}), \text{car} (\text{lr-compile-quote} (\mathbf{t}, \mathbf{t}, data-seg, table)))) \\
& = \text{fetch} (\text{add-addr} (\text{fetch} (\text{identity} (\text{LR-FP-ADDR}), data-seg), \\
& \quad \text{identity} (\text{LR-REF-COUNT-OFFSET})), \\
& \quad data-seg))
\end{aligned}$$

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
 $(\text{lr-boundary-offsetp} (offset))$
 $\wedge (offset \in \mathbf{N})$
 $\wedge (offset \neq \text{offset} (\text{LR-UNDEF-ADDR}))$
 $\wedge (offset \neq \text{offset} (\text{LR-F-ADDR}))$
 $\wedge (offset \neq \text{offset} (\text{LR-T-ADDR}))$
 $\wedge (offset \neq \text{offset} (\text{LR-0-ADDR}))$
 $\rightarrow (offset \not\prec \text{LR-MINIMUM-HEAP-SIZE})$

THEOREM: lessp-minimum-heap-size-not-0-f-t-must-be-undef-alt-1
 $((\text{offset} (addr) < \text{LR-MINIMUM-HEAP-SIZE})$
 $\wedge (\text{type} (addr) = \mathbf{'addr})$
 $\wedge (\text{caddr} (addr) = \mathbf{nil})$
 $\wedge \text{listp} (addr)$
 $\wedge (\text{area-name} (addr) = \mathbf{'heap})$
 $\wedge \text{adpp} (\text{untag} (addr), \text{lr-init-data-seg} (heap-size))$
 $\wedge \text{lr-boundary-nodep} (addr)$
 $\wedge (\text{offset} (addr) \neq \text{offset} (\text{LR-0-ADDR}))$
 $\wedge (\text{offset} (addr) \neq \text{offset} (\text{LR-T-ADDR}))$
 $\wedge (\text{offset} (addr) \neq \text{offset} (\text{LR-F-ADDR}))$
 $\rightarrow (\text{fetch} (addr, \text{lr-init-data-seg} (heap-size))$
 $\quad = \text{identity} (\text{tag} (\mathbf{'nat}, \text{LR-UNDEFINED-TAG})))$

THEOREM: lessp-lr-boundary-offsetp-3
 $(\text{lr-boundary-offsetp} (offset) \wedge (offset \in \mathbf{N}))$
 $\rightarrow ((offset < 3) = (offset = 0))$

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
 $((offset \in \mathbf{N}) \wedge (offset < (\text{LR-NODE-SIZE} + \text{offset} (\text{LR-F-ADDR}))))$
 $\rightarrow (\text{lr-boundary-offsetp} (offset))$

$$= ((offset = identity (offset (LR-F-ADDR))) \\ \vee (offset = identity (offset (LR-UNDEF-ADDR))))$$

THEOREM: fetch-ref-count-lr-init-data-seg-free-list
 $(((LR-NODE-SIZE * heap-size) \not< (offset(addr) + LR-NODE-SIZE))$
 $\wedge (type(addr) = 'addr)$
 $\wedge (caddr(addr) = nil)$
 $\wedge listp(addr)$
 $\wedge (area-name(addr) = LR-HEAP-NAME)$
 $\wedge listp(untag(addr))$
 $\wedge lr-boundary-nodep(addr)$
 $\wedge (heap-size \not< 2))$
 $\rightarrow (fetch(add-addr(addr, identity(LR-REF-COUNT-OFFSET)),$
 $lr-init-data-seg(heap-size))$
 $= \text{if } offset(LR-F-ADDR) < offset(addr)$
 $\text{then } add-addr(addr, identity(LR-NODE-SIZE))$
 $\text{else } identity(tag('nat, 1)) \text{ endif})$

THEOREM: lessp-offset-lr-init-data-seg-adpp-untag-lessp-offset
 $(adpp(untag(addr), lr-init-data-seg(heap-size))$
 $\wedge lr-boundary-offsetp(offset)$
 $\wedge lr-boundary-offsetp(offset(addr))$
 $\wedge (offset < offset(addr))$
 $\wedge (area-name(addr) = LR-HEAP-NAME)$
 $\wedge (heap-size \not< 2))$
 $\rightarrow ((offset < (identity(LR-NODE-SIZE) * heap-size)) = t)$

THEOREM: lr-count-free-nodes-lr-all-nodes
 $(lr-boundary-nodep(addr)$
 $\wedge (area-name(addr) = LR-HEAP-NAME)$
 $\wedge listp(untag(addr))$
 $\wedge (offset(addr) \in \mathbf{N})$
 $\wedge (type(addr) = 'addr)$
 $\wedge (caddr(addr) = nil)$
 $\wedge listp(addr)$
 $\wedge lr-boundary-nodep(max-addr)$
 $\wedge (area-name(max-addr) = LR-HEAP-NAME)$
 $\wedge adpp(untag(max-addr), lr-init-data-seg(heap-size))$
 $\wedge (type(max-addr) = 'addr)$
 $\wedge (caddr(max-addr) = nil)$
 $\wedge listp(max-addr)$
 $\wedge (offset(LR-F-ADDR) < offset(addr))$
 $\wedge (heap-size \not< 2))$
 $\rightarrow (lr-count-free-nodes(addr,$
 $lr-all-nodes(offset(addr), max-addr),$

$$\text{lr-init-data-seg}(\text{heap-size}))$$

$$= \text{length}(\text{lr-all-nodes}(\text{offset}(\text{addr}), \text{max-addr})))$$

THEOREM: same-signature-car-lr-init-data-seg-table-help-1

let *comp-obj* **be** $\text{lr-compile-quote}(\mathbf{t}, \text{object}, \text{data-seg}, \text{table})$
in
 $(\text{same-signature}(\text{car}(\text{comp-obj}),$
 $\quad \text{car}(\text{lr-init-data-seg-table}(\text{params},$
 $\quad \quad \text{car}(\text{comp-obj}),$
 $\quad \quad \text{cdr}(\text{comp-obj}))))$
 $\wedge \text{lr-proper-free-listp}(\text{data-seg})$
 $\wedge \text{lr-proper-p-areasp}(\text{data-seg})$
 $\wedge \text{definedp}(\text{LR-HEAP-NAME}, \text{data-seg})$
 $\wedge \text{lr-boundary-nodep}(\text{lr-max-node}(\text{data-seg}))$
 $\rightarrow \text{same-signature}(\text{data-seg},$
 $\quad \text{car}(\text{lr-init-data-seg-table}(\text{params},$
 $\quad \quad \text{car}(\text{comp-obj}),$
 $\quad \quad \text{cdr}(\text{comp-obj}))))$ **endlet**

THEOREM: same-signature-car-lr-init-data-seg-table

$(\text{lr-proper-free-listp}(\text{data-seg})$
 $\wedge \text{lr-proper-p-areasp}(\text{data-seg})$
 $\wedge \text{definedp}(\text{LR-HEAP-NAME}, \text{data-seg})$
 $\wedge \text{lr-boundary-nodep}(\text{lr-max-node}(\text{data-seg}))$
 $\rightarrow \text{same-signature}(\text{data-seg},$
 $\quad \text{car}(\text{lr-init-data-seg-table}(\text{params}, \text{data-seg}, \text{table})))$

THEOREM: lr-max-node-car-lr-init-data-seg-table

$(\text{lr-proper-free-listp}(\text{data-seg})$
 $\wedge \text{lr-proper-p-areasp}(\text{data-seg})$
 $\wedge \text{definedp}(\text{LR-HEAP-NAME}, \text{data-seg})$
 $\wedge \text{lr-boundary-nodep}(\text{lr-max-node}(\text{data-seg}))$
 $\rightarrow (\text{lr-max-node}(\text{car}(\text{lr-init-data-seg-table}(\text{params}, \text{data-seg}, \text{table})))$
 $\quad = \text{lr-max-node}(\text{data-seg}))$

THEOREM: s-heap-reqs-object-0

$((\text{flag} \neq \text{'list}) \wedge (\neg \text{definedp}(0, \text{table})))$
 $\rightarrow (\text{s-heap-reqs}(\text{flag}, 0, \text{data-seg}, \text{table}) = 1)$

THEOREM: lr-free-list-nodes-deposit-0

$((\text{type}(\text{max-addr}) = \text{'addr})$
 $\wedge (\text{caddr}(\text{max-addr}) = \text{nil})$
 $\wedge \text{listp}(\text{max-addr})$
 $\wedge \text{adpp}(\text{untag}(\text{max-addr}), \text{data-seg})$
 $\wedge \text{lr-boundary-nodep}(\text{max-addr})$

$$\begin{aligned}
& \wedge (\text{area-name } (max\text{-}addr) = \text{LR-HEAP-NAME}) \\
& \wedge (\neg \text{definedp } (0, table)) \\
& \wedge \text{lr-proper-free-listp } (data\text{-}seg) \\
& \wedge \text{adpp } (\text{untag } (\text{lr-max-node } (data\text{-}seg)), data\text{-}seg) \\
& \wedge \text{lr-boundary-nodep } (\text{lr-max-node } (data\text{-}seg)) \\
& \wedge \text{lr-proper-p-areasp } (data\text{-}seg) \\
& \wedge ((\text{length } (\text{cdr } (\text{assoc } (\text{LR-HEAP-NAME}, data\text{-}seg)))) - 1) \\
& \quad \not\leq (\text{offset } (\text{fetch } (\text{LR-FP-ADDR}, data\text{-}seg)) + \text{LR-NODE-SIZE})) \\
\rightarrow & (\text{lr-free-list-nodes } (max\text{-}addr, \\
& \quad \text{car } (\text{lr-compile-quote } (\mathbf{t}, 0, data\text{-}seg, table))) \\
& \quad = \text{lr-free-list-nodes } (max\text{-}addr, \\
& \quad \quad \text{deposit } (\text{identity } (\text{tag } ('nat, 1)), \\
& \quad \quad \quad \text{add-addr } (\text{fetch } (\text{identity } (\text{LR-FP-ADDR}), \\
& \quad \quad \quad \quad data\text{-}seg), \\
& \quad \quad \quad \quad \text{identity } (\text{LR-REF-COUNT-OFFSET})), \\
& \quad \quad \quad data\text{-}seg)))
\end{aligned}$$

THEOREM: lr-free-list-nodes-deposit-t

$$\begin{aligned}
& ((\text{type } (max\text{-}addr) = 'addr) \\
& \wedge (\text{caddr } (max\text{-}addr) = \mathbf{nil}) \\
& \wedge \text{listp } (max\text{-}addr) \\
& \wedge \text{adpp } (\text{untag } (max\text{-}addr), data\text{-}seg) \\
& \wedge \text{lr-boundary-nodep } (max\text{-}addr) \\
& \wedge (\text{area-name } (max\text{-}addr) = \text{LR-HEAP-NAME}) \\
& \wedge (\neg \text{definedp } (\mathbf{t}, table)) \\
& \wedge \text{lr-proper-free-listp } (data\text{-}seg) \\
& \wedge \text{adpp } (\text{untag } (\text{lr-max-node } (data\text{-}seg)), data\text{-}seg) \\
& \wedge \text{lr-boundary-nodep } (\text{lr-max-node } (data\text{-}seg)) \\
& \wedge \text{lr-proper-p-areasp } (data\text{-}seg) \\
& \wedge ((\text{length } (\text{cdr } (\text{assoc } (\text{LR-HEAP-NAME}, data\text{-}seg)))) - 1) \\
& \quad \not\leq (\text{offset } (\text{fetch } (\text{LR-FP-ADDR}, data\text{-}seg)) + \text{LR-NODE-SIZE})) \\
\rightarrow & (\text{lr-free-list-nodes } (max\text{-}addr, \\
& \quad \text{car } (\text{lr-compile-quote } (\mathbf{t}, \mathbf{t}, data\text{-}seg, table))) \\
& \quad = \text{lr-free-list-nodes } (max\text{-}addr, \\
& \quad \quad \text{deposit } (\text{identity } (\text{tag } ('nat, 1)), \\
& \quad \quad \quad \text{add-addr } (\text{fetch } (\text{identity } (\text{LR-FP-ADDR}), \\
& \quad \quad \quad \quad data\text{-}seg), \\
& \quad \quad \quad \quad \text{identity } (\text{LR-REF-COUNT-OFFSET})), \\
& \quad \quad \quad data\text{-}seg)))
\end{aligned}$$

THEOREM: fetch-lr-fp-addr-compile-quote-0

$$\begin{aligned}
& ((\neg \text{definedp } (0, table)) \\
& \wedge ((\text{length } (\text{cdr } (\text{assoc } (\text{LR-HEAP-NAME}, data\text{-}seg)))) - 1) \\
& \quad \not\leq (\text{offset } (\text{fetch } (\text{LR-FP-ADDR}, data\text{-}seg)) + \text{LR-NODE-SIZE}))
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{ definedp}(\text{area-name}(\text{LR-FP-ADDR}), \text{data-seg})) \\
\rightarrow & (\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \\
& \quad \text{car}(\text{lr-compile-quote}(\mathbf{t}, 0, \text{data-seg}, \text{table}))) \\
& = \text{fetch}(\text{add-addr}(\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \text{data-seg}), \\
& \quad \text{identity}(\text{LR-REF-COUNT-OFFSET})), \\
& \quad \text{data-seg}))
\end{aligned}$$

THEOREM: $\text{fetch-add-addr-ref-count-f-addr-lr-init-data-seg}$
 $(\text{heap-size} \not\leq 2)$
 $\rightarrow (\text{fetch}(\text{identity}(\text{add-addr}(\text{LR-F-ADDR}, \text{LR-REF-COUNT-OFFSET})),$
 $\quad \text{lr-init-data-seg}(\text{heap-size}))$
 $= \text{identity}(\text{tag}(\mathbf{'nat}, 1)))$

THEOREM: $\text{lr-good-pointer-tablep-f-lr-f-addr-lr-init-data-seg}$
 $(\text{heap-size} \not\leq 4)$
 $\rightarrow \text{lr-good-pointer-tablep}(\text{list}(\text{cons}(\mathbf{f}, \text{identity}(\text{LR-F-ADDR}))),$
 $\quad \text{lr-init-data-seg}(\text{heap-size}))$

THEOREM: $\text{lr-proper-heapp-nodep-lr-undef-addr-lr-init-data-seg}$
 $(\text{heap-size} \not\leq 2)$
 $\rightarrow \text{lr-proper-heapp-nodep}(\text{identity}(\text{LR-UNDEF-ADDR}),$
 $\quad \text{lr-init-data-seg}(\text{heap-size}))$

DEFINITION:
 $\text{induct-hint-2}(\text{offset})$
 $= \text{if } \text{offset} < \text{offset}(\text{add-addr}(\text{LR-F-ADDR}, \text{LR-NODE-SIZE})) \text{ then } \mathbf{t}$
 $\quad \text{else } \text{induct-hint-2}(\text{offset} - \text{LR-NODE-SIZE}) \text{ endif}$

THEOREM: $\text{lr-boundary-offsetp-difference-lr-node-size}$
 $\text{lr-boundary-offsetp}(\text{offset})$
 $\rightarrow \text{lr-boundary-offsetp}(\text{offset} - \text{identity}(\text{LR-NODE-SIZE}))$

THEOREM: $\text{fetch-lr-f-addr-lr-init-data-seg}$
 $(\text{heap-size} \not\leq 2)$
 $\rightarrow (\text{fetch}(\text{identity}(\text{LR-F-ADDR}), \text{lr-init-data-seg}(\text{heap-size}))$
 $\quad = \text{identity}(\text{tag}(\mathbf{'nat}, \text{LR-FALSE-TAG})))$

THEOREM: $\text{lr-proper-heapp-nodep-lr-init-data-seg-helper}$
 $((\text{offset} \not\leq \text{offset}(\text{LR-F-ADDR}))$
 $\wedge (\text{heap-size} \not\leq 2)$
 $\wedge (\text{offset} \in \mathbf{N})$
 $\wedge \text{lr-boundary-nodep}(\text{tag}(\mathbf{'addr}, \text{cons}(\text{LR-HEAP-NAME}, \text{offset})))$
 $\wedge ((\text{LR-NODE-SIZE} * \text{heap-size}) \not\leq (\text{offset} + \text{LR-NODE-SIZE})))$
 $\rightarrow \text{lr-proper-heapp-nodep}(\text{tag}(\mathbf{'addr}, \text{cons}(\text{identity}(\text{LR-HEAP-NAME}), \text{offset})),$
 $\quad \text{lr-init-data-seg}(\text{heap-size}))$

THEOREM: lr-proper-heapp2-lr-init-data-seg-helper
 $((heap-size \not\leq 2)$
 $\wedge (offset < \text{length}(\text{cdr}(\text{assoc}(\text{LR-HEAP-NAME},$
 $\text{lr-init-data-seg}(heap-size))))))$
 $\wedge (offset \in \mathbf{N})$
 $\wedge \text{lr-boundary-nodp}(\text{tag}('addr, \text{cons}(\text{LR-HEAP-NAME}, offset))))$
 $\rightarrow \text{lr-proper-heapp2}(\text{tag}('addr, \text{cons}(\text{LR-HEAP-NAME}, offset)),$
 $\text{lr-init-data-seg}(heap-size))$

THEOREM: lr-proper-heapp2-lr-init-data-seg
 $(heap-size \not\leq 2)$
 $\rightarrow \text{lr-proper-heapp2}(\text{tag}('addr,$
 $\text{cons}(\text{identity}(\text{LR-HEAP-NAME}),$
 $\text{identity}(\text{LR-NODE-SIZE}) * heap-size)),$
 $\text{lr-init-data-seg}(heap-size))$

THEOREM: fetch-add-addr-ref-count-lr-t-addr-lr-init-data-seg
 $(heap-size \not\leq 4)$
 $\rightarrow (\text{fetch}(\text{identity}(\text{add-addr}(\text{LR-T-ADDR}, \text{LR-REF-COUNT-OFFSET})),$
 $\text{lr-init-data-seg}(heap-size))$
 $= \text{identity}(\text{add-addr}(\text{LR-T-ADDR}, \text{LR-NODE-SIZE})))$

THEOREM: fetch-t-addr-compile-quote-list-init-data-seg
 $((heap-size \not\leq 4) \wedge (\neg \text{definedp}(\mathbf{t}, table)))$
 $\rightarrow (\text{fetch}(\text{identity}(\text{LR-T-ADDR}),$
 $\text{car}(\text{lr-compile-quote}('list,$
 $\text{list}(\mathbf{t}, 0),$
 $\text{lr-init-data-seg}(heap-size),$
 $table)))$
 $= \text{identity}(\text{tag}('nat, \text{LR-TRUE-TAG})))$

THEOREM: fetch-ref-count-t-addr-compile-quote-list-init-data-seg
 $((heap-size \not\leq 4) \wedge (\neg \text{definedp}(\mathbf{t}, table)))$
 $\rightarrow (\text{fetch}(\text{identity}(\text{add-addr}(\text{LR-T-ADDR}, \text{LR-REF-COUNT-OFFSET})),$
 $\text{car}(\text{lr-compile-quote}('list,$
 $\text{list}(\mathbf{t}, 0),$
 $\text{lr-init-data-seg}(heap-size),$
 $table)))$
 $= \text{identity}(\text{tag}('nat, 1)))$

THEOREM: fetch-add-addr-ref-count-lr-0-addr-lr-init-data-seg
 $(heap-size \not\leq 4)$
 $\rightarrow (\text{fetch}(\text{identity}(\text{add-addr}(\text{LR-0-ADDR}, \text{LR-REF-COUNT-OFFSET})),$
 $\text{lr-init-data-seg}(heap-size))$
 $= \text{identity}(\text{add-addr}(\text{LR-0-ADDR}, \text{LR-NODE-SIZE})))$

THEOREM: fetch-0-addr-compile-quote-list-init-data-seg
 $((heap-size \not\leq 4) \wedge (\neg \text{definedp}(\mathbf{t}, table)) \wedge (\neg \text{definedp}(0, table)))$
 $\rightarrow$ (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
 $((heap-size \not\leq 4) \wedge (\neg \text{definedp}(\mathbf{t}, table)) \wedge (\neg \text{definedp}(0, table)))$
 $\rightarrow$ (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
 $((heap-size \not\leq 4) \wedge (\neg \text{definedp}(\mathbf{t}, table)) \wedge (\neg \text{definedp}(0, table)))$
 $\rightarrow$ (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
 $(heap-size \not\leq 4)$
 $\rightarrow$ 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
 $((\neg \text{definedp}(0, table)) \wedge (\neg \text{definedp}(\mathbf{t}, table)) \wedge (heap-size \not\leq 4))$
 $\rightarrow$ (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
 $((heap-size \not\leq 4) \wedge (\neg \text{definedp}(\mathbf{t}, table)) \wedge (\neg \text{definedp}(0, table)))$

$$\begin{aligned}
&\rightarrow (\text{fetch}(\text{identity}(\text{LR-F-ADDR}), \\
&\quad \text{car}(\text{lr-compile-quote}('list, \\
&\quad \quad \text{list}(\mathbf{t}, 0), \\
&\quad \quad \text{lr-init-data-seg}(\text{heap-size}), \\
&\quad \quad \text{table}))) \\
&= \text{identity}(\text{tag}('nat, \text{LR-FALSE-TAG}))
\end{aligned}$$

THEOREM: fetch-ref-count-f-addr-compile-quote-list-init-data-seg
 $((\text{heap-size} \not\leq 4) \wedge (\neg \text{definedp}(\mathbf{t}, \text{table})) \wedge (\neg \text{definedp}(0, \text{table})))$
 $\rightarrow (\text{fetch}(\text{identity}(\text{add-addr}(\text{LR-F-ADDR}, \text{LR-REF-COUNT-OFFSET})),$
 $\quad \text{car}(\text{lr-compile-quote}('list,$
 $\quad \quad \text{list}(\mathbf{t}, 0),$
 $\quad \quad \text{lr-init-data-seg}(\text{heap-size}),$
 $\quad \quad \text{table})))$
 $= \text{identity}(\text{tag}('nat, 1)))$

THEOREM: lr-s-similar-const-table-compile-quote-t0-init-data-seg
 $(\text{heap-size} \not\leq 4)$
 $\rightarrow \text{lr-s-similar-const-table}(\text{cdr}(\text{lr-compile-quote}('list,$
 $\quad \text{list}(\mathbf{t}, 0),$
 $\quad \text{lr-init-data-seg}(\text{heap-size}),$
 $\quad \text{list}(\text{cons}(\mathbf{f},$
 $\quad \quad \text{identity}(\text{LR-F-ADDR}))))),$
 $\quad \text{car}(\text{lr-compile-quote}('list,$
 $\quad \quad \text{list}(\mathbf{t}, 0),$
 $\quad \quad \text{lr-init-data-seg}(\text{heap-size}),$
 $\quad \quad \text{list}(\text{cons}(\mathbf{f},$
 $\quad \quad \quad \text{identity}(\text{LR-F-ADDR}))))))$

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
 $((\neg \text{definedp}(0, \text{table})) \wedge (\neg \text{definedp}(\mathbf{t}, \text{table})) \wedge (\text{heap-size} \not\leq 4))$
 $\rightarrow (\text{fetch}(\text{identity}(\text{LR-FP-ADDR}),$
 $\quad \text{car}(\text{lr-compile-quote}('list,$
 $\quad \quad \text{list}(\mathbf{t}, 0),$
 $\quad \quad \text{lr-init-data-seg}(\text{heap-size}),$
 $\quad \quad \text{table})))$
 $= \text{identity}(\text{add-addr}(\text{LR-0-ADDR}, \text{LR-NODE-SIZE})))$

THEOREM: lr-free-list-nodes-lr-compile-quote-t0
 $((\neg \text{definedp}(\mathbf{t}, \text{table})) \wedge (\neg \text{definedp}(0, \text{table})) \wedge (\text{heap-size} \not\leq 4))$
 $\rightarrow (\text{lr-free-list-nodes}(\text{tag}('addr,$
 $\quad \text{cons}(\text{identity}(\text{LR-HEAP-NAME}),$

$$\begin{aligned}
& \text{identity (LR-NODE-SIZE) * heap-size)), \\
& \text{car (lr-compile-quote ('list,} \\
& \quad \text{list (t, 0),} \\
& \quad \text{lr-init-data-seg (heap-size),} \\
& \quad \text{table}))} \\
= & \text{lr-all-nodes (identity (LR-NODE-SIZE + offset (LR-0-ADDR)),} \\
& \text{tag ('addr,} \\
& \quad \text{cons (identity (LR-HEAP-NAME),} \\
& \quad \text{identity (LR-NODE-SIZE) * heap-size))})
\end{aligned}$$

THEOREM: listp-lr-all-nodes

$$\begin{aligned}
& \text{listp (lr-all-nodes (min-offset, max-addr))} \\
= & ((\text{offset (max-addr)} \neq 0) \wedge (\text{min-offset} < \text{offset (max-addr)}))
\end{aligned}$$

THEOREM: length-lr-all-nodes

$$\begin{aligned}
& (\text{lr-boundary-offsetp (offset)} \wedge (\text{offset} \in \mathbf{N}) \wedge \text{lr-boundary-nodep (addr)}) \\
\rightarrow & (\text{length (lr-all-nodes (offset, addr))} \\
= & ((\text{offset (addr)} - \text{offset}) \div \text{identity (LR-NODE-SIZE)}))
\end{aligned}$$

THEOREM: lr-count-free-nodes-append-lr-all-nodes-fact

$$\begin{aligned}
& (((\text{LR-NODE-SIZE} * \text{heap-size}) \not\leq \text{LR-MINIMUM-HEAP-SIZE}) \\
& \wedge (\neg \text{definedp (t, table)}) \\
& \wedge (\neg \text{definedp (0, table)})) \\
\rightarrow & (\text{lr-count-free-nodes (identity (add-addr (LR-0-ADDR, LR-NODE-SIZE)),} \\
& \quad \text{lr-all-nodes (identity (LR-MINIMUM-HEAP-SIZE),} \\
& \quad \text{tag ('addr,} \\
& \quad \text{cons (identity (LR-HEAP-NAME),} \\
& \quad \text{identity (LR-NODE-SIZE)} \\
& \quad \text{* heap-size))}), \\
& \quad \text{car (lr-compile-quote ('list,} \\
& \quad \text{list (t, 0),} \\
& \quad \text{lr-init-data-seg (heap-size),} \\
& \quad \text{table}))} \\
= & (\text{heap-size} - \text{LR-NODE-SIZE}))
\end{aligned}$$

THEOREM: proper-p-alistp-pair-formal-with-addresses

$$\begin{aligned}
& (\text{all-litatoms (strip-cars (params))} \\
& \wedge \text{s-init-data-seg-restrictedp (params)} \\
& \wedge (\text{heap-size} \not\leq \text{s-total-heap-reqs (s-progs (s), params, heap-size)}) \\
& \wedge (\text{word-size} \not\leq \text{s-total-ws-reqs (s-progs (s), params, heap-size)})) \\
\rightarrow & \text{proper-p-alistp (pair-formals-with-addresses (params,} \\
& \quad \text{cdr (lr-data-seg-table (s-progs (s),} \\
& \quad \text{params,} \\
& \quad \text{heap-size))),} \\
& \text{lr->p (s->lr1 (s,}
\end{aligned}$$

```

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)

```

$$\begin{aligned}
& \wedge \text{lr-boundary-nodep}(\text{lr-max-node}(\text{data-seg})) \\
& \wedge (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{data-seg}), \\
& \quad \quad \text{data-seg})) \\
& \quad \not\leq \text{s-heap-reqs}(\mathbf{t}, \text{object}, \text{data-seg}, \text{table})) \\
\rightarrow & ((\text{s-heap-reqs}(\mathbf{t}, \text{object}, \text{data-seg}, \text{table}) \\
& \quad + \text{lr-count-free-nodes}(\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \\
& \quad \quad \text{car}(\text{lr-compile-quote}(\mathbf{t}, \\
& \quad \quad \quad \text{object}, \\
& \quad \quad \quad \text{data-seg}, \\
& \quad \quad \quad \text{table}))), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{car}(\text{lr-compile-quote}(\mathbf{t}, \\
& \quad \quad \quad \text{object}, \\
& \quad \quad \quad \text{data-seg}, \\
& \quad \quad \quad \text{table}))), \\
& \quad \text{car}(\text{lr-compile-quote}(\mathbf{t}, \\
& \quad \quad \text{object}, \\
& \quad \quad \text{data-seg}, \\
& \quad \quad \text{table})))) \\
& = \text{lr-count-free-nodes}(\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{data-seg}), \\
& \quad \text{data-seg}))
\end{aligned}$$

THEOREM: $\text{lr-count-free-nodes-lr-data-seg-table-body-s-heap-reqs}$

$$\begin{aligned}
& (\text{lr-proper-free-listp}(\text{data-seg}) \\
& \wedge \text{lr-proper-p-areasp}(\text{data-seg}) \\
& \wedge \text{definedp}(\text{LR-HEAP-NAME}, \text{data-seg}) \\
& \wedge \text{lr-minimum-heapp}(\text{data-seg}) \\
& \wedge \text{lr-boundary-nodep}(\text{lr-max-node}(\text{data-seg})) \\
& \wedge (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{data-seg}), \\
& \quad \quad \text{data-seg})) \\
& \quad \not\leq \text{s-heap-reqs-body}(\text{flag}, \text{body}, \text{data-seg}, \text{table})) \\
\rightarrow & ((\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \\
& \quad \text{car}(\text{lr-data-seg-table-body}(\text{flag}, \\
& \quad \quad \text{body}, \\
& \quad \quad \text{data-seg}, \\
& \quad \quad \text{table}))), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{car}(\text{lr-data-seg-table-body}(\text{flag},
\end{aligned}$$

body,
data-seg,
table))),

```

car (lr-data-seg-table-body (flag,
                             body,
                             data-seg,
                             table)))
+ s-heap-reqs-body (flag, body, 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-data-seg-table-body-lr-good-pointer-tablep-help1

```

let data-seg-tab be lr-data-seg-table-body (t, car (body), data-seg, table)
in
(listp (body)
  ^ 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)
     $\not\leq$  (s-heap-reqs-body (t, car (body), data-seg, table)
      + s-heap-reqs-body ('list,
                          cdr (body),
                          car (data-seg-tab),
                          cdr (data-seg-tab))))))
 $\rightarrow$  (lr-count-free-nodes (fetch (identity (LR-FP-ADDR),
                                   car (data-seg-tab)),
                               lr-free-list-nodes (lr-max-node (data-seg),
                                                         car (data-seg-tab)),
                               car (data-seg-tab))
   $\not\leq$  s-heap-reqs-body ('list,
                      cdr (body),
                      car (data-seg-tab),
                      cdr (data-seg-tab))) endlet

```

THEOREM: lr-data-seg-table-body-lr-good-pointer-tablep

```

(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-good-pointerp-tablep (table, data-seg)
^  s-data-seg-body-restrictedp (flag, body)
^  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)
                        ✗ s-heap-reqs-body (flag, body, data-seg, table)))
→  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)
^  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)
                        ✗ (s-heap-reqs-body (t, s-body (car (progs)), data-seg, table)
                          + s-heap-reqs-list (cdr (progs),
                                                  car (dst-body),
                                                  cdr (dst-body))))))
→  (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)),
                        car (dst-body))

```

```

      ↯ s-heap-reqs-list (cdr (progs),
                          car (dst-body),
                          cdr (dst-body))) endlet

```

THEOREM: lr-init-data-seg-table-lr-good-pointer-tablep

```

(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-good-pointer-tablep (table, data-seg)
  ∧ s-init-data-seg-restrictedp (params)
  ∧ 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)
    ↯ s-init-heap-reqs (params, data-seg, table)))
→ lr-good-pointer-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)
  ∧ lr-proper-p-areasp (data-seg)
  ∧ lr-good-pointer-tablep (table, data-seg)
  ∧ s-data-seg-body-restrictedp (flag, body)
  ∧ 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)
    ↯ s-heap-reqs-body (flag, body, data-seg, table)))
→ 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)
  ∧ lr-proper-p-areasp (data-seg)
  ∧ lr-good-pointer-tablep (table, data-seg)
  ∧ s-data-seg-list-restrictedp (progs)
  ∧ definedp (f, table)
  ∧ (lr-count-free-nodes (fetch (LR-FP-ADDR, data-seg),

```

$$\begin{aligned}
& \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \text{data-seg}), \\
& \quad \text{data-seg}) \\
& \not\rightarrow \text{s-heap-reqs-list}(\text{progs}, \text{data-seg}, \text{table})) \\
\rightarrow & \text{lr-proper-heapp}(\text{car}(\text{lr-data-seg-table-list}(\text{progs}, \text{data-seg}, \text{table})))
\end{aligned}$$

THEOREM: $\text{lr-proper-heapp-car-lr-init-data-seg-table}$

$$\begin{aligned}
& (\text{lr-proper-heapp}(\text{data-seg}) \\
& \wedge \text{lr-proper-p-areasp}(\text{data-seg}) \\
& \wedge \text{lr-good-pointer-tablep}(\text{table}, \text{data-seg}) \\
& \wedge \text{s-init-data-seg-restrictedp}(\text{params}) \\
& \wedge \text{definedp}(\mathbf{f}, \text{table}) \\
& \wedge (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \text{data-seg}), \\
& \quad \text{data-seg}) \\
& \not\rightarrow \text{s-init-heap-reqs}(\text{params}, \text{data-seg}, \text{table}))) \\
\rightarrow & \text{lr-proper-heapp}(\text{car}(\text{lr-init-data-seg-table}(\text{params}, \text{data-seg}, \text{table})))
\end{aligned}$$

THEOREM: $\text{lr-count-free-nodes-lr-init-data-seg-table-s-init-heap-reqs}$

$$\begin{aligned}
& (\text{lr-proper-free-listp}(\text{data-seg}) \\
& \wedge \text{lr-proper-p-areasp}(\text{data-seg}) \\
& \wedge \text{definedp}(\text{LR-HEAP-NAME}, \text{data-seg}) \\
& \wedge \text{lr-minimum-heapp}(\text{data-seg}) \\
& \wedge \text{lr-boundary-nodep}(\text{lr-max-node}(\text{data-seg})) \\
& \wedge (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \text{data-seg}), \\
& \quad \text{data-seg}) \\
& \not\rightarrow \text{s-init-heap-reqs}(\text{params}, \text{data-seg}, \text{table}))) \\
\rightarrow & ((\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \\
& \quad \text{car}(\text{lr-init-data-seg-table}(\text{params}, \\
& \quad \text{data-seg}, \\
& \quad \text{table}))), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \text{car}(\text{lr-init-data-seg-table}(\text{params}, \\
& \quad \text{data-seg}, \\
& \quad \text{table}))), \\
& \quad \text{car}(\text{lr-init-data-seg-table}(\text{params}, \\
& \quad \text{data-seg}, \\
& \quad \text{table}))) \\
& + \text{s-init-heap-reqs}(\text{params}, \text{data-seg}, \text{table})) \\
= & \text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}),
\end{aligned}$$

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  $\nless$  (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))))))
   $\nless$  x) endlet

```

THEOREM: lr-proper-heapp-car-lr-data-seg-table

```

  ((heap-size  $\nless$  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
 $\text{lr-proper-p-areasp}(\text{car}(\text{lr-data-seg-table}(\text{progs}, \text{params}, \text{heap-size})))$

THEOREM: plist-strip-cdrs
 $\text{plist}(\text{strip-cdrs}(x)) = \text{strip-cdrs}(x)$

THEOREM: lr-make-temp-name-alist-1-plist-arg-1
 $\text{lr-make-temp-name-alist-1}(\text{initial}, \text{num-list}, \text{plist}(\text{temp-list}), \text{formals})$
 $= \text{lr-make-temp-name-alist-1}(\text{initial}, \text{num-list}, \text{temp-list}, \text{formals})$

THEOREM: lr-make-temp-name-alist-plist-arg-1
 $\text{lr-make-temp-name-alist}(\text{plist}(\text{temp-list}), \text{formals})$
 $= \text{lr-make-temp-name-alist}(\text{temp-list}, \text{formals})$

THEOREM: length-lr-make-initial-temps
 $\text{length}(\text{lr-make-initial-temps}(\text{temp-vars})) = \text{length}(\text{temp-vars})$

THEOREM: length-strip-cdrs
 $\text{length}(\text{strip-cdrs}(\text{alist})) = \text{length}(\text{alist})$

THEOREM: length-pair-formals-with-addresses
 $\text{length}(\text{pair-formals-with-addresses}(\text{formals}, \text{alist})) = \text{length}(\text{formals})$

THEOREM: s-good-statep-length-s-temp-list
 $\text{s-good-statep}(s, c)$
 $\rightarrow (\text{length}(\text{s-temp-list}(\text{s-prog}(s))) = \text{length}(\text{s-temps}(s)))$

THEOREM: plist-comp-programs-1
 $\text{plistp}(\text{comp-programs-1}(\text{progs}))$

THEOREM: proper-p-instructionp-ret
 $\text{proper-p-instructionp}('(\text{ret}), \text{name}, p)$

THEOREM: proper-p-instructionp-eq
 $\text{proper-p-instructionp}('(\text{eq}), \text{name}, p)$

THEOREM: proper-p-instructionp-fetch
 $\text{proper-p-instructionp}('(\text{fetch}), \text{name}, p)$

THEOREM: proper-p-instructionp-deposit
 $\text{proper-p-instructionp}('(\text{deposit}), \text{name}, p)$

THEOREM: proper-p-instructionp-add-addr
 $\text{proper-p-instructionp}('(\text{add-addr}), \text{name}, p)$

THEOREM: proper-p-instructionp-pop-global-free-ptr
 $\text{lr-proper-heapp}(\text{p-data-segment}(l))$
 $\rightarrow \text{proper-p-instructionp}('(\text{pop-global free-ptr}), \text{name}, \text{lr->p}(l))$

THEOREM: proper-p-instructionp-push-global-free-ptr
 lr-proper-heapp (p-data-segment (*l*))
 → proper-p-instructionp ('(push-global free-ptr), *name*, lr->p (*l*))

THEOREM: proper-p-instructionp-push-local-temp-car
 lr-programs-properp (*l*, *table*)
 → proper-p-instructionp ('(push-local x), 'car, lr->p (*l*))

THEOREM: proper-p-instructionp-push-local-temp-cdr
 lr-programs-properp (*l*, *table*)
 → proper-p-instructionp ('(push-local x), 'cdr, lr->p (*l*))

THEOREM: proper-p-instructionp-push-local-temp-cons
 lr-programs-properp (*l*, *table*)
 → proper-p-instructionp ('(push-local temp), 'cons, lr->p (*l*))

THEOREM: proper-p-instructionp-set-local-temp-cons
 lr-programs-properp (*l*, *table*)
 → proper-p-instructionp ('(set-local temp), 'cons, lr->p (*l*))

THEOREM: proper-labeled-p-instructionsp-append
 plistp (*instrs1*)
 → (proper-labeled-p-instructionsp (append (*instrs1*, *instrs2*), *name*, *p*)
 = (proper-labeled-p-instructionsp (*instrs1*, *name*, *p*)
 ∧ proper-labeled-p-instructionsp (*instrs2*, *name*, *p*)))

THEOREM: lessp-number-cons-cur-expr-dv-1-listp
 listp (cur-expr (*pos*, *body*))
 → (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*))
 → (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*))
 → (number-cons (cur-expr (dv (*pos*, 3), *body*))
 < number-cons (cur-expr (*pos*, *body*)))

THEOREM: lessp-number-cons-restn-cdr
 (listp (*pos*)
 ∧ (car (last (*pos*)) < length (cur-expr (butlast (*pos*), *body*)))
 ∧ listp (cur-expr (butlast (*pos*), *body*)))
 → (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 ($number$, $list$)) $\nless$ (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 ($n \bmod 10$),
 $list$)) **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 ÷ 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))
 → (lr-convert-num-to-ascii (number, cons (x, list1)) ≠ cons (y, list2))

THEOREM: lr-convert-num-to-ascii-equal-arg2-lengths
 ((length (list1) = length (list2)) ∧ (list1 ≠ list2))
 → (lr-convert-num-to-ascii (x, list1)
 ≠ lr-convert-num-to-ascii (y, list2))

THEOREM: lr-convert-num-to-ascii-equal-arg2-helper-1
 (x ≠ y)
 → (lr-convert-num-to-ascii (w, cons (x, list))
 ≠ 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 < n) ∧ (m < (n + length (instrs))))

THEOREM: find-labelp-lr-make-label-comp-body
 (n < length (comp-body (body)))
 → find-labelp (lr-make-label (n), comp-body (body))

THEOREM: label-instrs-append
 (n ∈ N)
 → (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-addr
 (all-litatoms (strip-cars (temp-var-dcls))
 ∧ all-undef-addr (strip-cadrs (temp-var-dcls))
 ∧ lr-proper-heapp (p-data-segment (p))
 ∧ lr-proper-p-areasp (p-data-segment (p)))
 → 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 ($\neg$ labelledp (car (*instrs*)))
 $\wedge$ not-labelledp-instrs (cdr (*instrs*))
 else t endif

THEOREM: label-instrs-proper-labeled-p-instructionsp
 (proper-labeled-p-instructionsp (*instrs*, *name*, *p*)
 $\wedge$ not-labelledp-instrs (*instrs*))
 $\rightarrow$ 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*) $\wedge$ 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
 $(\neg \text{listp}(\text{body})) \rightarrow (\text{comp-body-1}(\text{'list}, \text{body}, n) = \mathbf{nil})$

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))
 $\wedge$ proper-labeled-p-instructionsp (comp-body-1 (**t**, *then-body*, *n2*),
 name,
 p-state (*pc*,
 ctrl-stk,
 temp-stk,

$$\begin{aligned}
& \text{comp-programs} (prog\text{-}seg), \\
& \text{data-seg}, \\
& \text{max-ctrl}, \\
& \text{max-temp}, \\
& \text{word-size}, \\
& \text{psw})) \\
\wedge \quad & \text{proper-labeled-p-instructionsp} (else\text{-}instrs, \\
& \text{name}, \\
& \text{p-state} (pc, \\
& \text{ctrl-stk}, \\
& \text{temp-stk}, \\
& \text{comp-programs} (prog\text{-}seg), \\
& \text{data-seg}, \\
& \text{max-ctrl}, \\
& \text{max-temp}, \\
& \text{word-size}, \\
& \text{psw})) \\
\wedge \quad & \text{lr-proper-heapp} (data\text{-}seg) \\
\wedge \quad & \text{lr-proper-p-areasp} (data\text{-}seg) \\
\wedge \quad & \text{definedp} (name, prog\text{-}seg) \\
\wedge \quad & (((n \\
& \quad + \text{lr-p-c-size} (t, test\text{-}body) \\
& \quad + \text{lr-p-c-size} (t, then\text{-}body) \\
& \quad + \text{length} (else\text{-}instrs) \\
& \quad + 4) - 1) \\
& < \text{lr-p-c-size} (t, \text{program-body} (\text{assoc} (name, prog\text{-}seg)))))) \\
\rightarrow \quad & \text{proper-labeled-p-instructionsp} (\text{comp-if} (\text{comp-body-1} (t, test\text{-}body, n1), \\
& \text{comp-body-1} (t, then\text{-}body, n2), \\
& else\text{-}instrs, \\
& n), \\
& name, \\
& \text{p-state} (pc, \\
& \text{ctrl-stk}, \\
& \text{temp-stk}, \\
& \text{comp-programs} (prog\text{-}seg), \\
& \text{data-seg}, \\
& \text{max-ctrl}, \\
& \text{max-temp}, \\
& \text{word-size}, \\
& \text{psw}))
\end{aligned}$$

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-2

$$\begin{aligned}
& (\text{listp} (body)) \\
& \wedge \quad ((\text{car} (body) = \text{S-TEMP-FETCH}) \vee (\text{car} (body) = \text{S-TEMP-TEST}))
\end{aligned}$$

$$\begin{aligned}
& \wedge \text{lr-proper-exprp}(\mathbf{t}, \\
& \quad \text{body}, \\
& \quad \text{program-names}, \\
& \quad \text{formal-vars}(\text{assoc}(\text{name}, \text{prog-seg})), \\
& \quad \text{strip-cars}(\text{temp-var-dcls}(\text{assoc}(\text{name}, \text{prog-seg}))), \\
& \quad \text{table}) \\
& \wedge \text{definedp}(\text{name}, \text{prog-seg})) \\
\rightarrow & \text{proper-labeled-p-instructionsp}(\text{list}(\text{list}(\text{'push-local}, \text{caddr}(\text{body}))), \\
& \quad \text{name}, \\
& \quad \text{p-state}(\text{pc}, \\
& \quad \quad \text{ctrl-stk}, \\
& \quad \quad \text{temp-stk}, \\
& \quad \quad \text{comp-programs}(\text{prog-seg}), \\
& \quad \quad \text{data-seg}, \\
& \quad \quad \text{max-ctrl}, \\
& \quad \quad \text{max-temp}, \\
& \quad \quad \text{word-size}, \\
& \quad \quad \text{psw}))
\end{aligned}$$

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-3

$$\begin{aligned}
& (\text{listp}(\text{body})) \\
& \wedge ((\text{car}(\text{body}) = \text{S-TEMP-EVAL}) \vee (\text{car}(\text{body}) = \text{S-TEMP-TEST})) \\
& \wedge \text{lr-proper-exprp}(\mathbf{t}, \\
& \quad \text{body}, \\
& \quad \text{program-names}, \\
& \quad \text{formal-vars}(\text{assoc}(\text{name}, \text{prog-seg})), \\
& \quad \text{strip-cars}(\text{temp-var-dcls}(\text{assoc}(\text{name}, \text{prog-seg}))), \\
& \quad \text{table}) \\
& \wedge \text{definedp}(\text{name}, \text{prog-seg})) \\
\rightarrow & \text{proper-labeled-p-instructionsp}(\text{list}(\text{list}(\text{'set-local}, \text{caddr}(\text{body}))), \\
& \quad \text{name}, \\
& \quad \text{p-state}(\text{pc}, \\
& \quad \quad \text{ctrl-stk}, \\
& \quad \quad \text{temp-stk}, \\
& \quad \quad \text{comp-programs}(\text{prog-seg}), \\
& \quad \quad \text{data-seg}, \\
& \quad \quad \text{max-ctrl}, \\
& \quad \quad \text{max-temp}, \\
& \quad \quad \text{word-size}, \\
& \quad \quad \text{psw}))
\end{aligned}$$

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-4

$$\begin{aligned}
& (\text{listp}(\text{body})) \\
& \wedge (\text{car}(\text{body}) = \text{S-TEMP-TEST})
\end{aligned}$$

```

 $\wedge$  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))
 $\wedge$  lr-proper-exprp (t,
                    body,
                    program-names,
                    formal-vars (assoc (name, prog-seg)),
                    strip-cars (temp-var-dcls (assoc (name, prog-seg))),
                    table)
 $\wedge$  lr-proper-heapp (data-seg)
 $\wedge$  lr-proper-p-areasp (data-seg)
 $\wedge$  (lr-p-c-size (t, program-body (assoc (name, prog-seg)))
     $\not\leq$  (n + lr-p-c-size (t, cadr (body)) + 7))
 $\wedge$  definedp (name, prog-seg)
 $\rightarrow$  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
 $((addr \in \text{strip-cdrs}(table)) \wedge \text{lr-s-similar-const-table}(table, data-seg))$
 $\rightarrow \text{lr-valp}(\text{car}(\text{assoc-cdr}(addr, table)), addr, data-seg)$

THEOREM: lr-s-similar-const-table-type-addr-member-strip-cdrs
 $((addr \in \text{strip-cdrs}(table)) \wedge \text{lr-s-similar-const-table}(table, data-seg))$
 $\rightarrow ((\text{type}(addr) = \text{'addr})$
 $\quad \wedge \quad (\text{caddr}(addr) = \text{nil})$
 $\quad \wedge \quad \text{listp}(addr)$
 $\quad \wedge \quad \text{adpp}(\text{untag}(addr), data-seg)$
 $\quad \wedge \quad \text{lr-boundary-nodep}(addr)$
 $\quad \wedge \quad (\text{area-name}(addr) = \text{identity}(\text{LR-HEAP-NAME}))$
 $\quad \wedge \quad (\text{type}(\text{fetch}(\text{add-addr}(addr, \text{identity}(\text{LR-REF-COUNT-OFFSET})),$
 $\quad \quad \quad data-seg)))$
 $\quad = \quad \text{'nat}))$

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-5
 $(\text{listp}(body))$
 $\quad \wedge \quad (\text{car}(body) = \text{'quote})$
 $\quad \wedge \quad \text{lr-proper-heapp}(data-seg)$
 $\quad \wedge \quad \text{lr-proper-p-areasp}(data-seg)$
 $\quad \wedge \quad \text{lr-proper-exprp}(t,$
 $\quad \quad \quad body,$
 $\quad \quad \quad \text{strip-logic-fnames}(\text{cdr}(prog-seg)),$
 $\quad \quad \quad \text{formal-vars}(\text{assoc}(name, prog-seg)),$
 $\quad \quad \quad \text{strip-cars}(\text{temp-var-dcls}(\text{assoc}(name, prog-seg))),$
 $\quad \quad \quad table)$
 $\quad \wedge \quad \text{lr-s-similar-const-table}(table, data-seg)$
 $\quad \wedge \quad \text{definedp}(name, prog-seg))$
 $\rightarrow \text{proper-labeled-p-instructionsp}(\text{list}(\text{list}(\text{'push-constant},$
 $\quad \quad \quad \text{cadr}(body))),$
 $\quad \quad \quad name,$
 $\quad \quad \quad \text{p-state}(pc,$
 $\quad \quad \quad \quad \text{ctrl-stk},$
 $\quad \quad \quad \quad \text{temp-stk},$
 $\quad \quad \quad \text{comp-programs}(prog-seg),$
 $\quad \quad \quad \text{data-seg},$
 $\quad \quad \quad \text{max-ctrl},$
 $\quad \quad \quad \text{max-temp},$
 $\quad \quad \quad \text{word-size},$
 $\quad \quad \quad \text{psw}))$

EVENT: Disable lr-s-similar-const-table-type-addr-member-strip-cdrs.

THEOREM: lr-proper-exprp-flag-list-cdr-funcall

```

(listp (body)
  ^ (car (body) ≠ 'if)
  ^ (car (body) ≠ S-TEMP-FETCH)
  ^ (car (body) ≠ S-TEMP-EVAL)
  ^ (car (body) ≠ S-TEMP-TEST)
  ^ (car (body) ≠ 'quote)
  ^ lr-proper-exprp (t, body, program-names, formals, temps, table)
→ lr-proper-exprp ('list, cdr (body), program-names, formals, temps, table)

```

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-6-1

```

(listp (body)
  ^ (car (body) ≠ 'if)
  ^ (car (body) ≠ S-TEMP-FETCH)
  ^ (car (body) ≠ S-TEMP-EVAL)
  ^ (car (body) ≠ S-TEMP-TEST)
  ^ (car (body) ≠ 'quote)
  ^ lr-proper-exprp (t,
                    body,
                    strip-logic-fnames (cdr (prog-seg)),
                    formals,
                    temps,
                    table)
  ^ definedp (car (body), P-RUNTIME-SUPPORT-PROGRAMS))
→ proper-labeled-p-instructionsp (list (list ('call, 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-6-2

```

(listp (body)
  ^ (car (body) ≠ 'if)
  ^ (car (body) ≠ S-TEMP-FETCH)
  ^ (car (body) ≠ S-TEMP-EVAL)
  ^ (car (body) ≠ S-TEMP-TEST)
  ^ (car (body) ≠ 'quote)
  ^ lr-proper-exprp (t,
                    body,

```

$$\begin{aligned}
& \text{strip-logic-fnames}(\text{cdr}(\text{prog-seg})), \\
& \text{formals}, \\
& \text{temps}, \\
& \text{table}) \\
\wedge & (\neg \text{definedp}(\text{car}(\text{body}), \text{P-RUNTIME-SUPPORT-PROGRAMS})) \\
\wedge & \text{all-user-fnamesp}(\text{cdr}(\text{strip-cars}(\text{prog-seg}))) \\
\wedge & \text{definedp}(\text{name}, \text{prog-seg}) \\
\rightarrow & \text{proper-labeled-p-instructionsp}(\text{list}(\text{list}(\text{'call}, \\
& \text{user-fname}(\text{car}(\text{body}))), \\
& \text{name}, \\
& \text{p-state}(\text{pc}, \\
& \text{ctrl-stk}, \\
& \text{temp-stk}, \\
& \text{comp-programs}(\text{prog-seg}), \\
& \text{data-seg}, \\
& \text{max-ctrl}, \\
& \text{max-temp}, \\
& \text{word-size}, \\
& \text{psw}))
\end{aligned}$$

THEOREM: proper-labeled-p-instructionsp-comp-body-1-helper-7

$$\begin{aligned}
& ((\neg \text{listp}(\text{body})) \\
& \wedge \text{lr-proper-exprp}(\text{t}, \\
& \text{body}, \\
& \text{program-names}, \\
& \text{formal-vars}(\text{assoc}(\text{name}, \text{prog-seg})), \\
& \text{temps}, \\
& \text{table}) \\
& \wedge \text{definedp}(\text{name}, \text{prog-seg})) \\
\rightarrow & \text{proper-labeled-p-instructionsp}(\text{list}(\text{list}(\text{'push-local}, \text{body})), \\
& \text{name}, \\
& \text{p-state}(\text{pc}, \\
& \text{ctrl-stk}, \\
& \text{temp-stk}, \\
& \text{comp-programs}(\text{prog-seg}), \\
& \text{data-seg}, \\
& \text{max-ctrl}, \\
& \text{max-temp}, \\
& \text{word-size}, \\
& \text{psw}))
\end{aligned}$$

THEOREM: not-definedp-not-listp

$$(\neg \text{listp}(\text{alist})) \rightarrow (\neg \text{definedp}(\text{name}, \text{alist}))$$

THEOREM: proper-labeled-p-instructionsp-comp-body-1

```

(lr-proper-heapp (data-seg)
  ∧ lr-proper-p-areasp (data-seg)
  ∧ all-user-fnamesp (cdr (strip-cars (prog-seg)))
  ∧ 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)
  ∧ lr-s-similar-const-table (table, data-seg)
  ∧ ((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))))))
  ∧ definedp (name, prog-seg))
→ 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-addr-strip-cadrs-lr-make-temp-var-dcls
all-undef-addr-strip-cadrs (lr-make-temp-var-dcls (*temp-alist*)))

THEOREM: proper-p-program-p-car-code
(lr-programs-properp (*l*, *table*)
 ∧ (p-word-size (*l*) ∈ **N**)
 ∧ (p-word-size (*l*) < S-MAX-SUBR-REQS)
 ∧ lr-proper-heapp (p-data-segment (*l*))
 ∧ lr-proper-p-areasp (p-data-segment (*l*)))
→ proper-p-programp (P-CAR-CODE, lr->p (*l*))

THEOREM: proper-p-program-p-cdr-code
(lr-programs-properp (*l*, *table*)
 ∧ (p-word-size (*l*) ∈ **N**)
 ∧ (p-word-size (*l*) < S-MAX-SUBR-REQS)
 ∧ lr-proper-heapp (p-data-segment (*l*))
 ∧ lr-proper-p-areasp (p-data-segment (*l*)))

→ proper-p-programp (P-CDR-CODE, lr->p (l))

THEOREM: proper-p-programp-p-cons-code
 (lr-programs-properp (l, table)
 ∧ (p-word-size (l) ∈ **N**)
 ∧ (p-word-size (l) < S-MAX-SUBR-REQS)
 ∧ lr-proper-heapp (p-data-segment (l))
 ∧ lr-proper-p-areasp (p-data-segment (l)))
 → proper-p-programp (P-CONS-CODE, lr->p (l))

THEOREM: proper-p-programp-p-false-code
 (lr-programs-properp (l, table)
 ∧ (p-word-size (l) ∈ **N**)
 ∧ (p-word-size (l) < log (2, LR-FALSE-TAG))
 ∧ lr-proper-heapp (p-data-segment (l))
 ∧ lr-proper-p-areasp (p-data-segment (l)))
 → proper-p-programp (P-FALSE-CODE, lr->p (l))

THEOREM: proper-p-programp-p-falsep-code
 (lr-programs-properp (l, table)
 ∧ (p-word-size (l) ∈ **N**)
 ∧ (p-word-size (l) < log (2, LR-FALSE-TAG))
 ∧ lr-proper-heapp (p-data-segment (l))
 ∧ lr-proper-p-areasp (p-data-segment (l)))
 → proper-p-programp (P-FALSEP-CODE, lr->p (l))

THEOREM: proper-p-programp-p-listp-code
 (lr-programs-properp (l, table)
 ∧ (p-word-size (l) ∈ **N**)
 ∧ (p-word-size (l) < S-MAX-SUBR-REQS)
 ∧ lr-proper-heapp (p-data-segment (l))
 ∧ lr-proper-p-areasp (p-data-segment (l)))
 → proper-p-programp (P-LISTP-CODE, lr->p (l))

THEOREM: proper-p-programp-p-nlistp-code
 (lr-programs-properp (l, table)
 ∧ (p-word-size (l) ∈ **N**)
 ∧ (p-word-size (l) < S-MAX-SUBR-REQS)
 ∧ lr-proper-heapp (p-data-segment (l))
 ∧ lr-proper-p-areasp (p-data-segment (l)))
 → proper-p-programp (P-NLISTP-CODE, lr->p (l))

THEOREM: proper-p-programp-p-true-code
 (lr-programs-properp (l, table)
 ∧ (p-word-size (l) ∈ **N**)

$$\begin{aligned} & \wedge \text{ (p-word-size } (l) \not\leq \log(2, \text{LR-TRUE-TAG})) \\ & \wedge \text{ lr-proper-heapp (p-data-segment } (l)) \\ & \wedge \text{ lr-proper-p-areasp (p-data-segment } (l)) \\ \rightarrow & \text{ proper-p-programp (P-TRUE-CODE, lr->p } (l)) \end{aligned}$$
$$\begin{aligned} & \text{THEOREM: proper-p-program}(\text{p-truep-code} \\ & \quad (\text{lr-programs-properp}(l, \text{table}) \\ & \quad \wedge (\text{p-word-size}(l) \in \mathbf{N}) \\ & \quad \wedge (\text{p-word-size}(l) \not\leq \log(2, \text{LR-TRUE-TAG})) \\ & \quad \wedge \text{lr-proper-heapp}(\text{p-data-segment}(l)) \\ & \quad \wedge \text{lr-proper-p-areasp}(\text{p-data-segment}(l))) \\ & \rightarrow \text{proper-p-program}(\text{p-TRUEP-CODE}, \text{lr-}>\text{p}(l)) \end{aligned}$$
$$\begin{aligned}
& \text{THEOREM: } \text{lr-s-similar-const-table-lr-data-seg-table-body} \\
& (\text{lr-proper-p-areasp } (data-seg) \\
& \quad \wedge \text{ lr-proper-heapp } (data-seg) \\
& \quad \wedge \text{ lr-s-similar-const-table } (table, data-seg) \\
& \quad \wedge \text{ s-data-seg-body-restrictedp } (flag, body) \\
& \quad \wedge \text{ definedp } (f, table) \\
& \quad \wedge (\text{lr-count-free-nodes } (\text{fetch } (\text{LR-FP-ADDR}, data-seg), \\
& \quad \quad \quad \text{lr-free-list-nodes } (\text{lr-max-node } (data-seg), \\
& \quad \quad \quad \quad \quad \quad data-seg), \\
& \quad \quad \quad \quad \quad \quad data-seg) \\
& \quad \quad \not\wedge \text{ s-heap-reqs-body } (flag, body, data-seg, table))) \\
& \rightarrow \text{lr-s-similar-const-table } (\text{cdr } (\text{lr-data-seg-table-body } (flag, \\
& \quad \quad \quad \quad \quad \quad body, \\
& \quad \quad \quad \quad \quad \quad data-seg, \\
& \quad \quad \quad \quad \quad \quad table)), \\
& \quad \quad \quad \text{car } (\text{lr-data-seg-table-body } (flag, \\
& \quad \quad \quad \quad \quad \quad body, \\
& \quad \quad \quad \quad \quad \quad data-seg, \\
& \quad \quad \quad \quad \quad \quad table)))
\end{aligned}$$
$$\begin{aligned} & \text{THEOREM: lr-s-similar-const-table-lr-data-seg-table-list} \\ & (\text{lr-proper-p-areasp } (data-seg) \\ & \wedge \text{ lr-proper-heapp } (data-seg) \\ & \wedge \text{ lr-s-similar-const-table } (table, data-seg) \\ & \wedge \text{ s-data-seg-list-restrictedp } (progs) \\ & \wedge \text{ definedp } (f, table) \\ & \wedge (\text{lr-count-free-nodes } (\text{fetch } (LR-FP-ADDR, data-seg), \\ & \text{lr-free-list-nodes } (\text{lr-max-node } (data-seg), \\ & \text{data-seg}), \\ & \text{data-seg}) \\ & \not\wedge \text{ s-heap-reqs-list } (progs, data-seg, table))) \\ \rightarrow & \text{ lr-s-similar-const-table } (\text{cdr } (\text{lr-data-seg-table-list } (progs, \end{aligned}$$

data-seg,
table)),
car (lr-data-seg-table-list (*progs,*
data-seg,
table)))

THEOREM: lr-s-similar-const-table-lr-init-data-seg-table
(lr-proper-p-areasp (*data-seg*)
 $\wedge$ lr-proper-heapp (*data-seg*)
 $\wedge$ lr-s-similar-const-table (*table, data-seg*)
 $\wedge$ s-init-data-seg-restrictedp (*params*)
 $\wedge$ 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)
 $\not\prec$ s-init-heap-reqs (*params, data-seg, table*)))
 $\rightarrow$ lr-s-similar-const-table (cdr (lr-init-data-seg-table (*params,*
data-seg,
table)),
car (lr-init-data-seg-table (*params,*
data-seg,
table)))

THEOREM: lr-s-similar-const-table-cdr-car-lr-data-seg-table
((*heap-size* $\not\prec$ s-total-heap-reqs (*progs, params, heap-size*))
 $\wedge$ s-restrictedp (*progs, params*)
 $\rightarrow$ lr-s-similar-const-table (cdr (lr-data-seg-table (*progs, params, heap-size*)),
car (lr-data-seg-table (*progs, params, heap-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*)
 $\wedge$ 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*)
 $\wedge$ induct-hint-3 (**t**, dv (*pos*, 2), *prog*)
 $\wedge$ 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-pospl-cons-lessp-4-if-s-proper-exprp

```

((car (cur-expr (pos, body))) = 'if)
∧ good-pospl (pos, body)
∧ s-proper-exprp (t, body, program-names, formals, temp-list))
→ (good-pospl (dv (pos, 1), body)
    ∧ good-pospl (dv (pos, 2), body)
    ∧ good-pospl (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,

```


$$\begin{aligned}
& \text{test,} \\
& \text{program-names,} \\
& \text{formals,} \\
& \text{temp-alist,} \\
& \text{table}) \\
\wedge \quad & \text{lr-proper-exprp (t,} \\
& \text{then,} \\
& \text{program-names,} \\
& \text{formals,} \\
& \text{temp-alist,} \\
& \text{table})} \\
\wedge \quad & \text{lr-proper-exprp (t,} \\
& \text{else,} \\
& \text{program-names,} \\
& \text{formals,} \\
& \text{temp-alist,} \\
& \text{table}))
\end{aligned}$$

THEOREM: lr-proper-exprp-flag-not-list-cons-if

$$\begin{aligned}
& ((\text{flag} \neq \text{'list}) \\
& \wedge \quad \text{listp (cur-expr (pos, body))} \\
& \wedge \quad (\text{car (cur-expr (pos, body))} = \text{'if}) \\
& \wedge \quad \text{good-posp1 (pos, body)} \\
& \wedge \quad \text{lr-proper-exprp (t,} \\
& \qquad \qquad \text{lr-compile-body (t,} \\
& \qquad \qquad \qquad \text{cur-expr (dv (pos, 3), body),} \\
& \qquad \qquad \qquad \text{temp-alist,} \\
& \qquad \qquad \qquad \text{table),} \\
& \qquad \qquad \text{program-names,} \\
& \qquad \qquad \text{formals,} \\
& \qquad \qquad \text{strip-cdrs (temp-alist),} \\
& \qquad \qquad \text{table)} \\
& \wedge \quad \text{lr-proper-exprp (t,} \\
& \qquad \qquad \text{lr-compile-body (t,} \\
& \qquad \qquad \qquad \text{cur-expr (dv (pos, 2), body),} \\
& \qquad \qquad \qquad \text{temp-alist,} \\
& \qquad \qquad \qquad \text{table),} \\
& \qquad \qquad \text{program-names,} \\
& \qquad \qquad \text{formals,} \\
& \qquad \qquad \text{strip-cdrs (temp-alist),} \\
& \qquad \qquad \text{table)} \\
& \wedge \quad \text{lr-proper-exprp (t,} \\
& \qquad \qquad \text{lr-compile-body (t,} \\
& \qquad \qquad \qquad \text{cur-expr (dv (pos, 1), body),}
\end{aligned}$$

$$\begin{aligned}
& \text{temp-alist,} \\
& \text{table),} \\
& \text{program-names,} \\
& \text{formals,} \\
& \text{strip-cdrs (temp-alist),} \\
& \text{table)} \\
\wedge & \text{ s-proper-exprp (t, body, program-names, formals, strip-cars (temp-alist))} \\
\rightarrow & \text{ lr-proper-exprp (flag,} \\
& \text{cons ('if,} \\
& \text{lr-compile-body ('list,} \\
& \text{cdr (cur-expr (pos, body)),} \\
& \text{temp-alist,} \\
& \text{table)),} \\
& \text{program-names,} \\
& \text{formals,} \\
& \text{strip-cdrs (temp-alist),} \\
& \text{table)}
\end{aligned}$$

THEOREM: lr-proper-exprp-flag-not-list-cons-temp-fetch

$$\begin{aligned}
& ((\text{flag} \neq \text{'list}) \\
& \wedge \text{ good-posp1 (pos, body)} \\
& \wedge \text{ listp (cur-expr (pos, body))} \\
& \wedge (\text{car (cur-expr (pos, body))} = \text{S-TEMP-FETCH}) \\
& \wedge \text{ s-proper-exprp (t, body, program-names, formals, strip-cars (temp-alist))} \\
\rightarrow & \text{ lr-proper-exprp (flag,} \\
& \text{list (identity (S-TEMP-FETCH),} \\
& \text{x,} \\
& \text{cdr (assoc (cadr (cur-expr (pos, body)), temp-alist))),} \\
& \text{program-names,} \\
& \text{formals,} \\
& \text{strip-cdrs (temp-alist),} \\
& \text{table)}
\end{aligned}$$

THEOREM: good-posp1-dv-1-temp-eval-test

$$\begin{aligned}
& (\text{good-posp1 (pos, body)} \\
& \wedge ((\text{car (cur-expr (pos, body))} = \text{S-TEMP-EVAL}) \\
& \quad \vee (\text{car (cur-expr (pos, body))} = \text{S-TEMP-TEST})) \\
& \wedge \text{ listp (cur-expr (pos, body))} \\
\rightarrow & \text{ good-posp1 (dv (pos, 1), body)}
\end{aligned}$$

THEOREM: length-last-fact

$$(\text{length (x)} = 1) \rightarrow (\text{last (x)} = x)$$

THEOREM: lr-proper-exprp-flag-not-list-cons-temp-eval

$$((\text{flag} \neq \text{'list})$$

$$\begin{aligned}
& \wedge \text{listp}(\text{cur-expr}(\text{pos}, \text{body})) \\
& \wedge \text{good-posp1}(\text{pos}, \text{body}) \\
& \wedge (\text{car}(\text{cur-expr}(\text{pos}, \text{body})) = \text{S-TEMP-EVAL}) \\
& \wedge \text{lr-proper-exprp}(\mathbf{t}, \\
& \quad \text{lr-compile-body}(\mathbf{t}, \\
& \quad \quad \text{cur-expr}(\text{dv}(\text{pos}, 1), \text{body}), \\
& \quad \quad \text{temp-alist}, \\
& \quad \quad \text{table}), \\
& \quad \text{program-names}, \\
& \quad \text{formals}, \\
& \quad \text{strip-cdrs}(\text{temp-alist}), \\
& \quad \text{table}) \\
& \wedge \text{s-proper-exprp}(\mathbf{t}, \text{body}, \text{program-names}, \text{formals}, \text{strip-cars}(\text{temp-alist})) \\
\rightarrow & \text{lr-proper-exprp}(\text{flag}, \\
& \quad \text{list}(\text{identity}(\text{S-TEMP-EVAL}), \\
& \quad \quad \text{lr-compile-body}(\mathbf{t}, \\
& \quad \quad \quad \text{cadr}(\text{cur-expr}(\text{pos}, \text{body})), \\
& \quad \quad \quad \text{temp-alist}, \\
& \quad \quad \quad \text{table}), \\
& \quad \quad \text{cdr}(\text{assoc}(\text{cadr}(\text{cur-expr}(\text{pos}, \text{body})), \text{temp-alist}))), \\
& \quad \text{program-names}, \\
& \quad \text{formals}, \\
& \quad \text{strip-cdrs}(\text{temp-alist}), \\
& \quad \text{table})
\end{aligned}$$

THEOREM: lr-proper-exprp-flag-not-list-cons-temp-test

$$\begin{aligned}
& ((\text{flag} \neq \text{'list}) \\
& \wedge \text{listp}(\text{cur-expr}(\text{pos}, \text{body})) \\
& \wedge \text{good-posp1}(\text{pos}, \text{body}) \\
& \wedge (\text{car}(\text{cur-expr}(\text{pos}, \text{body})) = \text{S-TEMP-TEST}) \\
& \wedge \text{lr-proper-exprp}(\mathbf{t}, \\
& \quad \text{lr-compile-body}(\mathbf{t}, \\
& \quad \quad \text{cur-expr}(\text{dv}(\text{pos}, 1), \text{body}), \\
& \quad \quad \text{temp-alist}, \\
& \quad \quad \text{table}), \\
& \quad \text{program-names}, \\
& \quad \text{formals}, \\
& \quad \text{strip-cdrs}(\text{temp-alist}), \\
& \quad \text{table}) \\
& \wedge \text{s-proper-exprp}(\mathbf{t}, \text{body}, \text{program-names}, \text{formals}, \text{strip-cars}(\text{temp-alist})) \\
\rightarrow & \text{lr-proper-exprp}(\text{flag}, \\
& \quad \text{list}(\text{identity}(\text{S-TEMP-TEST}), \\
& \quad \quad \text{lr-compile-body}(\mathbf{t}, \\
& \quad \quad \quad \text{cadr}(\text{cur-expr}(\text{pos}, \text{body})),
\end{aligned}$$

```

                                temp-alist,
                                table),
  cdr (assoc (cadr (cur-expr (pos, body)), temp-alist))),
program-names,
formals,
strip-cdrs (temp-alist),
table)

```

DEFINITION:

[illegible]

DEFINITION:

[illegible]

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*)
→ 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*)))
∧ (*m* < *n*))
→ 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*))
→ (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))))))
∧ (*n* ∈ **N**)
∧ (*n* < length (*body*)))
→ 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*))
∧ (car (cur-expr (*pos*, *body*)) = 'quote)
∧ good-posp1 (*pos*, *body*))
→ 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
 (listp (cur-expr (*pos*, s-body (*prog*))))
 ∧ (car (cur-expr (*pos*, s-body (*prog*)))) = 'quote)
 ∧ (*prog* ∈ *progs*)
 ∧ good-posp1 (*pos*, s-body (*prog*)))
 → definedp (cadr (cur-expr (*pos*, s-body (*prog*))),
 cdr (lr-data-seg-table-list (*progs*, *data-seg*, *table*)))

THEOREM: definedp-cadr-cur-expr-quote-lr-data-seg-table
 (listp (cur-expr (*pos*, s-body (*prog*))))
 ∧ (car (cur-expr (*pos*, s-body (*prog*)))) = 'quote)
 ∧ (*prog* ∈ *progs*)
 ∧ good-posp1 (*pos*, s-body (*prog*)))
 → definedp (cadr (cur-expr (*pos*, s-body (*prog*))),
 cdr (lr-data-seg-table (*progs*, *params*, *heap-size*)))

THEOREM: lr-proper-exrp-p-lr-compile-programs-helper-2
 (listp (cur-expr (*pos*, s-body (*prog*))))
 ∧ (car (cur-expr (*pos*, s-body (*prog*)))) = 'quote)
 ∧ (*prog* ∈ *progs*)
 ∧ good-posp1 (*pos*, s-body (*prog*)))
 ∧ (*heap-size* < s-total-heap-reqs (*progs*, *params*, *heap-size*))
 ∧ s-restrictedp (*progs*, *params*)
 → (type (cdr (assoc (cadr (cur-expr (*pos*, s-body (*prog*))),
 cdr (lr-data-seg-table (*progs*, *params*, *heap-size*))))))
 = 'addr)

THEOREM: good-posp-dv-1-funcall-opened
 (listp (cur-expr (*pos*, s-body (*prog*))))
 ∧ (car (cur-expr (*pos*, s-body (*prog*)))) ≠ 'if)
 ∧ (car (cur-expr (*pos*, s-body (*prog*)))) ≠ S-TEMP-EVAL)
 ∧ (car (cur-expr (*pos*, s-body (*prog*)))) ≠ S-TEMP-TEST)
 ∧ (car (cur-expr (*pos*, s-body (*prog*)))) ≠ S-TEMP-FETCH)
 ∧ (car (cur-expr (*pos*, s-body (*prog*)))) ≠ 'quote)
 ∧ good-posp1 (*pos*, s-body (*prog*)))
 → good-posp ('list, dv (*pos*, 1), s-body (*prog*)))

THEOREM: s-restrict-subrps-list-lr-proper-get-t
 (s-restrict-subrps ('list, cdr (*expr*)))
 ∧ listp (*expr*)
 ∧ (*n* ≠ 0)
 ∧ (*n* < length (*expr*)))
 → s-restrict-subrps (*t*, get (*n*, *expr*)))

THEOREM: s-restrict-subrps-t-lr-proper-get-t

```

((car (body) ≠ S-TEMP-FETCH)
 ∧ (car (body) ≠ S-TEMP-EVAL)
 ∧ (car (body) ≠ S-TEMP-TEST)
 ∧ (car (body) ≠ 'quote)
 ∧ listp (body)
 ∧ s-proper-exprp (t, body, program-names, formals, temp-list)
 ∧ (n ≠ 0)
 ∧ (n < length (body))
 ∧ s-restrict-subrps (t, body))
→ 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)
 ∧ s-proper-exprp (t, body, program-names, formals, temp-list)
 ∧ good-posp1 (pos, body))
→ s-restrict-subrps (t, cur-expr (pos, body))

```

THEOREM: lr-proper-exprp-flag-not-list-cons-funcall

```

((flag ≠ 'list)
 ∧ listp (cur-expr (pos, s-body (prog)))
 ∧ good-posp1 (pos, s-body (prog))
 ∧ (car (cur-expr (pos, s-body (prog))) ≠ 'if)
 ∧ (car (cur-expr (pos, s-body (prog))) ≠ S-TEMP-FETCH)
 ∧ (car (cur-expr (pos, s-body (prog))) ≠ S-TEMP-EVAL)
 ∧ (car (cur-expr (pos, s-body (prog))) ≠ S-TEMP-TEST)
 ∧ (car (cur-expr (pos, s-body (prog))) ≠ 'quote)
 ∧ lr-proper-exprp ('list,
                    lr-compile-body ('list,
                                      cdr (cur-expr (pos, s-body (prog))),
                                      temp-alist,
                                      table),
                    program-names,
                    formals,
                    temps,
                    table)
 ∧ s-restrict-subrps (t, s-body (prog))
 ∧ s-proper-exprp (t, s-body (prog), program-names, formals, temp-list))
→ 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-names,
                      formals,
                      temps,
                      table))

```

table)),
program-names,
formals,
temps,
table)

THEOREM: lr-proper-exrp-flag-not-list-not-listp

((*flag* ≠ 'list)
 ∧ (¬ listp (cur-expr (*pos*, *body*)))
 ∧ good-posp1 (*pos*, *body*)
 ∧ s-proper-exrp (*t*, *body*, *program-names*, *formals*, *temp-list*))
 → lr-proper-exrp (*flag*,
 cur-expr (*pos*, *body*),
 program-names,
 formals,
 temps,
 table)

THEOREM: lr-proper-exrp-p-lr-compile-programs

(s-restrict-subrps (*t*, s-body (*prog*))
 ∧ (*prog* ∈ *progs*)
 ∧ subsetp (*progs*, *all-progs*)
 ∧ s-proper-exrp (*t*,
 s-body (*prog*),
 program-names,
 formals,
 strip-cars (*temp-alist*))
 ∧ good-posp (*flag*, *pos*, s-body (*prog*))
 ∧ (*heap-size* < s-total-heap-reqs (*all-progs*, *params*, *heap-size*))
 ∧ s-restrictedp (*all-progs*, *params*))
 → lr-proper-exrp (*flag*,
 lr-compile-body (*flag*,
 if *flag* = 'list
 then restn (car (last (*pos*)),
 cur-expr (butlast (*pos*),
 s-body (*prog*)))
 else cur-expr (*pos*, s-body (*prog*)) endif,
 temp-alist,
 cdr (lr-data-seg-table (*all-progs*,
 params,
 heap-size))),
 program-names,
 formals,
 strip-cdrs (*temp-alist*),
 cdr (lr-data-seg-table (*all-progs*, *params*, *heap-size*)))

EVENT: Disable lr-proper-exrp-p-lr-compile-programs-helper-1.

THEOREM: all-litatoms-s-formals-member-s-programs-properp
 $((prog \in progs) \wedge \text{s-programs-properp}(progs, program-names))$
 $\rightarrow \text{all-litatoms}(\text{s-formals}(prog))$

THEOREM: s-restrict-subrps-s-body-member-s-restrict-subrps-progs
 $(\text{s-restrict-subrps-progs}(progs) \wedge (prog \in progs))$
 $\rightarrow \text{s-restrict-subrps}(t, \text{s-body}(prog))$

THEOREM: lr-proper-exrp-p-lr-compile-programs-flag-t
 $(\text{s-restrict-subrps-progs}(all-progs)$
 $\wedge (prog \in all-progs)$
 $\wedge \text{s-programs-properp}(all-progs, program-names)$
 $\wedge (heap-size \not\leq \text{s-total-heap-reqs}(all-progs, params, heap-size))$
 $\wedge \text{s-restrictedp}(all-progs, params)$
 $\wedge (temp-alist = \text{lr-make-temp-name-alist}(\text{s-temp-list}(prog),$
 $\text{s-formals}(prog))))$
 $\rightarrow \text{lr-proper-exrp}(t,$
 $\text{lr-compile-body}(t,$
 $\text{s-body}(prog),$
 $temp-alist,$
 $\text{cdr}(\text{lr-data-seg-table}(all-progs,$
 $params,$
 $heap-size))),$
 $program-names,$
 $\text{s-formals}(prog),$
 $\text{strip-cdrs}(temp-alist),$
 $\text{cdr}(\text{lr-data-seg-table}(all-progs, params, heap-size)))$

THEOREM: lr-programs-properp-1-lr-compile-programs
 $(\text{s-programs-properp}(all-progs, program-names)$
 $\wedge \text{s-programs-okp}(progs)$
 $\wedge \text{s-restrict-subrps-progs}(all-progs)$
 $\wedge \text{subsetp}(progs, all-progs)$
 $\wedge (heap-size \not\leq \text{s-total-heap-reqs}(all-progs, params, heap-size))$
 $\wedge \text{s-restrictedp}(all-progs, params))$
 $\rightarrow \text{lr-programs-properp-1}(\text{lr-compile-programs}(progs,$
 $\text{cdr}(\text{lr-data-seg-table}(all-progs,$
 $params,$
 $heap-size))),$
 $program-names,$
 $\text{cdr}(\text{lr-data-seg-table}(all-progs,$
 $params,$
 $heap-size)))$

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-size* < s-total-heap-reqs (s-progs (*s*), s-params (*s*), *heap-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-stk,
 temp-stk,
 anything,
 data-seg,
 max-ctrl,
 max-temp,
 word-size,
 nil),
 cdr (lr-data-seg-table (s-progs (*s*),
 s-params (*s*),
 heap-size))),
 cdr (lr-data-seg-table (s-progs (*s*),
 s-params (*s*),
 heap-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* ∈ *y*) → definedp (car (*x*), *y*)

THEOREM: all-litatoms-s-formals-member-lr-programs-properp
 ((*prog* ∈ *prog-seg*) ∧ (¬ all-litatoms (formal-vars (*prog*))))
 → (¬ lr-programs-properp-1 (*prog-seg*, *program-names*, *table*))

THEOREM: properp-p-temp-var-dclps-member-lr-programs-properp
 $((prog \in progs)$
 $\wedge$ lr-proper-heapp (*data-seg*)
 $\wedge$ lr-proper-p-areasp (*data-seg*)
 $\wedge$ lr-programs-properp-1 (*progs*, strip-logic-fnames (cdr (*prog-seg*)), *table*))
 $\rightarrow$ proper-p-temp-var-dclsp (temp-var-dcls (*prog*),
p-state (*pc*,
ctrl-stk,
temp-stk,
comp-programs (*prog-seg*),
data-seg,
max-ctrl-stk-size,
max-temp-stk-size,
word-size,
psw))

THEOREM: member-f-definedp-0
 $(f \in alist) \rightarrow \text{definedp}(0, alist)$

THEOREM: member-no-duplicatesp-assoc-equal
 $((x \in alist) \wedge \text{no-duplicatesp}(\text{strip-cars}(alist)) \wedge \text{good-alistp}(alist))$
 $\rightarrow (\text{assoc}(\text{car}(x), alist) = x)$

EVENT: Disable member-f-definedp-0.

THEOREM: lr-proper-exrp-program-body-not-listp
 $(\neg \text{listp}(prog))$
 $\rightarrow (\neg \text{lr-proper-exrp}(t, \text{program-body}(prog), pnames, formals, temps, table))$

THEOREM: good-alistp-lr-programs-properp
 $(\neg \text{good-alistp}(prog-seg))$
 $\rightarrow (\neg \text{lr-programs-properp-1}(prog-seg, program-names, table))$

EVENT: Disable lr-proper-exrp-program-body-not-listp.

THEOREM: proper-p-prog-segmentp-comp-programs-1-helper
(lr-proper-heapp (*data-seg*)
 $\wedge$ lr-proper-p-areasp (*data-seg*)
 $\wedge$ lr-programs-properp-1 (*prog-seg*,
strip-logic-fnames (cdr (*prog-seg*)),
table)
 $\wedge$ all-user-fnamesp (cdr (strip-cars (*prog-seg*)))
 $\wedge$ lr-s-similar-const-table (*table*, *data-seg*)
 $\wedge$ (*prog* $\in$ *prog-seg*))

$\wedge$ 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*)
 $\wedge$ lr-proper-p-areasp (*data-seg*)
 $\wedge$ lr-programs-properp-1 (*prog-seg*,
strip-logic-fnames (cdr (*prog-seg*)),

 $\wedge$ all-user-fnamesp (cdr (strip-cars (*prog-seg*)))
 $\wedge$ lr-s-similar-const-table (*table*, *data-seg*)
 $\wedge$ no-duplicatesp (strip-cars (*prog-seg*))
 $\wedge$ subsetp (*progs*, *prog-seg*)
 $\wedge$ 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*)
 $\wedge$ definedp (area-name (p-pc (*l*)), p-prog-segment (*l*))

```

^ lr-proper-heapp (p-data-segment (l))
^ lr-proper-p-areasp (p-data-segment (l))
^ lr-s-similar-const-table (table, p-data-segment (l))
^ adpp (untag (LR-ANSWER-ADDR), p-data-segment (l))
^ (prog-seg = p-prog-segment (l))
^ all-litatoms (strip-cars (prog-seg))
^ all-user-fnamesp (strip-cars (cdr (p-prog-segment (l))))
→ 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
 $(\text{all-user-fnamesp}(\text{list}) \wedge \text{plistp}(\text{list})) \rightarrow \text{all-litatoms}(\text{list})$

THEOREM: plistp-strip-cars
 $\text{plistp}(\text{strip-cars}(x))$

THEOREM: lr-programs-properp-all-user-fnamesp-strip-cars-cdr
 $\text{lr-programs-properp}(l, \text{table})$
 $\rightarrow \text{all-user-fnamesp}(\text{strip-cars}(\text{cdr}(\text{p-prog-segment}(l))))$

THEOREM: lr-programs-properp-caar-main
 $\text{lr-programs-properp}(l, \text{table}) \rightarrow (\text{caar}(\text{p-prog-segment}(l)) = \text{'main})$

THEOREM: proper-p-prog-segmentp-p-runtime-support-programs
 $(\text{lr-programs-properp}(l, \text{table})$
 $\wedge (\text{p-word-size}(l) \in \mathbf{N})$
 $\wedge (\text{p-word-size}(l) \not\prec \text{S-MAX-SUBR-REQS})$
 $\wedge \text{lr-proper-heapp}(\text{p-data-segment}(l))$
 $\wedge \text{lr-proper-p-areasp}(\text{p-data-segment}(l)))$
 $\rightarrow \text{proper-p-prog-segmentp}(\text{P-RUNTIME-SUPPORT-PROGRAMS}, \text{lr->p}(l))$

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*) $\not\leq$ 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-seg*, *p*)
 $\wedge$ (offset (*addr*) < length (cdr (assoc (area-name (*addr*), *data-seg*))))
 $\wedge$ p-objectp (*value*, *p*))
 $\rightarrow$ proper-p-data-segmentp (deposit (*value*, *addr*, *data-seg*), *p*)

THEOREM: proper-p-data-segmentp-deposit
 (proper-p-data-segmentp (*data-seg*, *p*)
 $\wedge$ adpp (untag (*addr*), *data-seg*)
 $\wedge$ p-objectp (*value*, *p*))
 $\rightarrow$ proper-p-data-segmentp (deposit (*value*, *addr*, *data-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-seg*, *p*)
 $\wedge$ (offset (*addr*) < length (cdr (assoc (area-name (*addr*), *data-seg*))))
 $\wedge$ definedp (area-name (*addr*), *data-seg*))
 $\rightarrow$ p-objectp (fetch (*addr*, *data-seg*), *p*)


```

^ proper-p-data-segmentp (car (ds-tab), p)
^ ((length (cdr (assoc (LR-HEAP-NAME, data-seg)))) - 1)
  < (offset (fetch (LR-FP-ADDR, car (ds-tab)))
      + LR-NODE-SIZE))
^ (p-word-size (p) ∈ ℕ)
^ (p-word-size (p) < S-MAX-SUBR-REQS)
^ same-signature (p-data-segment (p), data-seg)
^ s-restricted-objectp ('list, list (x, y))
→ 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)

```

^ lr-proper-heapp (data-seg)
^ (p-word-size (p) < log (2, number))
^ (number ∈ ℕ)
^ 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) < S-MAX-SUBR-REQS)
^ same-signature (p-data-segment (p), data-seg)
→ 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)

$$\begin{aligned}
& \wedge \text{lr-proper-heapp}(data-seg) \\
& \wedge \text{proper-p-data-segmentp}(data-seg, p) \\
& \wedge ((\text{length}(\text{cdr}(\text{assoc}(\text{LR-HEAP-NAME}, data-seg)))) - 1) \\
& \quad \not\leq (\text{offset}(\text{fetch}(\text{LR-FP-ADDR}, data-seg)) + \text{LR-NODE-SIZE})) \\
& \wedge (\text{p-word-size}(p) \in \mathbf{N}) \\
& \wedge (\text{p-word-size}(p) \not\leq \log(2, \text{LR-TRUE-TAG})) \\
& \wedge \text{same-signature}(\text{p-data-segment}(p), data-seg) \\
\rightarrow & \text{proper-p-data-segmentp}(\text{deposit-a-list}(\text{list}(\text{identity}(\text{tag}('nat, \\
& \hspace{15em} \text{LR-TRUE-TAG})), \\
& \hspace{15em} \text{identity}(\text{tag}('nat, 1))), \\
& \hspace{15em} \text{identity}(\text{LR-UNDEF-ADDR}), \\
& \hspace{15em} \text{identity}(\text{LR-UNDEF-ADDR})), \\
& \hspace{10em} \text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \\
& \hspace{10em} data-seg), \\
& \hspace{10em} data-seg), \\
& p)
\end{aligned}$$

EVENT: Disable proper-p-data-segmentp-deposit-a-list-cons.

THEOREM: s-ws-reqs-flag-not-list-t
 $((flag \neq 'list) \wedge (\neg \text{definedp}(t, table)))$
 $\rightarrow (\text{s-ws-reqs}(flag, t, data-seg, table) = \text{identity}(\log(2, \text{LR-TRUE-TAG})))$

THEOREM: lr-compile-quote-preserves-proper-p-data-segmentp
 $(\text{lr-s-similar-const-table}(table, data-seg)$
 $\wedge \text{s-restricted-objectp}(flag, object)$
 $\wedge \text{lr-proper-heapp}(data-seg)$
 $\wedge \text{lr-proper-p-areasp}(data-seg)$
 $\wedge \text{definedp}(f, table)$
 $\wedge (\text{p-word-size}(p) \in \mathbf{N})$
 $\wedge (\text{p-word-size}(p) \not\leq \text{s-ws-reqs}(flag, object, data-seg, table))$
 $\wedge (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, data-seg),$
 $\hspace{15em} \text{lr-free-list-nodes}(\text{lr-max-node}(data-seg),$
 $\hspace{15em} data-seg),$
 $\hspace{10em} data-seg)$
 $\not\leq \text{s-heap-reqs}(flag, object, data-seg, table))$
 $\wedge \text{proper-p-data-segmentp}(data-seg, p)$
 $\wedge \text{same-signature}(\text{p-data-segment}(p), data-seg)$
 $\rightarrow \text{proper-p-data-segmentp}(\text{car}(\text{lr-compile-quote}(flag,$
 $\hspace{15em} object,$
 $\hspace{15em} data-seg,$
 $\hspace{15em} table))),$
 $p)$

THEOREM: lr-data-seg-table-body-preserves-proper-p-data-segmentp
 (lr-s-similar-const-table (*table*, *data-seg*)
 $\wedge$ s-data-seg-body-restrictedp (*flag*, *body*)
 $\wedge$ 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*) $\not\leq$ s-ws-reqs-body (*flag*, *body*, *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)
 $\not\leq$ s-heap-reqs-body (*flag*, *body*, *data-seg*, *table*))
 $\wedge$ same-signature (p-data-segment (*p*), *data-seg*))
 $\rightarrow$ 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*)
 $\wedge$ s-data-seg-list-restrictedp (*progs*)
 $\wedge$ 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*) $\not\leq$ s-ws-reqs-list (*progs*, *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)
 $\not\leq$ s-heap-reqs-list (*progs*, *data-seg*, *table*))
 $\wedge$ same-signature (p-data-segment (*p*), *data-seg*))
 $\rightarrow$ 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*)
 $\wedge$ s-init-data-seg-restrictedp (*params*)

$$\begin{aligned}
& \wedge \text{lr-proper-heapp} (data-seg) \\
& \wedge \text{lr-proper-p-areasp} (data-seg) \\
& \wedge \text{proper-p-data-segmentp} (data-seg, p) \\
& \wedge \text{definedp} (f, table) \\
& \wedge (p\text{-word-size} (p) \in \mathbf{N}) \\
& \wedge (p\text{-word-size} (p) \not\leq \text{s-init-ws-reqs} (params, data-seg, table)) \\
& \wedge (\text{lr-count-free-nodes} (\text{fetch} (\text{LR-FP-ADDR}, data-seg), \\
& \quad \text{lr-free-list-nodes} (\text{lr-max-node} (data-seg), \\
& \quad \quad data-seg), \\
& \quad \quad data-seg) \\
& \quad \not\leq \text{s-init-heap-reqs} (params, data-seg, table)) \\
& \wedge \text{same-signature} (p\text{-data-segment} (p), data-seg)) \\
\rightarrow & \text{proper-p-data-segmentp} (\text{car} (\text{lr-init-data-seg-table} (params, \\
& \quad data-seg, \\
& \quad table)), \\
& \quad p)
\end{aligned}$$

THEOREM: all-p-objectps-lr-init-heap-contents-helper-helper
 $(heap\text{-size} \neq 0)$

$$\begin{aligned}
\rightarrow & ((\text{identity} (\text{LR-NODE-SIZE}) \\
& + x \\
& + (\text{identity} (\text{LR-NODE-SIZE}) * (heap\text{-size} - 1))) \\
& = (x + (\text{identity} (\text{LR-NODE-SIZE}) * heap\text{-size})))
\end{aligned}$$

THEOREM: all-p-objectps-lr-init-heap-contents-helper

$$\begin{aligned}
& ((p\text{-word-size} (p) \in \mathbf{N}) \\
& \wedge (p\text{-word-size} (p) \not\leq \text{S-MAX-SUBR-REQS}) \\
& \wedge (heap\text{-size} \not\leq 4) \\
& \wedge \text{lr-proper-p-areasp} (p\text{-data-segment} (p)) \\
& \wedge \text{lr-minimum-heapp} (p\text{-data-segment} (p)) \\
& \wedge (\text{type} (addr) = \text{'addr}) \\
& \wedge (\text{caddr} (addr) = \text{nil}) \\
& \wedge \text{listp} (addr) \\
& \wedge (\text{offset} (addr) \in \mathbf{N}) \\
& \wedge \text{lr-boundary-nodep} (addr) \\
& \wedge (\text{area-name} (addr) = \text{LR-HEAP-NAME}) \\
& \wedge \text{definedp} (\text{LR-HEAP-NAME}, p\text{-data-segment} (p)) \\
& \wedge (((\text{LR-NODE-SIZE} * heap\text{-size}) + \text{offset} (addr)) \\
& \quad < \text{length} (\text{cdr} (\text{assoc} (\text{LR-HEAP-NAME}, p\text{-data-segment} (p))))) \\
\rightarrow & \text{all-p-objectps} (\text{lr-init-heap-contents} (addr, heap\text{-size}), p)
\end{aligned}$$

EVENT: Disable all-p-objectps-lr-init-heap-contents-helper-helper.

THEOREM: all-p-objectps-lr-init-heap-contents

$((p\text{-word-size } (p) \in \mathbf{N})$
 $\wedge (p\text{-word-size } (p) \not\leq \text{S-MAX-SUBR-REQS})$
 $\wedge (\text{heap-size} \not\leq 4)$
 $\wedge \text{lr-proper-p-areasp } (p\text{-data-segment } (p))$
 $\wedge \text{same-signature } (p\text{-data-segment } (p), \text{lr-init-data-seg } (\text{heap-size}))$
 $\rightarrow \text{all-p-objectps } (\text{lr-init-heap-contents } (\text{identity } (\text{tag } (' \mathbf{addr},$
 $\qquad \qquad \qquad \text{cons } (\text{LR-HEAP-NAME},$
 $\qquad \qquad \qquad 0))),$
 $\qquad \qquad \qquad \text{heap-size}),$
 $p)$

THEOREM: proper-p-data-segmentp-lr-init-data-seg-helper

$((p\text{-word-size } (p) \in \mathbf{N})$
 $\wedge (p\text{-word-size } (p) \not\leq \text{S-MAX-SUBR-REQS})$
 $\wedge (\text{heap-size} \not\leq 4)$
 $\wedge \text{lr-proper-p-areasp } (p\text{-data-segment } (p))$
 $\wedge \text{same-signature } (p\text{-data-segment } (p), \text{lr-init-data-seg } (\text{heap-size}))$
 $\rightarrow \text{proper-p-data-segmentp } (\text{deposit-a-list } (\text{list } (\text{tag } (' \mathbf{nat}, \text{LR-FALSE-TAG}),$
 $\qquad \qquad \qquad \text{tag } (' \mathbf{nat}, 1),$
 $\qquad \qquad \qquad \text{LR-UNDEF-ADDR},$
 $\qquad \qquad \qquad \text{LR-UNDEF-ADDR}),$
 $\qquad \qquad \qquad \text{LR-F-ADDR},$
 $\qquad \qquad \qquad \text{deposit-a-list } (\text{list } (\text{tag } (' \mathbf{nat},$
 $\qquad \qquad \qquad \text{LR-UNDEFINED-TAG}),$
 $\qquad \qquad \qquad \text{tag } (' \mathbf{nat},$
 $\qquad \qquad \qquad 1),$
 $\qquad \qquad \qquad \text{LR-UNDEF-ADDR},$
 $\qquad \qquad \qquad \text{LR-UNDEF-ADDR}),$
 $\qquad \qquad \qquad \text{LR-UNDEF-ADDR},$
 $\qquad \qquad \qquad \text{list } (\text{list } (\text{area-name } (\text{LR-FP-ADDR}),$
 $\qquad \qquad \qquad \text{add-addr } (\text{LR-F-ADDR},$
 $\qquad \qquad \qquad \text{LR-NODE-SIZE}))),$
 $\qquad \qquad \qquad \text{list } (\text{area-name } (\text{LR-ANSWER-ADDR}),$
 $\qquad \qquad \qquad \text{tag } (' \mathbf{nat},$
 $\qquad \qquad \qquad 0)),$
 $\qquad \qquad \qquad \text{cons } (\text{LR-HEAP-NAME},$
 $\qquad \qquad \qquad \text{lr-init-heap-contents } (\text{tag } (' \mathbf{addr},$
 $\qquad \qquad \qquad \text{cons } (\text{LR-HEAP-NAME},$
 $\qquad \qquad \qquad 0))),$
 $\qquad \qquad \qquad \text{heap-size}))))),$
 $p)$

THEOREM: proper-p-data-segmentp-lr-init-data-seg

$((p\text{-word-size } (p) \in \mathbf{N})$

$\wedge$ (p-word-size (p) $\not\leq$ S-MAX-SUBR-REQS)
 $\wedge$ (*heap-size* $\not\leq$ 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

$((p\text{-word-size } (p) \in \mathbf{N})$
 $\wedge$ (p-word-size (p) $\not\leq$ S-MAX-SUBR-REQS)
 $\wedge$ (*heap-size* $\not\leq$ 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

$(\text{lr-proper-free-listp } (data\text{-seg2}))$
 $\wedge$ 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

$(\text{lr-proper-free-listp } (data\text{-seg2}))$
 $\wedge$ 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

$(\text{lr-proper-free-listp } (data\text{-seg2}))$
 $\wedge$ 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
 $((p\text{-word-size } (p) \in \mathbf{N})$
 $\wedge (p\text{-word-size } (p) \not\leq \text{S-MAX-SUBR-REQS})$
 $\wedge \text{lr-proper-p-areasp } (p\text{-data-segment } (p))$
 $\wedge \text{s-restrictedp } (progs, params)$
 $\wedge (heap\text{-size} \not\leq \text{s-total-heap-reqs } (progs, params, heap\text{-size}))$
 $\wedge (p\text{-word-size } (p) \not\leq \text{s-total-ws-reqs } (progs, params, heap\text{-size}))$
 $\wedge \text{same-signature } (p\text{-data-segment } (p),$
 $\quad \text{car } (lr\text{-data-seg-table } (progs, params, heap\text{-size})))$
 $\rightarrow \text{proper-p-data-segmentp } (\text{car } (lr\text{-data-seg-table } (progs, params, heap\text{-size})),$
 $\quad p)$

THEOREM: adpp-untag-answer-addr-car-lr-data-seg-table
 $\text{adpp } (\text{identity } (\text{untag } (\text{LR-ANSWER-ADDR})),$
 $\quad \text{car } (lr\text{-data-seg-table } (progs, params, heap\text{-size})))$

THEOREM: program-body-assoc-cdr-lr-compile-programs
 $(name \neq \text{caar } (progs))$
 $\rightarrow (\text{program-body } (\text{assoc } (name, \text{cdr } (lr\text{-compile-programs } (progs, table))))$
 $\quad = \text{lr-compile-body } (t,$
 $\quad \quad \text{s-body } (\text{assoc } (name, \text{cdr } (progs))),$
 $\quad \quad \text{lr-make-temp-name-alist } (\text{s-temp-list } (\text{assoc } (name,$
 $\quad \quad \quad \text{cdr } (progs))),$
 $\quad \quad \text{s-formals } (\text{assoc } (name,$
 $\quad \quad \quad \text{cdr } (progs))),$
 $\quad \quad table))$

THEOREM: s-total-ws-reqs-not-lessp-s-max-subr-reqs
 $(word\text{-size} \not\leq \text{s-total-ws-reqs } (progs, params, heap\text{-size}))$
 $\rightarrow (word\text{-size} \not\leq \text{S-MAX-SUBR-REQS})$

THEOREM: proper-p-statep-lr->p-s->lr
 $(\text{s-good-statep } (s, c)$
 $\wedge \text{all-litatoms } (\text{strip-cars } (\text{s-params } (s)))$
 $\wedge (max\text{-ctrl} \not\leq (2 + \text{length } (\text{s-params } (s)) + \text{length } (\text{s-temps } (s))))$
 $\wedge (word\text{-size} \in \mathbf{N})$
 $\wedge (max\text{-temp} < \exp(2, word\text{-size}))$
 $\wedge (max\text{-ctrl} < \exp(2, word\text{-size}))$
 $\wedge (max\text{-temp} \in \mathbf{N})$
 $\wedge (max\text{-ctrl} \in \mathbf{N})$
 $\wedge (heap\text{-size} \not\leq \text{s-total-heap-reqs } (\text{s-progs } (s), \text{s-params } (s), heap\text{-size}))$
 $\wedge \text{s-restrict-subrps-progs } (\text{s-progs } (s))$
 $\wedge \text{litatom } (\text{name } (\text{car } (\text{s-progs } (s))))$
 $\wedge \text{no-duplicatesp } (\text{strip-cars } (\text{s-progs } (s)))$
 $\wedge (\text{strip-cars } (\text{s-params } (s)) = \text{s-formals } (\text{s-prog } (s)))$

$$\begin{aligned}
& \wedge \text{ s-restrictedp } (s\text{-progs } (s), s\text{-params } (s)) \\
& \wedge (\text{word-size} \not\leq \text{s-total-ws-reqs } (s\text{-progs } (s), s\text{-params } (s), \text{heap-size})) \\
& \rightarrow \text{proper-p-statep } (\text{lr} \rightarrow \text{p } (s \rightarrow \text{lr } (s, \text{heap-size}, \text{max-ctrl}, \text{max-temp}, \text{word-size})))
\end{aligned}$$

THEOREM: $\text{lr-programs-properp-s} \rightarrow \text{lr}$

$$\begin{aligned}
& (\text{s-good-statep } (s, c) \\
& \wedge \text{ s-restrict-subrps-progs } (s\text{-progs } (s)) \\
& \wedge \text{ s-restrictedp } (s\text{-progs } (s), s\text{-params } (s)) \\
& \wedge (\text{heap-size} \not\leq \text{s-total-heap-reqs } (s\text{-progs } (s), s\text{-params } (s), \text{heap-size})) \\
& \wedge \text{litatom } (\text{name } (\text{car } (s\text{-progs } (s)))) \\
& \rightarrow \text{lr-programs-properp } (s \rightarrow \text{lr } (s, \text{heap-size}, \text{max-ctrl}, \text{max-temp}, \text{word-size}), \\
& \quad \text{cdr } (\text{lr-data-seg-table } (s\text{-progs } (s), \\
& \quad \quad \quad s\text{-params } (s), \\
& \quad \quad \quad \text{heap-size})))
\end{aligned}$$

DEFINITION:

$$\begin{aligned}
& \text{lr-total-heap-reqs } (expr, alist, program\text{-names}, \text{heap-size}, c) \\
& = (\text{s-total-heap-reqs } (s\text{-progs } (\text{logic} \rightarrow s (expr, alist, program\text{-names})), \\
& \quad \quad \quad alist, \\
& \quad \quad \quad \text{heap-size}) \\
& \quad + \text{ s-eval-heap-r } (t, \text{logic} \rightarrow s (expr, alist, program\text{-names}), c))
\end{aligned}$$

DEFINITION:

$$\begin{aligned}
& \text{lr-max-ctrl-reqs } (expr, alist, program\text{-names}, c) \\
& = (2 \\
& \quad + \text{length } (s\text{-params } (\text{logic} \rightarrow s (expr, alist, program\text{-names}))) \\
& \quad + \text{length } (s\text{-temps } (\text{logic} \rightarrow s (expr, alist, program\text{-names}))) \\
& \quad + \text{ s-eval-ctrl-r } (t, \text{logic} \rightarrow s (expr, alist, program\text{-names}), c))
\end{aligned}$$

DEFINITION:

$$\begin{aligned}
& \text{lr-max-temp-reqs } (expr, alist, program\text{-names}, c) \\
& = \text{ s-eval-temp-r } (t, \text{logic} \rightarrow s (expr, alist, program\text{-names}), c)
\end{aligned}$$

DEFINITION:

$$\begin{aligned}
& \text{lr-max-word-size-reqs } (expr, alist, program\text{-names}, \text{heap-size}, c) \\
& = \max (\text{s-total-ws-reqs } (s\text{-progs } (\text{logic} \rightarrow s (expr, alist, program\text{-names})), \\
& \quad \quad \quad s\text{-params } (\text{logic} \rightarrow s (expr, alist, program\text{-names})), \\
& \quad \quad \quad \text{heap-size}), \\
& \quad \text{ s-eval-ws-r } (t, \text{logic} \rightarrow s (expr, alist, program\text{-names}), c))
\end{aligned}$$

THEOREM: $\text{lr-eval-s-eval-equivalence-s} \rightarrow \text{lr}$

$$\begin{aligned}
& \text{let } s\text{-lr} \text{ be } s \rightarrow \text{lr } (s, \text{heap-size}, \text{max-ctrl}, \text{max-temp}, \text{word-size}) \\
& \text{in} \\
& (\text{proper-p-statep } (\text{lr} \rightarrow \text{p } (s\text{-lr})) \\
& \quad \wedge \text{ lr-programs-properp } (s\text{-lr},
\end{aligned}$$

$$\begin{aligned}
& \text{cdr}(\text{lr-data-seg-table}(\text{s-progs}(s), \\
& \quad \text{s-params}(s), \\
& \quad \text{heap-size})) \\
\wedge \quad & \text{lr-s-similar-statesp}(\text{s-params}(s), \\
& \quad \text{s-temps}(s), \\
& \quad s\text{-lr}, \\
& \quad \text{cdr}(\text{lr-data-seg-table}(\text{s-progs}(s), \\
& \quad \quad \text{s-params}(s), \\
& \quad \quad \text{heap-size}))) \\
\wedge \quad & \text{s-restrictedp}(\text{s-progs}(s), \text{s-params}(s)) \\
\wedge \quad & (\text{heap-size} \not\leq \text{s-total-heap-reqs}(\text{s-progs}(s), \\
& \quad \text{s-params}(s), \\
& \quad \text{heap-size})) \\
\wedge \quad & \text{s-good-statep}(s, c) \\
\wedge \quad & (\text{p-psw}(\text{lr-eval}(\mathbf{t}, s\text{-lr}, c)) = \text{'run}) \\
\wedge \quad & (\text{s-pname}(s) = \text{name}(\text{car}(\text{s-progs}(s)))) \\
\wedge \quad & (\text{s-pos}(s) = \mathbf{nil}) \\
\rightarrow \quad & \text{lr-valp}(\text{s-ans}(\text{s-eval}(\mathbf{t}, s, c)), \\
& \quad \text{car}(\text{p-temp-stk}(\text{lr-eval}(\mathbf{t}, s\text{-lr}, c))), \\
& \quad \text{p-data-segment}(\text{lr-eval}(\mathbf{t}, s\text{-lr}, c))) \text{ endlet}
\end{aligned}$$

THEOREM: same-signature-car-lr-data-seg-table
 $(\text{heap-size} \not\leq \text{s-total-heap-reqs}(\text{progs}, \text{params}, \text{heap-size}))$
 $\rightarrow$ same-signature($\text{lr-init-data-seg}(\text{heap-size})$,
 $\text{car}(\text{lr-data-seg-table}(\text{progs}, \text{params}, \text{heap-size}))$)

THEOREM: lr-max-node-car-lr-data-seg-table
 $(\text{heap-size} \not\leq \text{s-total-heap-reqs}(\text{progs}, \text{params}, \text{heap-size}))$
 $\rightarrow$ ($\text{lr-max-node}(\text{car}(\text{lr-data-seg-table}(\text{progs}, \text{params}, \text{heap-size})))$
 $=$ tag('addr,
 $\text{cons}(\text{identity}(\text{LR-HEAP-NAME})$,
 $\text{identity}(\text{LR-NODE-SIZE}) * \text{heap-size}))$)

THEOREM: lr-count-free-nodes-lr-data-seg-table-list-s-heap-reqs-help
 $(\text{lr-proper-free-listp}(\text{data-seg}))$
 $\wedge$ $\text{lr-proper-p-areasp}(\text{data-seg})$
 $\wedge$ $\text{definedp}(\text{LR-HEAP-NAME}, \text{data-seg})$
 $\wedge$ $\text{lr-minimum-heap}(\text{data-seg})$
 $\wedge$ $\text{lr-boundary-nodep}(\text{lr-max-node}(\text{data-seg}))$
 $\wedge$ ($\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg})$,
 $\text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg})$,
 $\text{data-seg})$,
 $\text{data-seg})$
 $\not\leq$ $\text{s-heap-reqs-body}(\mathbf{t}, \text{body}, \text{data-seg}, \text{table}))$
 $\rightarrow$ ($\text{s-heap-reqs-body}(\mathbf{t}, \text{body}, \text{data-seg}, \text{table})$)

$$\begin{aligned}
& + \text{lr-count-free-nodes}(\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \\
& \quad \text{car}(\text{lr-data-seg-table-body}(\mathbf{t}, \\
& \quad \quad \text{body}, \\
& \quad \quad \text{data-seg}, \\
& \quad \quad \text{table}))), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{car}(\text{lr-data-seg-table-body}(\mathbf{t}, \\
& \quad \quad \quad \text{body}, \\
& \quad \quad \quad \text{data-seg}, \\
& \quad \quad \quad \text{table}))), \\
& \quad \text{car}(\text{lr-data-seg-table-body}(\mathbf{t}, \\
& \quad \quad \text{body}, \\
& \quad \quad \text{data-seg}, \\
& \quad \quad \text{table})))) \\
& = \text{lr-count-free-nodes}(\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{data-seg}), \\
& \quad \text{data-seg}))
\end{aligned}$$

THEOREM: lr-count-free-nodes-lr-data-seg-table-list-s-heap-reqs

$$\begin{aligned}
& (\text{lr-proper-free-listp}(\text{data-seg}) \\
& \wedge \text{lr-proper-p-areasp}(\text{data-seg}) \\
& \wedge \text{definedp}(\text{LR-HEAP-NAME}, \text{data-seg}) \\
& \wedge \text{lr-minimum-heapp}(\text{data-seg}) \\
& \wedge \text{lr-boundary-nodep}(\text{lr-max-node}(\text{data-seg})) \\
& \wedge (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{data-seg}), \\
& \quad \quad \text{data-seg}) \\
& \quad \not\leq \text{s-heap-reqs-list}(\text{progs}, \text{data-seg}, \text{table}))) \\
\rightarrow & ((\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \\
& \quad \text{car}(\text{lr-data-seg-table-list}(\text{progs}, \\
& \quad \quad \text{data-seg}, \\
& \quad \quad \text{table}))), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{car}(\text{lr-data-seg-table-list}(\text{progs}, \\
& \quad \quad \quad \text{data-seg}, \\
& \quad \quad \quad \text{table}))), \\
& \quad \text{car}(\text{lr-data-seg-table-list}(\text{progs}, \text{data-seg}, \text{table}))) \\
& + \text{s-heap-reqs-list}(\text{progs}, \text{data-seg}, \text{table})) \\
& = \text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{data-seg}), \\
& \quad \text{data-seg}))
\end{aligned}$$

THEOREM: lr-max-node-car-lr-data-seg-table-list
 (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-max-node (car (lr-data-seg-table-list (*progs*, *data-seg*, *table*)))
 $=$ lr-max-node (*data-seg*))

THEOREM: lr-count-free-nodes-lr-data-seg-table-list-s-heap-reqs-1
 (lr-proper-heapp (*data-seg*)
 $\wedge$ 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)
 $\not\leq$ s-heap-reqs-list (*progs*, *data-seg*, *table*))
 $\wedge$ (*max-addr* = lr-max-node (car (lr-data-seg-table-list (*progs*,
data-seg,
table))))))
 $\rightarrow$ (lr-count-free-nodes (fetch (identity (LR-FP-ADDR),
 car (lr-data-seg-table-list (*progs*,
data-seg,
table))),
 lr-free-list-nodes (*max-addr*,
 car (lr-data-seg-table-list (*progs*,
data-seg,
table))),
 car (lr-data-seg-table-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)
 $-$ s-heap-reqs-list (*progs*, *data-seg*, *table*)))

THEOREM: lr-count-free-nodes-lr-init-data-seg-table-s-heap-reqs-1
 (lr-proper-heapp (*data-seg*)
 $\wedge$ 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)
 $\not\leq$ s-init-heap-reqs (*params*, *data-seg*, *table*))
 $\wedge$ (*max-addr* = lr-max-node (car (lr-init-data-seg-table (*params*,
data-seg,
table))))

$$\begin{aligned}
& \rightarrow (\text{lr-count-free-nodes}(\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \\
& \quad \text{car}(\text{lr-init-data-seg-table}(\text{params}, \\
& \quad \quad \text{data-seg}, \\
& \quad \quad \text{table}))), \\
& \quad \text{lr-free-list-nodes}(\text{max-addr}, \\
& \quad \quad \text{car}(\text{lr-init-data-seg-table}(\text{params}, \\
& \quad \quad \quad \text{data-seg}, \\
& \quad \quad \quad \text{table}))), \\
& \quad \text{car}(\text{lr-init-data-seg-table}(\text{params}, \text{data-seg}, \text{table}))) \\
& = (\text{lr-count-free-nodes}(\text{fetch}(\text{LR-FP-ADDR}, \text{data-seg}), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{data-seg}), \\
& \quad \quad \text{data-seg}), \\
& \quad \quad \text{data-seg}) \\
& \quad - \text{s-init-heap-reqs}(\text{params}, \text{data-seg}, \text{table}))
\end{aligned}$$

THEOREM: not-lessp-lr-count-free-nodes-lr-data-seg-table-heap-r
 $((\text{heap-size} \not\prec (\text{s-total-heap-reqs}(\text{progs}, \text{params}, \text{heap-size}) + x))$
 $\wedge \text{s-restrictedp}(\text{progs}, \text{params}))$

$$\begin{aligned}
& \rightarrow (\text{lr-count-free-nodes}(\text{fetch}(\text{identity}(\text{LR-FP-ADDR}), \\
& \quad \text{car}(\text{lr-data-seg-table}(\text{progs}, \\
& \quad \quad \text{params}, \\
& \quad \quad \text{heap-size}))), \\
& \quad \text{lr-free-list-nodes}(\text{lr-max-node}(\text{car}(\text{lr-data-seg-table}(\text{progs}, \\
& \quad \quad \quad \text{params}, \\
& \quad \quad \quad \text{heap-size}))), \\
& \quad \quad \text{car}(\text{lr-data-seg-table}(\text{progs}, \\
& \quad \quad \quad \text{params}, \\
& \quad \quad \quad \text{heap-size}))), \\
& \quad \quad \text{car}(\text{lr-data-seg-table}(\text{progs}, \text{params}, \text{heap-size}))) \\
& \not\prec x)
\end{aligned}$$

THEOREM: lr-eval-s-eval-flag-run-s->lr
let $s\text{-lr}$ **be** $s\text{->lr}(s, \text{heap-size}, \text{max-ctrl}, \text{max-temp}, \text{word-size})$
in
 $(\text{proper-p-statep}(\text{lr->p}(s\text{-lr}))$
 $\wedge \text{lr-programs-properp}(s\text{-lr},$
 $\quad \text{cdr}(\text{lr-data-seg-table}(\text{s-progs}(s),$
 $\quad \quad \text{s-params}(s),$
 $\quad \quad \text{heap-size})))$
 $\wedge \text{lr-s-similar-statep}(\text{s-params}(s),$
 $\quad \text{s-temps}(s),$
 $\quad s\text{-lr},$
 $\quad \text{cdr}(\text{lr-data-seg-table}(\text{s-progs}(s),$

```

                                                    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  $\not\leq$  (s-total-heap-reqs (s-progs (s),
                                   s-params (s),
                                   heap-size)
                                   + s-eval-heap-r (t, s, c)))
 $\wedge$  (max-ctrl  $\not\leq$  (p-ctrl-stk-size (p-ctrl-stk (s-lr))
                        + s-eval-ctrl-r (t, s, c)))
 $\wedge$  (max-temp  $\not\leq$  s-eval-temp-r (t, s, c))
 $\wedge$  (word-size  $\not\leq$  s-eval-ws-r (t, s, c))
 $\wedge$  (word-size  $\not\leq$  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-addr (strip-cdrs (lr-make-initial-temps (x)))

THEOREM: lr-s-similar-temps-make-temps-entries-initial-temps
(all-undef-addr (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-addr (object-list, table)
= if listp (object-list)
  then cons (cdr (assoc (car (object-list), table)),
            object-addr (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)
  s-heap-reqs (flag, object, data-seg, table)))
→ if flag = 'list
then lr-check-result1 (object,
                      object-addr (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)
  ∧ lr-good-pointer-tablep (table, data-seg)
  ∧ s-init-data-seg-restrictedp (params)
  ∧ lr-proper-heapp (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)
      s-init-heap-reqs (params, data-seg, table))
  ∧ lr-valp (value, addr, data-seg))
→ 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)
  ∧ lr-s-similar-const-table (table, data-seg)
  ∧ s-restricted-objectp (t, object)
  ∧ lr-proper-heapp (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)

```

$\not\rightarrow$ s-heap-reqs ($\mathbf{t}$, $object$, $data-seg$, $table$)))
 $\rightarrow$ lr-valp ($object$,
 $\quad$ cdr (assoc ($object$,
 $\quad\quad$ cdr (lr-compile-quote ($\mathbf{t}$, $object$, $data-seg$, $table$))))),
 $\quad$ car (lr-compile-quote ($\mathbf{t}$, $object$, $data-seg$, $table$)))

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 ($\mathbf{f}$, $table$)
 $\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, $data-seg$),
 $\quad$ lr-free-list-nodes (lr-max-node ($data-seg$),
 $\quad\quad$ $data-seg$),
 $\quad\quad$ $data-seg$)
 $\not\rightarrow$ s-init-heap-reqs ($params$, $data-seg$, $table$)))
 $\rightarrow$ lr-s-similar-params ($params$,
 $\quad$ pair-formals-with-addresses ($params$,
 $\quad\quad$ cdr (lr-init-data-seg-table ($params$,
 $\quad\quad\quad$ $data-seg$,
 $\quad\quad\quad$ $table$))),
 $\quad$ 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-pointer-tablep ($table$, $data-seg$)
 $\wedge$ s-data-seg-body-restrictedp ($flag$, $body$)
 $\wedge$ lr-proper-heapp ($data-seg$)
 $\wedge$ definedp ($\mathbf{f}$, $table$)
 $\wedge$ (lr-count-free-nodes (fetch (LR-FP-ADDR, $data-seg$),
 $\quad$ lr-free-list-nodes (lr-max-node ($data-seg$),
 $\quad\quad$ $data-seg$),
 $\quad\quad$ $data-seg$)
 $\not\rightarrow$ s-heap-reqs-body ($flag$, $body$, $data-seg$, $table$))
 $\wedge$ lr-valp ($value$, $addr$, $data-seg$)
 $\rightarrow$ lr-valp ($value$,
 $\quad$ $addr$,
 $\quad$ 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-pointer-tablep ($table$, $data-seg$)
 $\wedge$ s-data-seg-list-restrictedp ($progs$)
 $\wedge$ 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)
 $\not\wedge$ s-heap-reqs-list (*progs*, *data-seg*, *table*))
 $\wedge$ 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*)
 $\wedge$ lr-good-pointerp-tablep (*table*, *data-seg*)
 $\wedge$ s-data-seg-list-restrictedp (*progs*)
 $\wedge$ 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)
 $\not\wedge$ s-heap-reqs-list (*progs*, *data-seg*, *table*))
 $\wedge$ 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*) $\wedge$ 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*,

$$\begin{aligned}
& \text{data-seg,} \\
& \text{table})) \\
= & \text{pair-formals-with-addresses} (params, table))
\end{aligned}$$

THEOREM: all-definedp-strip-cdrs-lr-init-data-seg-table
 $(\text{lr-s-similar-const-table} (table, data-seg)$
 $\wedge \text{lr-proper-p-areasp} (data-seg)$
 $\wedge \text{lr-proper-heapp} (data-seg)$
 $\wedge \text{definedp} (f, table)$
 $\wedge \text{s-init-data-seg-restrictedp} (params)$
 $\wedge (\text{lr-count-free-nodes} (\text{fetch} (\text{LR-FP-ADDR}, data-seg),$
 $\text{lr-free-list-nodes} (\text{lr-max-node} (data-seg),$
 $data-seg),$
 $data-seg)$
 $\not\wedge \text{s-init-heap-reqs} (params, data-seg, table)))$
 $\rightarrow \text{all-definedp} (\text{strip-cdrs} (params),$
 $\text{cdr} (\text{lr-init-data-seg-table} (params, data-seg, table)))$

THEOREM: lr-s-similar-params-pair-formals-with-addresses
 $((\text{heap-size} \not\wedge \text{s-total-heap-reqs} (progs, params, \text{heap-size}))$
 $\wedge \text{s-restrictedp} (progs, params))$
 $\rightarrow \text{lr-s-similar-params} (params,$
 $\text{pair-formals-with-addresses} (params,$
 $\text{cdr} (\text{lr-data-seg-table} (progs,$
 $params,$
 $\text{heap-size}))),$
 $\text{car} (\text{lr-data-seg-table} (progs, params, \text{heap-size})))$

THEOREM: lr-s-similar-statesp-s->lr-lr-data-seg-table
 $((\text{heap-size} \not\wedge \text{s-total-heap-reqs} (\text{s-progs} (s), \text{s-params} (s), \text{heap-size}))$
 $\wedge \text{s-restrictedp} (\text{s-progs} (s), \text{s-params} (s))$
 $\wedge (\text{strip-cars} (\text{s-params} (s)) = \text{s-formals} (\text{car} (\text{s-progs} (s))))$
 $\wedge (\text{s-pname} (s) = \text{name} (\text{car} (\text{s-progs} (s))))$
 $\wedge (\text{s-temps} (s) = \text{make-temps-entries} (\text{s-temp-list} (\text{car} (\text{s-progs} (s)))))$
 $\rightarrow \text{lr-s-similar-statesp} (\text{s-params} (s),$
 $\text{s-temps} (s),$
 $\text{s->lr} (s, \text{heap-size}, \text{max-ctrl}, \text{max-temp}, \text{word-size}),$
 $\text{cdr} (\text{lr-data-seg-table} (\text{s-progs} (s),$
 $\text{s-params} (s),$
 $\text{heap-size})))$

THEOREM: p-ctrl-stk-size-p-ctrl-stk-s->lr
 $\text{p-ctrl-stk-size} (\text{p-ctrl-stk} (\text{s->lr} (s, \text{heap-size}, \text{max-ctrl}, \text{max-temp}, \text{word-size})))$
 $= (2 + \text{length} (\text{s-params} (s)) + \text{length} (\text{s-temps} (s)))$

THEOREM: $s \rightarrow lr\text{-ok}$

```

let  $s\text{-}lr$  be  $s \rightarrow lr(s, heap\text{-}size, max\text{-}ctrl, max\text{-}temp, word\text{-}size)$ 
in
  (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-temps( $s$ )
      = make-temps-entries(s-temp-list(car(s-progs( $s$ ))))
     $\wedge$  s-check-temps-setp(s-temps( $s$ ))
     $\wedge$  s-restrictedp(s-progs( $s$ ), s-params( $s$ ))
     $\wedge$  ( $heap\text{-}size \not\leq$  (s-total-heap-reqs(s-progs( $s$ ),
      s-params( $s$ ),
       $heap\text{-}size$ )
      + s-eval-heap-r( $t, s, c$ )))
     $\wedge$  s-restrict-subrps-progs(s-progs( $s$ ))
     $\wedge$  ( $max\text{-}ctrl \not\leq$  (2
      + length(s-params( $s$ ))
      + length(s-temps( $s$ ))
      + s-eval-ctrl-r( $t, s, c$ )))
     $\wedge$  ( $max\text{-}ctrl \in \mathbf{N}$ )
     $\wedge$  ( $max\text{-}ctrl < \exp(2, word\text{-}size)$ )
     $\wedge$  ( $max\text{-}temp \in \mathbf{N}$ )
     $\wedge$  ( $max\text{-}temp \not\leq$  s-eval-temp-r( $t, s, c$ ))
     $\wedge$  ( $max\text{-}temp < \exp(2, word\text{-}size)$ )
     $\wedge$  ( $word\text{-}size \not\leq$  max(s-total-ws-reqs(s-progs( $s$ ),
      s-params( $s$ ),
       $heap\text{-}size$ ),
      s-eval-ws-r( $t, s, c$ )))
     $\wedge$  ( $word\text{-}size \in \mathbf{N}$ )
     $\wedge$  litatom(name(car(s-progs( $s$ ))))
     $\wedge$  (s-pname( $s$ ) = name(car(s-progs( $s$ ))))
     $\wedge$  (s-pos( $s$ ) = nil)
     $\wedge$  all-litatoms(strip-cars(s-params( $s$ )))
     $\wedge$  no-duplicatesp(strip-cars(s-progs( $s$ )))
     $\wedge$  (strip-cars(s-params( $s$ )) = s-formals(car(s-progs( $s$ ))))
     $\wedge$  (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( $t, s\text{-}lr, c$ ))),
    p-data-segment(lr-eval( $t, s\text{-}lr, c$ ))) endlet

```

THEOREM: s-good-state-logic- $\rightarrow s$

(l-proper-programsp($prog\text{-}names$))

$\wedge$ 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*)
 → s-data-seg-body-restrictedp (*flag*, *body*)

THEOREM: l-data-seg-body-restrictedp-delete-all
 l-data-seg-list-restrictedp (*pnames*)
 → 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*) ∧ l-data-seg-body-restrictedp (**t**, *expr*))
 → s-data-seg-list-restrictedp (s-progs (logic->s (*expr*, *alist*, *pnames*)))

THEOREM: l-restrict-subrps-s-restrict-subrps
 (l-restrict-subrps (*flag*, *expr*)
 ∧ l-proper-exprp (*flag*, *expr*, *program-names*, *formals*))
 → s-restrict-subrps (*flag*, *expr*)

THEOREM: l-restrict-subrps-progs-delete-all
 l-restrict-subrps-progs (*pnames*)
 → l-restrict-subrps-progs (delete-all (*name*, *pnames*))

THEOREM: l-restrict-subrps-progs-s-restrict-subrps-progs
 (l-restrict-subrps-progs (*pnames*)
 ∧ l-proper-programsp-1 (*pnames*, *program-names*))
 → 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*)
 ∧ l-restrict-subrps (**t**, *expr*)
 ∧ l-proper-exprp (**t**, *expr*, *program-names*, *formals*)
 ∧ l-proper-programsp (*pnames*))
 → 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*) ∧ all-litatoms-not-plist (*pnames*))
 → (definedp (user-fname (*name*), s-construct-programs (*pnames*))
 = (*name* ∈ *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*))
 $\wedge$ l-proper-programsp (*pnames*)
 $\wedge$ all-litatoms (strip-cars (*alist*))
 $\wedge$ l-data-seg-body-restrictedp (*t*, *expr*)
 $\wedge$ l-restrictedp (*pnames*, *alist*)
 $\wedge$ v&c\$ (*t*, *expr*, *alist*)
 $\wedge$ (cdr (v&c\$ (*t*, *expr*, *alist*)) < *c*)
 $\wedge$ l-restrict-subrps (*t*, *expr*)
 $\wedge$ l-restrict-subrps-progs (*pnames*)
 $\wedge$ (*heap-size* $\not\leq$ lr-total-heap-reqs (*expr*, *alist*, *pnames*, *heap-size*, *c*))
 $\wedge$ (*max-ctrl* $\not\leq$ lr-max-ctrl-reqs (*expr*, *alist*, *pnames*, *c*))
 $\wedge$ (*max-ctrl* < exp (2, *word-size*))
 $\wedge$ (*max-ctrl* $\in \mathbf{N}$)
 $\wedge$ (*max-temp* $\not\leq$ lr-max-temp-reqs (*expr*, *alist*, *pnames*, *c*))
 $\wedge$ (*max-temp* < exp (2, *word-size*))
 $\wedge$ (*max-temp* $\in \mathbf{N}$)
 $\wedge$ (*word-size* $\not\leq$ lr-max-word-size-reqs (*expr*, *alist*, *pnames*, *heap-size*, *c*))
 $\wedge$ (*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)
        endif
    endif

```

```

+ 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*)
 $\wedge$ 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)
 $\wedge$ 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*)
 $\wedge$ lr-good-pointerps (*temp-stk*, *data-seg*)
 $\wedge$ lr-proper-heapp (*data-seg*)
 $\wedge$ lr-good-pointerp-tablep (*table*, *data-seg*))

EVENT: Disable lr-p-proper-statep.

THEOREM: definedp-cdr-assoc-lr-good-pointerps
 ((*addr* $\in$ *list*) $\wedge$ 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*))
 $\wedge$ 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-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*)
 $\wedge$ 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*)
 $\wedge$ listp (*temp-stk*)
 $\wedge$ listp (*ctrl-stk*)
 $\wedge$ (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*))
 $\wedge$ good-posp (*flag*, *pos*, program-body (p-current-program (*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*, 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*))
 $\wedge$ lr-p-proper-statep (*temp-stk*, p-ctrl-stk (*l*), *data-seg*, *table*)
 $\wedge$ good-posp1 (offset (p-pc (*l*)), program-body (p-current-program (*l*)))
 $\wedge$ lr-programs-properp (*l*, *table*)
 $\wedge$ listp (lr-expr (*l*))
 $\wedge$ ((car (lr-expr (*l*)) = S-TEMP-TEST)
 $\vee$ (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
 $((object \in \text{strip-cdrs}(table)) \wedge \text{lr-good-pointerp-tablep}(table, data-seg))$
 $\rightarrow \text{lr-good-pointerp}(object, data-seg)$

THEOREM: lr-p-proper-statep-p-temps-stk-lr-push-tstk-quote
 $(\text{listp}(\text{lr-expr}(l))$
 $\wedge (\text{car}(\text{lr-expr}(l)) = \text{'quote})$
 $\wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{lr-programs-properp}(l, table)$
 $\wedge (\text{p-psw}(\text{lr-push-tstk}(l, \text{cadr}(\text{lr-expr}(l)))) = \text{'run})$
 $\wedge \text{lr-p-proper-statep}(\text{p-temp-stk}(l), \text{ctrl-stk}, \text{data-seg}, table))$
 $\rightarrow \text{lr-p-proper-statep}(\text{p-temp-stk}(\text{lr-push-tstk}(l, \text{cadr}(\text{lr-expr}(l)))),$
 $\text{ctrl-stk},$
 $\text{data-seg},$
 $table)$

THEOREM: p-run-subr-preserves-lr-good-pointerp-tablep
 $(\text{listp}(\text{lr-expr}(l))$
 $\wedge (\text{car}(\text{lr-expr}(l)) \neq \text{'if})$
 $\wedge (\text{car}(\text{lr-expr}(l)) \neq \text{'quote})$
 $\wedge (\text{p-psw}(new-l) = \text{'run})$
 $\wedge \text{subrp}(\text{car}(\text{lr-expr}(l)))$
 $\wedge \text{lr-good-pointerp-tablep}(table2, \text{p-data-segment}(new-l))$
 $\wedge \text{proper-p-statep}(\text{lr->p}(l))$
 $\wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{lr-programs-properp}(l, table1)$
 $\wedge \text{lr-programs-properp}(new-l, table1)$
 $\wedge (\text{p-psw}(\text{p-run-subr}(\text{car}(\text{lr-expr}(l)),$
 $\text{p-set-pc}(\text{lr->p}(new-l), \text{lr-return-pc}(l))))$
 $= \text{'run})$
 $\wedge \text{lr-proper-heapp}(\text{p-data-segment}(new-l))$
 $\wedge \text{proper-p-statep}(\text{lr->p}(new-l))$
 $\wedge (\text{p-prog-segment}(new-l) = \text{p-prog-segment}(l))$
 $\rightarrow \text{lr-good-pointerp-tablep}(table2,$
 $\text{p-data-segment}(\text{p-run-subr}(\text{car}(\text{lr-expr}(l)),$
 $\text{p-set-pc}(\text{lr->p}(new-l),$
 $\text{lr-return-pc}(l))))))$

THEOREM: lr-good-pointerps-deposit-free-ptr
 $\text{lr-good-pointerps}(list, \text{deposit}(\text{anything}, \text{identity}(\text{LR-FP-ADDR}), data-seg))$
 $= \text{lr-good-pointerps}(list, data-seg)$

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)
 ∧ (caddr (*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-pointersp-deposit-a-list-node
 (lr-good-pointersp (*list*, *data-seg*)
 ∧ (type (*addr*) = 'addr)
 ∧ (caddr (*addr*) = nil)
 ∧ listp (*addr*)
 ∧ adpp (untag (*addr*), *data-seg*)
 ∧ lr-boundary-nodep (*addr*)
 ∧ (area-name (*addr*) = 'heap)
 ∧ (type (*ref-count*) = 'nat))
 → lr-good-pointersp (*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)
 ∧ (caddr (*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*))

$$\begin{aligned}
& \wedge \text{listp}(\text{lr-expr}(l)) \\
& \wedge \text{subrp}(\text{car}(\text{lr-expr}(l))) \\
& \wedge (\text{car}(\text{lr-expr}(l)) \neq \text{'if}) \\
& \wedge (\text{car}(\text{lr-expr}(l)) \neq \text{'quote}) \\
& \wedge \text{proper-p-statep}(\text{lr->p}(l)) \\
& \wedge \text{lr-proper-heapp}(\text{p-data-segment}(\text{lr-eval}(\text{'list}, \text{lr-set-pos}(l, pos), c))) \\
& \wedge \text{lr-proper-ctrl-stkp}(\text{p-ctrl-stk}(\text{lr-eval}(\text{'list}, \text{lr-set-pos}(l, pos), c)), \\
& \quad \text{p-data-segment}(\text{lr-eval}(\text{'list}, \\
& \quad \quad \text{lr-set-pos}(l, pos), \\
& \quad \quad \quad c))) \\
& \wedge (\text{p-psw}(\text{lr-eval}(\text{'list}, \text{lr-set-pos}(l, pos), c)) = \text{'run}) \\
& \wedge (\text{p-psw}(\text{p-run-subr}(\text{car}(\text{lr-expr}(l)), \\
& \quad \text{p-set-pc}(\text{lr->p}(\text{lr-eval}(\text{'list}, \text{lr-set-pos}(l, pos), c)), \\
& \quad \quad \text{lr-return-pc}(l)))) \\
& \quad = \text{'run}) \\
& \wedge (pos = \text{dv}(\text{offset}(\text{p-pc}(l), 1))) \\
\rightarrow & \text{lr-proper-ctrl-stkp}(\text{p-ctrl-stk}(\text{lr-eval}(\text{'list}, \text{lr-set-pos}(l, pos), c)), \\
& \quad \text{p-data-segment}(\text{p-run-subr}(\text{car}(\text{lr-expr}(l)), \\
& \quad \quad \text{p-set-pc}(\text{lr->p}(\text{lr-eval}(\text{'list}, \\
& \quad \quad \quad \text{lr-set-pos}(l, \\
& \quad \quad \quad \quad pos), \\
& \quad \quad \quad \quad c)), \\
& \quad \quad \quad \text{lr-return-pc}(l))))))
\end{aligned}$$

THEOREM: lr-good-pointerps-cons-lr-f-addr-lr-proper-heapp
 $(\text{lr-good-pointerps}(list, data-seg)$
 $\wedge \text{lr-proper-heapp}(data-seg)$
 $\wedge \text{lr-proper-p-areasp}(data-seg))$
 $\rightarrow \text{lr-good-pointerps}(\text{cons}(\text{identity}(\text{LR-F-ADDR}), list), data-seg)$

THEOREM: lr-good-pointerps-cons-lr-t-addr-lr-proper-heapp
 $(\text{lr-good-pointerps}(list, data-seg)$
 $\wedge \text{lr-proper-heapp}(data-seg)$
 $\wedge \text{lr-proper-p-areasp}(data-seg))$
 $\rightarrow \text{lr-good-pointerps}(\text{cons}(\text{identity}(\text{LR-T-ADDR}), list), data-seg)$

THEOREM: lr-good-pointerps-cons-lr-0-addr-lr-proper-heapp
 $(\text{lr-good-pointerps}(list, data-seg)$
 $\wedge \text{lr-proper-heapp}(data-seg)$
 $\wedge \text{lr-proper-p-areasp}(data-seg))$
 $\rightarrow \text{lr-good-pointerps}(\text{cons}(\text{identity}(\text{LR-0-ADDR}), list), data-seg)$

THEOREM: lr-good-pointerps-cdr
 $\text{lr-good-pointerps}(list, data-seg) \rightarrow \text{lr-good-pointerps}(\text{cdr}(list), data-seg)$

THEOREM: lr-good-pointerps-cons-fetch-car-temp-stk-cdr-car
 (lr-good-pointerps (*temp-stk*, *data-seg*)
 $\wedge$ lr-proper-heapp (*data-seg*)
 $\wedge$ lr-proper-p-areasp (*data-seg*)
 $\wedge$ (length (*temp-stk*) $\not\leq$ 1)
 $\wedge$ (fetch (car (*temp-stk*), *data-seg*) = tag ('nat, LR-CONS-TAG)))
 $\rightarrow$ 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*)
 $\wedge$ lr-proper-heapp (*data-seg*)
 $\wedge$ lr-proper-p-areasp (*data-seg*)
 $\wedge$ (length (*temp-stk*) $\not\leq$ 1)
 $\wedge$ (fetch (car (*temp-stk*), *data-seg*) = tag ('nat, LR-CONS-TAG)))
 $\rightarrow$ 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*)
 $\wedge$ lr-good-pointerps (*temp-stk*, *data-seg*)
 $\wedge$ (*fp-addr* = fetch (identity (LR-FP-ADDR), *data-seg*))
 $\wedge$ (type (*ref-count*) = 'nat))
 $\rightarrow$ 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*)))
 $\wedge$ 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*))
 $\wedge$ lr-proper-heapp (p-data-segment (*lr-eval*))

```

 $\wedge$  lr-good-pointerps (p-temp-stk (lr-eval), p-data-segment (lr-eval))
 $\wedge$  (p-psw (lr-eval) = 'run)
 $\wedge$  (p-psw (p-run-subr (car (lr-expr (l))),
    p-set-pc (lr->p (lr-eval), lr-return-pc (l))))
    = 'run)
 $\wedge$  (length (p-temp-stk (lr-eval))  $\not\leq$  arity (car (lr-expr (l))))
 $\wedge$  (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*)))

```

 $\wedge$  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-good-pointerps (p-temp-stk (new-l), p-data-segment (new-l))
 $\wedge$  (length (p-temp-stk (new-l))  $\not\leq$  arity (car (lr-expr (l))))
 $\wedge$  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))
 $\rightarrow$  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*)))

```

 $\wedge$  lr-programs-properp (l, table)
 $\wedge$  lr-programs-properp (new-l, table)
 $\wedge$  listp (lr-expr (l))
 $\wedge$  (car (lr-expr (l))  $\neq$  'if)
 $\wedge$  subrp (car (lr-expr (l)))

```

$$\begin{aligned}
& \wedge \text{proper-p-statep}(\text{lr-}\>\text{p}(\text{new-}l)) \\
& \wedge (\text{p-psw}(\text{new-}l) = \text{'run}) \\
& \wedge (\text{p-psw}(\text{p-run-subr}(\text{car}(\text{lr-expr}(l)), \\
& \qquad \qquad \qquad \text{p-set-pc}(\text{lr-}\>\text{p}(\text{new-}l), \text{lr-return-pc}(l)))) \\
& \qquad = \text{'run}) \\
& \wedge (\text{p-prog-segment}(l) = \text{p-prog-segment}(\text{new-}l)) \\
& \wedge (\text{area-name}(\text{p-pc}(l)) = \text{area-name}(\text{p-pc}(\text{new-}l))) \\
& \wedge \text{lr-proper-heapp}(\text{p-data-segment}(\text{new-}l)) \\
& \wedge \text{lr-good-pointerps}(\text{p-temp-stk}(\text{new-}l), \text{p-data-segment}(\text{new-}l)) \\
& \wedge (\text{length}(\text{p-temp-stk}(\text{new-}l)) \not\leq \text{arity}(\text{car}(\text{lr-expr}(l)))) \\
\rightarrow & \text{lr-proper-heapp}(\text{p-data-segment}(\text{p-run-subr}(\text{car}(\text{lr-expr}(l)), \\
& \qquad \qquad \qquad \text{p-set-pc}(\text{lr-}\>\text{p}(\text{new-}l), \\
& \qquad \qquad \qquad \text{lr-return-pc}(l))))))
\end{aligned}$$

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*) $\not\leq$ 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 ($list$) $\not\leq n$)
 $\rightarrow$ 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-addr

(lr-proper-heapp ($data-seg$)
 $\wedge$ lr-proper-p-areasp ($data-seg$)
 $\wedge$ all-undef-addr ($list$))
 $\rightarrow$ lr-good-pointerps ($list$, $data-seg$)

THEOREM: all-undef-addr-strip-cadrs-temp-vars-programs-properp-1

(lr-programs-properp-1 ($programs$, $program-names$, $table$)
 $\wedge$ definedp ($name$, $programs$))
 $\rightarrow$ all-undef-addr (strip-cadrs (temp-var-dcls (assoc ($name$, $programs$))))

THEOREM: all-undef-addr-strip-cadrs-temp-vars-programs-properp

(lr-programs-properp (l , $table$) $\wedge$ definedp ($name$, cdr (p-prog-segment (l))))
 $\rightarrow$ all-undef-addr (strip-cadrs (temp-var-dcls (assoc ($name$,
cdr (p-prog-segment (l))))))

THEOREM: lr-good-pointerps-popn
 $(\text{lr-good-pointerps}(\text{list}, \text{data-seg}) \wedge (\text{length}(\text{list}) \not\leq n))$
 $\rightarrow \text{lr-good-pointerps}(\text{popn}(n, \text{list}), \text{data-seg})$

THEOREM: lr-p-proper-statep-lr-funcall
 $((\text{p-psw}(\text{lr-eval}(\mathbf{t}, \text{lr-funcall}(l, \text{new-l}), c - 1)) = \mathbf{'run})$
 $\wedge \text{listp}(\text{lr-expr}(l))$
 $\wedge (\text{car}(\text{lr-expr}(l)) \neq \mathbf{'if})$
 $\wedge (\text{car}(\text{lr-expr}(l)) \neq \mathbf{'quote})$
 $\wedge \text{litatom}(\text{car}(\text{lr-expr}(l)))$
 $\wedge (\neg \text{subrp}(\text{car}(\text{lr-expr}(l))))$
 $\wedge \text{proper-p-statep}(\text{lr->p}(\text{new-l}))$
 $\wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{lr-p-proper-statep}(\text{p-temp-stk}(l),$
 $\quad \text{p-ctrl-stk}(l),$
 $\quad \text{p-data-segment}(l),$
 $\quad \text{table})$
 $\wedge \text{lr-p-proper-statep}(\text{p-temp-stk}(\text{new-l}),$
 $\quad \text{p-ctrl-stk}(\text{new-l}),$
 $\quad \text{p-data-segment}(\text{new-l}),$
 $\quad \text{table})$
 $\wedge (\text{length}(\text{p-temp-stk}(\text{new-l})) \not\leq \text{arity}(\text{car}(\text{lr-expr}(l))))$
 $\wedge \text{lr-proper-formalsp}(\text{cdr}(\text{p-prog-segment}(l)))$
 $\wedge \text{lr-programs-properp}(l, \text{table})$
 $\rightarrow \text{lr-p-proper-statep}(\text{p-temp-stk}(\text{lr-funcall}(l, \text{new-l})),$
 $\quad \text{p-ctrl-stk}(\text{lr-funcall}(l, \text{new-l})),$
 $\quad \text{p-data-segment}(\text{new-l}),$
 $\quad \text{table})$

THEOREM: length-p-temp-stk-lr-eval-flag-list-alt
 $(\text{proper-p-statep}(\text{lr->p}(l))$
 $\wedge \text{listp}(\text{lr-expr}(l))$
 $\wedge (\text{car}(\text{lr-expr}(l)) \neq \mathbf{'if})$
 $\wedge (\text{car}(\text{lr-expr}(l)) \neq \mathbf{'quote})$
 $\wedge (\neg \text{subrp}(\text{car}(\text{lr-expr}(l))))$
 $\wedge \text{litatom}(\text{car}(\text{lr-expr}(l)))$
 $\wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))$
 $\wedge \text{lr-programs-properp}(l, \text{table})$
 $\wedge \text{lr-proper-formalsp}(\text{cdr}(\text{p-prog-segment}(l)))$
 $\wedge (\text{p-psw}(\text{lr-eval}(\mathbf{'list}, \text{lr-set-pos}(l, \text{dv}(\text{offset}(\text{p-pc}(l)), 1)), c))$
 $\quad = \mathbf{'run}))$
 $\rightarrow (\text{length}(\text{p-temp-stk}(\text{lr-eval}(\mathbf{'list},$
 $\quad \text{lr-set-pos}(l, \text{dv}(\text{offset}(\text{p-pc}(l)), 1)),$
 $\quad c)))$
 $\quad = (\text{arity}(\text{car}(\text{lr-expr}(l))) + \text{length}(\text{p-temp-stk}(l)))$

THEOREM: lr-p-proper-statep-cdr-lr-ctrl-stk

```

(listp (lr-expr (l))
  ∧ (car (lr-expr (l)) ≠ 'if)
  ∧ (¬ subrp (car (lr-expr (l))))
  ∧ (car (lr-expr (l)) ≠ 'quote)
  ∧ litatom (car (lr-expr (l)))
  ∧ lr-p-proper-statep (temp-stk,
                        p-ctrl-stk (lr-eval (t,
                                              lr-funcall (l,
                                                            lr-eval ('list,
                                                                lr-set-pos (l,
                                                                    pos),
                                                                    c)),
                                                            c - 1)),
                        data-segment,
                        table)
  ∧ good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
  ∧ lr-programs-properp (l, table)
  ∧ proper-p-statep (lr->p (l))
  ∧ (p-psw (lr-eval (t,
                    lr-funcall (l, lr-eval ('list, lr-set-pos (l, pos), c)),
                    c - 1))
    = 'run)
  ∧ (p-psw (lr-eval ('list, lr-set-pos (l, pos), c)) = 'run)
  ∧ (pos = dv (offset (p-pc (l)), 1)))
→ lr-p-proper-statep (temp-stk,
                      cdr (p-ctrl-stk (lr-eval (t,
                                                  lr-funcall (l,
                                                                lr-eval ('list,
                                                                    lr-set-pos (l,
                                                                        pos),
                                                                        c)),
                                                                c - 1))),
                      data-segment,
                      table)

```

THEOREM: lr-eval-preserves-lr-p-proper-statep

```

(proper-p-statep (lr->p (l))
  ∧ good-posp (flag, offset (p-pc (l)), program-body (p-current-program (l)))
  ∧ (p-psw (lr-eval (flag, l, c)) = 'run)
  ∧ lr-p-proper-statep (p-temp-stk (l),
                        p-ctrl-stk (l),
                        p-data-segment (l),
                        table)

```

$\wedge$ lr-programs-properp (l , $table$)
 $\wedge$ 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))
 $\wedge$ (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)))
 $\wedge$ 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 ('d1,
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
 $(\text{listp}(\text{body}) \wedge (\text{car}(\text{body}) = \text{'if}) \wedge (n \neq 0))$
 $\rightarrow (\text{lr-p-pc-1}(\text{body}, \text{cons}(n, \text{pos})))$
 $= \text{if } n = 1 \text{ then lr-p-pc-1}(\text{cadr}(\text{body}), \text{pos})$
 $\text{elseif } n = 2$
 $\text{then } 3$
 $+ \text{lr-p-c-size}(\mathbf{t}, \text{cadr}(\text{body}))$
 $+ \text{lr-p-pc-1}(\text{caddr}(\text{body}), \text{pos})$
 $\text{else lr-p-c-size}(\mathbf{t}, \text{cadr}(\text{body}))$
 $+ \text{lr-p-c-size}(\mathbf{t}, \text{caddr}(\text{body}))$
 $+ \text{lr-p-pc-1}(\text{caddr}(\text{body}), \text{pos})$
 $+ \mathbf{4} \text{ endif})$

THEOREM: lr-p-pc-1-append-helper-2
 $(\text{listp}(\text{body}) \wedge (\text{car}(\text{body}) = \text{S-TEMP-EVAL}))$
 $\rightarrow (\text{lr-p-pc-1}(\text{body}, \text{cons}(\mathbf{1}, \text{pos})) = \text{lr-p-pc-1}(\text{cadr}(\text{body}), \text{pos}))$

THEOREM: lr-p-pc-1-append-helper-3
 $(\text{listp}(\text{body}) \wedge (\text{car}(\text{body}) = \text{S-TEMP-TEST}))$
 $\rightarrow (\text{lr-p-pc-1}(\text{body}, \text{cons}(\mathbf{1}, \text{pos})) = (\text{lr-p-pc-1}(\text{cadr}(\text{body}), \text{pos}) + \mathbf{4}))$

THEOREM: lr-p-pc-1-append-helper-4
 $(\text{listp}(\text{body})$
 $\wedge (\text{car}(\text{body}) \neq \text{S-TEMP-FETCH})$
 $\wedge (\text{car}(\text{body}) \neq \text{S-TEMP-EVAL})$
 $\wedge (\text{car}(\text{body}) \neq \text{S-TEMP-TEST})$
 $\wedge (\text{car}(\text{body}) \neq \text{'quote})$
 $\wedge (\text{car}(\text{body}) \neq \text{'if})$
 $\wedge (n \neq 0))$
 $\rightarrow (\text{lr-p-pc-1}(\text{body}, \text{cons}(n, \text{pos})))$
 $= (\text{lr-p-c-size-list}(n - 1, \text{body}) + \text{lr-p-pc-1}(\text{get}(n, \text{body}), \text{pos})))$

THEOREM: lr-p-pc-1-append
 $(\text{good-pospl}(\text{pos1}, \text{body})$
 $\wedge \text{lr-proper-exprp}(\mathbf{t}, \text{body}, \text{pnames}, \text{formals}, \text{temps}, \text{table}))$
 $\rightarrow (\text{lr-p-pc-1}(\text{body}, \text{append}(\text{pos1}, \text{pos2})))$
 $= (\text{lr-p-pc-1}(\text{body}, \text{pos1}) + \text{lr-p-pc-1}(\text{cur-expr}(\text{pos1}, \text{body}), \text{pos2})))$

THEOREM: append-butlast-list-car-last
 $\text{listp}(x) \rightarrow (\text{append}(\text{butlast}(x), \text{list}(\text{car}(\text{last}(x)))) = \text{plist}(x))$

THEOREM: listp-plist-car
 $\text{listp}(x) \rightarrow (\text{car}(\text{plist}(x)) = \text{car}(x))$

THEOREM: lr-p-pc-1-plist
 $\text{lr-p-pc-1}(\text{body}, \text{plist}(\text{pos})) = \text{lr-p-pc-1}(\text{body}, \text{pos})$

THEOREM: lr-p-pc-1-listp-offset

```
(listp (pos)
  ∧ good-posp1 (butlast (pos), body)
  ∧ lr-proper-exrp (t, body, pnames, formals, temps, table))
→ (lr-p-pc-1 (body, pos)
    = (lr-p-pc-1 (body, butlast (pos))
      + lr-p-pc-1 (cur-expr (butlast (pos), body),
                   list (car (last (pos))))))
```

THEOREM: lr-p-pc-1-nil

```
lr-p-pc-1 (body, nil) = 0
```

THEOREM: lr-p-pc-1-nx-helper

```
(listp (expr)
  ∧ (car (expr) ≠ S-TEMP-FETCH)
  ∧ (car (expr) ≠ S-TEMP-EVAL)
  ∧ (car (expr) ≠ S-TEMP-TEST)
  ∧ (car (expr) ≠ 'quote)
  ∧ (n ≠ 0)
  ∧ (n < length (expr)))
→ (lr-p-pc-1 (expr, list (n))
    = if car (expr) = 'if
      then case on n:
        case = 1
        then 0
        case = 2
        then 3 + lr-p-c-size (t, cadr (expr))
        otherwise lr-p-c-size (t, cadr (expr))
                  + lr-p-c-size (t, caddr (expr))
                  + 4 endcase
      else lr-p-c-size-list (n - 1, expr) endif)
```

THEOREM: lr-p-pc-1-nx

```
(lr-programs-properp (l, table)
  ∧ good-posp ('list, offset (p-pc (l)), program-body (p-current-program (l)))
  ∧ listp (offset (p-pc (l)))
  ∧ listp (lr-expr-list (l))
  ∧ (car (cur-expr (butlast (offset (p-pc (l))),
                    program-body (p-current-program (l))))
    ≠ 'if))
→ (lr-p-pc-1 (program-body (p-current-program (l)), nx (offset (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))))
```

EVENT: Disable lr-p-pc-1-listp-offset.

THEOREM: lr-p-pc-1-dv-1-car-lr-expr-if

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

THEOREM: lr-p-pc-1-dv-2-car-lr-expr-if

$$\begin{aligned}
& (\text{listp}(\text{lr-expr}(l)) \\
& \quad \wedge (\text{car}(\text{lr-expr}(l)) = \text{'if}) \\
& \quad \wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l))) \\
& \quad \wedge \text{lr-programs-properp}(l, \text{table})) \\
& \rightarrow (\text{lr-p-pc-1}(\text{program-body}(\text{p-current-program}(l)), \text{dv}(\text{offset}(\text{p-pc}(l)), 2)) \\
& \quad = (3 \\
& \quad \quad + \text{lr-p-c-size}(\mathbf{t}, \text{cadr}(\text{lr-expr}(l))) \\
& \quad \quad + \text{lr-p-pc-1}(\text{program-body}(\text{p-current-program}(l)), \\
& \quad \quad \quad \text{offset}(\text{p-pc}(l))))
\end{aligned}$$

THEOREM: lr-p-pc-1-dv-3-car-lr-expr-if

$$\begin{aligned}
& (\text{listp}(\text{lr-expr}(l)) \\
& \quad \wedge (\text{car}(\text{lr-expr}(l)) = \text{'if}) \\
& \quad \wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l))) \\
& \quad \wedge \text{lr-programs-properp}(l, \text{table})) \\
& \rightarrow (\text{lr-p-pc-1}(\text{program-body}(\text{p-current-program}(l)), \text{dv}(\text{offset}(\text{p-pc}(l)), 3)) \\
& \quad = (\text{lr-p-c-size}(\mathbf{t}, \text{cadr}(\text{lr-expr}(l))) \\
& \quad \quad + 3 \\
& \quad \quad + \text{lr-p-c-size}(\mathbf{t}, \text{caddr}(\text{lr-expr}(l))) \\
& \quad \quad + 1 \\
& \quad \quad + \text{lr-p-pc-1}(\text{program-body}(\text{p-current-program}(l)), \\
& \quad \quad \quad \text{offset}(\text{p-pc}(l))))
\end{aligned}$$

THEOREM: lr-p-c-size-not-1-car-if

$$(\text{listp}(\text{expr}) \wedge (\text{car}(\text{expr}) = \text{'if})) \rightarrow (\text{lr-p-c-size}(\mathbf{t}, \text{expr}) \neq 1)$$

THEOREM: lr-p-c-size-ge-plus-2-size-cadr-car-if

$$\begin{aligned}
& (\text{listp}(\text{expr}) \wedge (\text{car}(\text{expr}) = \text{'if})) \\
& \rightarrow ((1 + (1 + \text{lr-p-c-size}(\mathbf{t}, \text{cadr}(\text{expr})))) < \text{lr-p-c-size}(\mathbf{t}, \text{expr}))
\end{aligned}$$

THEOREM: get-comp-if-helper-helper

$$\begin{aligned}
& ((n \not\leq (x + y + 3)) \wedge (n \neq (x + y + 3))) \\
& \rightarrow (\text{get}(n - (x + 3 + y), \text{cons}(w, l3)) \\
& \quad = \text{get}(n - (x + y + 4), l3))
\end{aligned}$$

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

$$\begin{aligned}
& (\text{proper-p-statep } (p) \wedge (\text{area-name } (\text{p-pc } (p)) = \text{area-name } (pc))) \\
\rightarrow & (\text{proper-p-statep } (\text{p-set-pc } (p, pc)) = \text{p-objectp-type } ('pc, pc, p))
\end{aligned}$$

$$\begin{aligned}
& \text{THEOREM: proper-p-statep-p-set-pc-equal-p-set-pc} \\
& (\text{good-posp1 } (\text{offset } (\text{p-pc } (l)), \text{program-body } (\text{p-current-program } (l))) \\
& \wedge \text{lr-programs-properp } (l, \text{table}) \\
& \wedge (\text{p } (\text{lr->p } (l), \text{p-clock1 } (\mathbf{t}, l, c)) \\
& \quad = \text{p-set-pc } (\text{lr->p } (\text{lr-eval } (\mathbf{t}, l, c)), \\
& \quad \quad \text{tag } ('pc, \\
& \quad \quad \quad \text{cons } (\text{area-name } (\text{p-pc } (l)), \\
& \quad \quad \quad \quad \text{lr-p-pc-1 } (\text{program-body } (\text{p-current-program } (l)), \\
& \quad \quad \quad \quad \quad \text{offset } (\text{p-pc } (l))) \\
& \quad \quad \quad \quad + \text{lr-p-c-size } (\mathbf{t}, \text{lr-expr } (l)))))) \\
& \wedge \text{proper-p-statep } (\text{lr->p } (\text{lr-eval } (\mathbf{t}, l, c))) \\
\rightarrow & \text{proper-p-statep } (\text{p } (\text{lr->p } (l), \text{p-clock1 } (\mathbf{t}, l, c)))
\end{aligned}$$

$$\begin{aligned}
& \text{THEOREM: lr-eval-p-pc-equivalence-helper-1} \\
& \text{p } (\text{lr->p } (l1), \text{p-clock1 } (flag1, l2, c1) + \text{p-clock1 } (flag2, l3, c2)) \\
= & \text{p } (\text{p } (\text{lr->p } (l1), \text{p-clock1 } (flag1, l2, c1)), \text{p-clock1 } (flag2, l3, c2))
\end{aligned}$$

$$\begin{aligned}
& \text{THEOREM: lr-eval-p-pc-equivalence-helper-1-5} \\
& (\text{listp } (\text{offset } (\text{p-pc } (l))) \wedge \text{listp } (\text{lr-expr-list } (l))) \\
\rightarrow & (\text{p-final-pc } ('list, \\
& \quad \text{lr-set-expr } (\text{lr-eval } (\mathbf{t}, l, c), l, \text{nx } (\text{offset } (\text{p-pc } (l)))), \\
& \quad 0) \\
= & \text{tag } ('pc, \\
& \quad \text{cons } (\text{area-name } (\text{p-pc } (l)), \\
& \quad \quad \text{lr-p-pc-1 } (\text{program-body } (\text{p-current-program } (l)), \\
& \quad \quad \quad \text{nx } (\text{offset } (\text{p-pc } (l)))) \\
& \quad + \text{lr-p-c-size-list } (\text{length } (\text{lr-expr-list } (l)) - 1, \\
& \quad \quad \text{lr-expr-list } (l))))
\end{aligned}$$

$$\begin{aligned}
& \text{THEOREM: lr-eval-p-pc-equivalence-helper-2} \\
& ((\text{p-psw } (l) = 'run) \\
& \wedge \text{listp } (\text{offset } (\text{p-pc } (l))) \\
& \wedge (\neg \text{listp } (\text{lr-expr-list } (l)))) \\
\rightarrow & (\text{p-set-pc } (\text{lr->p } (l), \\
& \quad \text{tag } ('pc, \\
& \quad \quad \text{cons } (\text{area-name } (\text{p-pc } (l)), \\
& \quad \quad \quad \text{lr-p-pc-1 } (\text{program-body } (\text{p-current-program } (l)), \\
& \quad \quad \quad \quad \text{offset } (\text{p-pc } (l)))))) \\
= & \text{p } (\text{lr->p } (l), 0)
\end{aligned}$$

$$\begin{aligned}
& \text{THEOREM: lr-eval-p-pc-equivalence-helper-3} \\
& ((\text{p-psw } (l) = 'run)
\end{aligned}$$

$$\begin{aligned}
& \wedge (flag \neq 'list) \\
& \wedge litatom(lr\text{-}expr(l)) \\
& \wedge good\text{-}posp1(offset(p\text{-}pc(l), program\text{-}body(p\text{-}current\text{-}program(l))) \\
& \wedge lr\text{-}programs\text{-}properp(l, table) \\
& \wedge (length(p\text{-}temp\text{-}stk(l)) < p\text{-}max\text{-}temp\text{-}stk\text{-}size(l)) \\
\rightarrow & (p\text{-}set\text{-}pc(lr\text{-}>p(lr\text{-}set\text{-}tstk(l, \\
& \qquad \qquad \qquad cons(cdr(assoc(lr\text{-}expr(l), \\
& \qquad \qquad \qquad \qquad \qquad bindings(car(p\text{-}ctrl\text{-}stk(l))))), \\
& \qquad \qquad \qquad p\text{-}temp\text{-}stk(l))), \\
& \qquad \qquad \qquad p\text{-}final\text{-}pc(flag, l, 0)) \\
& = p(lr\text{-}>p(l, 1))
\end{aligned}$$

THEOREM: $lr\text{-}>p\text{-}lr\text{-}set\text{-}pos\text{-}dv\text{-}1\text{-}car\text{-}lr\text{-}expr\text{-}if$

$$\begin{aligned}
& ((car(lr\text{-}expr(l)) = 'if) \\
& \wedge listp(lr\text{-}expr(l)) \\
& \wedge good\text{-}posp1(offset(p\text{-}pc(l), program\text{-}body(p\text{-}current\text{-}program(l))) \\
& \wedge lr\text{-}programs\text{-}properp(l, table)) \\
\rightarrow & (lr\text{-}>p(lr\text{-}set\text{-}pos(l, dv(offset(p\text{-}pc(l), 1))) = lr\text{-}>p(l))
\end{aligned}$$

THEOREM: $lr\text{-}p\text{-}proper\text{-}statep\text{-}listp\text{-}p\text{-}temp\text{-}stk\text{-}type\text{-}car\text{-}addr$

$$\begin{aligned}
& (lr\text{-}p\text{-}proper\text{-}statep(temp\text{-}stk, ctrl\text{-}stk, data\text{-}seg, table) \wedge listp(temp\text{-}stk)) \\
\rightarrow & (type(car(temp\text{-}stk)) = 'addr)
\end{aligned}$$

THEOREM: $proper\text{-}p\text{-}statep\text{-}lessp\text{-}length\text{-}p\text{-}temp\text{-}stk\text{-}max\text{-}temp\text{-}stk\text{-}size$

$$\begin{aligned}
& proper\text{-}p\text{-}statep(lr\text{-}>p(l)) \\
\rightarrow & (p\text{-}max\text{-}temp\text{-}stk\text{-}size(l) \not< length(p\text{-}temp\text{-}stk(l)))
\end{aligned}$$

THEOREM: $length\text{-}p\text{-}temp\text{-}stk\text{-}lr\text{-}eval\text{-}lr\text{-}set\text{-}pos$

$$\begin{aligned}
& (proper\text{-}p\text{-}statep(lr\text{-}>p(l)) \\
& \wedge good\text{-}posp1(pos, program\text{-}body(p\text{-}current\text{-}program(l))) \\
& \wedge lr\text{-}programs\text{-}properp(l, table) \\
& \wedge lr\text{-}proper\text{-}formalsp(cdr(p\text{-}prog\text{-}segment(l))) \\
& \wedge (p\text{-}psw(lr\text{-}eval(t, lr\text{-}set\text{-}pos(l, pos), c)) = 'run)) \\
\rightarrow & (length(p\text{-}temp\text{-}stk(lr\text{-}eval(t, lr\text{-}set\text{-}pos(l, pos), c))) \\
& \quad = (1 + length(p\text{-}temp\text{-}stk(l)))
\end{aligned}$$

THEOREM: $not\text{-}lessp\text{-}length\text{-}proper\text{-}p\text{-}statep\text{-}lr\text{-}eval\text{-}lr\text{-}set\text{-}pos$

$$\begin{aligned}
& (good\text{-}posp1(pos, program\text{-}body(p\text{-}current\text{-}program(l))) \\
& \wedge lr\text{-}programs\text{-}properp(l, table) \\
& \wedge proper\text{-}p\text{-}statep(lr\text{-}>p(l)) \\
& \wedge (length(p\text{-}temp\text{-}stk(l)) \not< (p\text{-}max\text{-}temp\text{-}stk\text{-}size(l) - 1)) \\
& \wedge lr\text{-}proper\text{-}formalsp(cdr(p\text{-}prog\text{-}segment(l))) \\
\rightarrow & (p\text{-}psw(lr\text{-}if\text{-}ok(lr\text{-}eval(t, lr\text{-}set\text{-}pos(l, pos), c))) \neq 'run)
\end{aligned}$$

THEOREM: $lr\text{-}pop\text{-}tstk\text{-}lr\text{-}if\text{-}ok$

$$\begin{aligned}
& (p\text{-}psw(lr\text{-}if\text{-}ok(l)) = 'run) \\
\rightarrow & (lr\text{-}pop\text{-}tstk(lr\text{-}if\text{-}ok(l)) = lr\text{-}pop\text{-}tstk(l))
\end{aligned}$$

THEOREM: add-addr-p-final-pc

$$\begin{aligned} & (\text{add-addr}(\text{p-final-pc}(\text{flag}, l, n), 1 + m) \\ & = \text{add-addr}(\text{p-final-pc}(\text{flag}, l, 1 + n), m)) \\ \wedge & (\text{add-addr}(\text{p-final-pc}(\text{flag}, l, n), 0) = \text{p-final-pc}(\text{flag}, l, n)) \end{aligned}$$

THEOREM: lessp-3-lr-p-c-size-car-if

$$(\text{listp}(\text{expr}) \wedge (\text{car}(\text{expr}) = \text{'if})) \rightarrow (3 < \text{lr-p-c-size}(\mathbf{t}, \text{expr}))$$

THEOREM: comp-body-1-car-expr-if

$$\begin{aligned} & ((\text{car}(\text{expr}) = \text{'if}) \wedge \text{listp}(\text{expr})) \\ \rightarrow & (\text{comp-body-1}(\mathbf{t}, \text{expr}, n) \\ & = \text{comp-if}(\text{comp-body-1}(\mathbf{t}, \text{cadr}(\text{expr}), n), \\ & \quad \text{comp-body-1}(\mathbf{t}, \\ & \quad \quad \text{caddr}(\text{expr}), \\ & \quad \quad n + 3 + \text{lr-p-c-size}(\mathbf{t}, \text{cadr}(\text{expr}))), \\ & \quad \text{comp-body-1}(\mathbf{t}, \\ & \quad \quad \text{caddr}(\text{expr}), \\ & \quad \quad n \\ & \quad \quad + 4 \\ & \quad \quad + \text{lr-p-c-size}(\mathbf{t}, \text{cadr}(\text{expr})) \\ & \quad \quad + \text{lr-p-c-size}(\mathbf{t}, \text{caddr}(\text{expr}))), \\ & \quad n)) \end{aligned}$$

THEOREM: get-lr-p-c-size-lessp-restn-lr-p-pc-1-comp-body-1

$$\begin{aligned} & (\text{good-posp1}(\text{pos}, \text{program-body}(\text{p-current-program}(l))) \\ & \wedge \text{lr-programs-properp}(l, \text{table}) \\ & \wedge \text{listp}(\text{lr-expr}(l)) \\ & \wedge (\text{car}(\text{lr-expr}(l)) = \text{'if}) \\ & \wedge (m < 3) \\ & \wedge (\text{name} = \text{area-name}(\text{p-pc}(l))) \\ & \wedge (\text{pos} = \text{offset}(\text{p-pc}(l))) \\ \rightarrow & (\text{unlabel}(\text{get}(\text{offset}(\text{p-final-pc}(\mathbf{t}, \text{lr-set-pos}(l, \text{dv}(\text{pos}, 1)), m)), \\ & \quad \text{program-body}(\text{assoc}(\text{name}, \text{comp-programs}(\text{p-prog-segment}(l))))) \\ & = \text{get}(m, \\ & \quad \text{list}(\text{list}(\text{'push-constant}, \text{LR-F-ADDR}), \\ & \quad \quad \text{'(eq)}, \\ & \quad \quad \text{list}(\text{'test-bool-and-jump}, \\ & \quad \quad \quad \mathbf{t}, \\ & \quad \quad \quad \text{lr-make-label}(\text{lr-p-pc-1}(\text{program-body}(\text{p-current-program}(l)), \\ & \quad \quad \quad \quad \text{pos}) \\ & \quad \quad \quad + 4 \\ & \quad \quad \quad + \text{lr-p-c-size}(\mathbf{t}, \\ & \quad \quad \quad \quad \text{cadr}(\text{lr-expr}(l))) \\ & \quad \quad \quad + \text{lr-p-c-size}(\mathbf{t}, \\ & \quad \quad \quad \quad \text{caddr}(\text{lr-expr}(l))))) \end{aligned}$$

THEOREM: area-name-p-final-pc
 $\text{area-name (p-final-pc (flag, l, n))} = \text{area-name (p-pc (l))}$

THEOREM: lr-eval-p-pc-equivalence-helper-4-helper-1
let *test* **be** $\text{lr-eval (t, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c)}$
in
 $((\text{p-psw (l)} = \text{'run})$
 $\wedge (c \neq 0)$
 $\wedge \text{listp (lr-expr (l))}$
 $\wedge (\text{car (lr-expr (l))} = \text{'if})$
 $\wedge (\text{p-psw (lr-if-ok (test))} = \text{'run})$
 $\wedge (\text{car (p-temp-stk (test))} \neq \text{LR-F-ADDR})$
 $\wedge \text{proper-p-statep (lr->p (l))}$
 $\wedge \text{good-posp1 (offset (p-pc (l)), \text{program-body (p-current-program (l))})}$
 $\wedge \text{lr-programs-properp (l, table)}$
 $\wedge \text{lr-p-proper-statep (p-temp-stk (l),}$
 p-ctrl-stk (l),
 $\text{p-data-segment (l),}$
 $\text{table})}$
 $\wedge \text{lr-proper-formalsp (cdr (p-prog-segment (l)))}$
 $\wedge (\text{p-psw (lr-eval (t,}$
 $\text{lr-set-expr (lr-pop-tstk (test),}$
 l,
 $\text{dv (offset (p-pc (l)), 2)),}$
 c))
 $= \text{'run}))$
 $\rightarrow (\text{p (p-set-pc (lr->p (test),}$
 p-final-pc (t,
 $\text{lr-set-pos (l, dv (offset (p-pc (l)), 1)),}$
 0)),
 3)
 $= \text{p-set-pc (lr->p (lr-pop-tstk (test)),}$
 tag ('pc,
 $\text{cons (area-name (p-pc (l)),}$
 3
 + lr-p-c-size (t,
 $\text{cadr (lr-expr (l))}$
 $\text{+ lr-p-pc-1 (program-body (p-current-program (l)),}$
 $\text{offset (p-pc (l)))))) \text{endlet}$

THEOREM: lessp-plus-lr-p-c-size-cadr-caddr-3-car-if
 $(\text{listp (expr)} \wedge (\text{car (expr)} = \text{'if}))$
 $\rightarrow ((\text{lr-p-c-size (t, cadr (expr))} + \text{lr-p-c-size (t, caddr (expr))})$
 $< (((\text{lr-p-c-size (t, expr)} - 1) - 1) - 1))$

THEOREM: get-plus-lr-p-c-size-cadr-caddr-4-comp-body-cur-expr

$$\begin{aligned}
 & ((size1 = \text{length}(test)) \wedge (size2 = \text{length}(then))) \\
 \rightarrow & (\text{get}(3 + size1 + size2, \text{comp-if}(test, then, else, n)) \\
 & = \text{list}('jump, \\
 & \quad \text{lr-make-label}(4 \\
 & \quad \quad + n \\
 & \quad \quad + \text{length}(test) \\
 & \quad \quad + \text{length}(then) \\
 & \quad \quad + \text{length}(else))))
 \end{aligned}$$

THEOREM: cur-expr-add1-opener

$$\text{cur-expr}(\text{cons}(1 + n, pos), body) = \text{cur-expr}(pos, \text{get}(n, \text{cdr}(body)))$$

THEOREM: lr-p-pc-1-car-expr-if-2

$$\begin{aligned}
 & (\text{listp}(expr) \wedge (\text{car}(expr) = 'if)) \\
 \rightarrow & (\text{lr-p-pc-1}(expr, '(2)) = (3 + \text{lr-p-c-size}(\mathbf{t}, \text{cadr}(expr))))
 \end{aligned}$$

THEOREM: get-plus-lr-p-pc-1-lr-pc-size-cadr-assoc-comp-body-if-4

$$\begin{aligned}
 & (\text{good-posp1}(\text{offset}(\text{p-pc}(l1)), \text{program-body}(\text{p-current-program}(l1))) \\
 & \wedge \text{lr-programs-properp}(l1, table) \\
 & \wedge \text{listp}(\text{lr-expr}(l1)) \\
 & \wedge (\text{car}(\text{lr-expr}(l1)) = 'if) \\
 & \wedge (pos = \text{dv}(\text{offset}(\text{p-pc}(l1)), 2)) \\
 & \wedge (\text{area-name}(\text{p-pc}(l2)) = \text{area-name}(\text{p-pc}(l1))) \\
 & \wedge (\text{p-prog-segment}(l2) = \text{p-prog-segment}(l1))) \\
 \rightarrow & (\text{get}(\text{offset}(\text{p-final-pc}(\mathbf{t}, \text{lr-set-expr}(l2, l1, pos), 0)), \\
 & \quad \text{program-body}(\text{assoc}(\text{area-name}(\text{p-pc}(l1)), \\
 & \quad \quad \text{comp-programs}(\text{p-prog-segment}(l1))))) \\
 & = \text{list}('d1, \\
 & \quad \text{lr-make-label}(\text{offset}(\text{p-final-pc}(\mathbf{t}, \\
 & \quad \quad \text{lr-set-expr}(l2, l1, pos), \\
 & \quad \quad \quad 0))), \\
 & \quad \mathbf{nil}, \\
 & \quad \text{list}('jump, \\
 & \quad \quad \text{lr-make-label}(4 \\
 & \quad \quad \quad + \text{lr-p-pc-1}(\text{program-body}(\text{p-current-program}(l1)), \\
 & \quad \quad \quad \quad \text{offset}(\text{p-pc}(l1))) \\
 & \quad \quad \quad + \text{lr-p-c-size}(\mathbf{t}, \\
 & \quad \quad \quad \quad \text{cadr}(\text{lr-expr}(l1))) \\
 & \quad \quad \quad + \text{lr-p-c-size}(\mathbf{t}, \\
 & \quad \quad \quad \quad \text{caddr}(\text{lr-expr}(l1))) \\
 & \quad \quad \quad + \text{lr-p-c-size}(\mathbf{t}, \\
 & \quad \quad \quad \quad \text{caddr}(\text{lr-expr}(l1))))))
 \end{aligned}$$

THEOREM: find-label-lr-make-label-label-instrs

$$((m \not\leq n) \wedge (m \in \mathbf{N}) \wedge (m < (n + \text{length}(instrs))))$$

$$\rightarrow (\text{find-label}(\text{lr-make-label}(m), \text{label-instrs}(instrs, n)) = (m - n))$$

THEOREM: find-label-past-else-lr-expr-car-if

```

(listp (lr-expr (l))
  ∧ (car (lr-expr (l)) = 'if)
  ∧ (expr = lr-expr (l))
  ∧ (body = program-body (p-current-program (l)))
  ∧ (pos = offset (p-pc (l)))
  ∧ lr-programs-properp (l, table)
  ∧ good-posp1 (pos, body)
→ (find-label (lr-make-label (1 + (1 + (1 + (1 + (lr-p-c-size (t,
                                                                    cadr (expr))
                                                                    + lr-p-c-size (t,
                                                                    caddr (expr))
                                                                    + lr-p-c-size (t,
                                                                    caddr (expr))
                                                                    + lr-p-pc-1 (body,
                                                                    pos)))))),
                program-body (assoc (area-name (p-pc (l)),
                                     comp-programs (p-prog-segment (l))))))
= (1 + (1 + (1 + (1 + (lr-p-c-size (t, cadr (expr))
                                   + lr-p-c-size (t, caddr (expr))
                                   + lr-p-c-size (t, caddr (expr))
                                   + lr-p-pc-1 (body, pos)))))))

```

THEOREM: lr-eval-p-pc-equivalence-helper-4-helper-2

```

let test be lr-eval (t, lr-set-pos (l, dv (offset (p-pc (l)), 1)), c)
in
let then be lr-set-expr (lr-pop-tstk (test), 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)
  ∧ 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, 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),

```

```

                                table)
^  lr-proper-formalsp (cdr (p-prog-segment (l)))
^  (p-psw (lr-eval (t, then, c)) = 'run)
^  (pos = dv (offset (p-pc (l)), 1)))
→  (p (lr->p (l),
        p-clock1 (t, lr-set-pos (l, pos), c)
        + 3
        + p-clock1 (t, then, c)
        + 1)
    =  p-set-pc (lr->p (lr-eval (t, then, c)),
        p-final-pc (flag, l, 0))) endlet

```

THEOREM: lessp-plus-lr-p-pc-1-lr-p-c-size-3-1-lr-expr-car-if

```

((car (lr-expr (l)) = 'if)
^  listp (lr-expr (l))
^  good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))
^  lr-programs-properp (l, table))
→  ((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))))
    < length (program-body (assoc (area-name (p-pc (l)),
                                comp-programs (p-prog-segment (l))))))

```

THEOREM: find-label-else-start-lr-expr-car-if

```

(listp (lr-expr (l))
^  (car (lr-expr (l)) = 'if)
^  (expr = lr-expr (l))
^  (body = program-body (p-current-program (l)))
^  (pos = offset (p-pc (l)))
^  lr-programs-properp (l, table)
^  good-posp1 (pos, body))
→  (find-label (lr-make-label (1 + (1 + (1 + (1 + (lr-p-c-size (t,
                                                                cadr (expr))
                                                                + lr-p-c-size (t,
                                                                caddr (expr))
                                                                + lr-p-pc-1 (body,
                                                                pos)))))),
                    program-body (assoc (area-name (p-pc (l)),
                    comp-programs (p-prog-segment (l))))))
    =  (1 + (1 + (1 + (1 + (lr-p-c-size (t, cadr (expr))
    + lr-p-c-size (t, caddr (expr))
    + lr-p-pc-1 (body, pos))))))

```

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 ≠ 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))

```


→ (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)
 ∧ (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 (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)))
 ∧ (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)))
 ∧ 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, else, c)) = 'run)
 ∧ (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 ≠ 0)
  ^ (c ∈ 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 ≠ 0)
  ^ (c ∈ 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, cadr(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, cadr(expr)),
                     list('jump,
                           lr-make-label(n
                                           + 7
                                           + lr-p-c-size(t,
                                                           cadr(expr))))),
               list('push-local, cadr(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, cadr(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*) $\wedge$ (car(*expr*) = S-TEMP-TEST))
 $\rightarrow$ (car(comp-body-1(**t**, *expr*, *n*)) = list('push-local, cadr(*expr*)))

THEOREM: get-+-lr-p-pc-1-n-2-size-temp-test-assoc-comp-programs
 (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*)
 $\wedge$ (*n* $\not\leq$ 4)
 $\wedge$ (6 $\not\leq$ *n*)
 $\wedge$ (*progs* = p-prog-segment(*l*))
 $\wedge$ (*body* = program-body(p-current-program(*l*)))
 $\wedge$ (*pos* = offset(p-pc(*l*)))
 $\rightarrow$ (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, cadr(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, cadr(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*)
 $\wedge$ ((car(lr-expr(*l*)) = S-TEMP-FETCH)
 $\vee$ (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-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 , $body$)
 $\wedge$ ($7 \not\leq n$)
 $\rightarrow$ (find-label (lr-make-label (lr-p-pc-1 ($body$, pos)
 $\quad + n$
 $\quad +$ 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) $\not\leq$ (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)

$$\begin{aligned}
& \wedge (\text{p-psw}(\text{lr-do-temp-fetch}(l)) = \text{'run})) \\
\rightarrow & (\text{p-set-pc}(\text{lr->p}(\text{lr-do-temp-fetch}(l)), \text{p-final-pc}(flag, l, 0)) \\
& = \text{p}(\text{lr->p}(l), 5))
\end{aligned}$$

THEOREM: lr-p-pc-dv-1-s-temp-test

$$\begin{aligned}
& (\text{listp}(\text{lr-expr}(l)) \\
& \wedge (\text{car}(\text{lr-expr}(l)) = \text{S-TEMP-TEST}) \\
& \wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l))) \\
& \wedge \text{lr-programs-properp}(l, table)) \\
\rightarrow & (\text{lr-p-pc-1}(\text{program-body}(\text{p-current-program}(l)), \text{dv}(\text{offset}(\text{p-pc}(l)), 1)) \\
& = (\text{lr-p-pc-1}(\text{program-body}(\text{p-current-program}(l)), \text{offset}(\text{p-pc}(l)) \\
& + 4))
\end{aligned}$$

THEOREM: lr-eval-p-pc-equivalence-helper-8-helper-1

$$\begin{aligned}
& ((\text{p-psw}(l) = \text{'run}) \\
& \wedge \text{listp}(\text{lr-expr}(l)) \\
& \wedge (\text{car}(\text{lr-expr}(l)) = \text{S-TEMP-TEST}) \\
& \wedge (\text{p-max-temp-stk-size}(l) \not\leq (2 + \text{length}(\text{p-temp-stk}(l)))) \\
& \wedge (\text{local-var-value}(\text{caddr}(\text{lr-expr}(l)), \text{p-ctrl-stk}(l)) = \text{LR-UNDEF-ADDR}) \\
& \wedge \text{proper-p-statep}(\text{lr->p}(l)) \\
& \wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l))) \\
& \wedge \text{lr-programs-properp}(l, table)) \\
\rightarrow & (\text{lr->p}(\text{lr-set-pos}(l, \text{dv}(\text{offset}(\text{p-pc}(l)), 1))) = \text{p}(\text{lr->p}(l), 4))
\end{aligned}$$

THEOREM: lr-eval-p-pc-equivalence-helper-8-helper-2-helper

$$(4 + \text{lr-p-c-size}(t, expr) + 1) = (5 + \text{lr-p-c-size}(t, expr))$$

THEOREM: lr-eval-p-pc-equivalence-helper-8-helper-2

let *lr-eval* **be** $\text{lr-eval}(t, \text{lr-set-pos}(l, \text{dv}(\text{offset}(\text{p-pc}(l)), 1)), c)$

in

$$\begin{aligned}
& ((flag \neq \text{'list}) \\
& \wedge \text{listp}(\text{lr-expr}(l)) \\
& \wedge (\text{car}(\text{lr-expr}(l)) = \text{S-TEMP-TEST}) \\
& \wedge (\text{p-max-temp-stk-size}(l) \neq 0) \\
& \wedge (\text{p-max-temp-stk-size}(l) \in \mathbf{N}) \\
& \wedge (\text{p-max-temp-stk-size}(l) \neq 1) \\
& \wedge (((\text{p-max-temp-stk-size}(l) - 1) - 1) \not\leq \text{length}(\text{p-temp-stk}(l))) \\
& \wedge \text{proper-p-statep}(\text{lr->p}(l)) \\
& \wedge \text{good-posp1}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l))) \\
& \wedge \text{lr-programs-properp}(l, table) \\
& \wedge (\text{p-psw}(lr-eval) = \text{'run})) \\
\rightarrow & (\text{p-set-pc}(\text{lr->p}(\text{lr-set-temp}(lr-eval, \\
& \qquad \qquad \qquad \text{car}(\text{p-temp-stk}(lr-eval)), \\
& \qquad \qquad \qquad \text{caddr}(\text{lr-expr}(l))), \\
& \qquad \qquad \text{p-final-pc}(flag, l, 0))
\end{aligned}$$

```

=  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) < (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 , $table$)
 $\wedge$ ($prog = \text{assoc}(\text{area-name}(\text{p-pc}(l)), \text{p-prog-segment}(l))$)
 $\rightarrow$ (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

$((\text{p-psw}(l) = \text{'run})$
 $\wedge$ ($flag \neq \text{'list}$)
 $\wedge$ 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

$((\text{p-psw}(l) = \text{'run})$
 $\wedge$ ($flag \neq \text{'list}$)
 $\wedge$ 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

$(\text{car}(\text{cur-expr}(\text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l)))) = \text{'if})$
 $= (\text{car}(\text{lr-expr}(l)) = \text{'if})$

THEOREM: lr-p-pc-lr-set-pos-dv-1-car-lr-expr-funcall

$(\text{listp}(\text{lr-expr}(l))$
 $\wedge$ (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-pospl (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-pospl (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*))),

$$\begin{aligned}
& \text{lr-expr}(l))))) \\
= & \text{p}(\text{p-set-pc}(\text{lr->p}(new-l), \\
& \text{add-addr}(\text{lr-p-pc}(l), \\
& \text{lr-p-c-size-list}(\text{arity}(\text{car}(\text{lr-expr}(l))), \\
& \text{lr-expr}(l))), \\
& \text{p-run-subr-clock}(l, new-l)))
\end{aligned}$$

THEOREM: p-pc-run-car

$$\begin{aligned}
& (\text{proper-p-statep}(\text{lr->p}(l)) \\
& \wedge (\text{p-psw}(l) = \text{'run}) \\
& \wedge (\text{p-psw}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \text{p-car-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
& \quad = \text{'run}) \\
& \wedge \text{lr-programs-properp}(l, table) \\
& \wedge (\text{unlabel}(\text{get}(\text{offset}(pc), \\
& \quad \text{program-body}(\text{assoc}(\text{area-name}(pc), \\
& \quad \quad \text{p-prog-segment}(\text{lr->p}(l))))) \\
& \quad = \text{'(call car)})) \\
\rightarrow & (\text{p-pc}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \text{p-car-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
& \quad = \text{add-addr}(pc, 1))
\end{aligned}$$

THEOREM: p-pc-run-cdr

$$\begin{aligned}
& (\text{proper-p-statep}(\text{lr->p}(l)) \\
& \wedge (\text{p-psw}(l) = \text{'run}) \\
& \wedge (\text{p-psw}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \text{p-cdr-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
& \quad = \text{'run}) \\
& \wedge \text{lr-programs-properp}(l, table) \\
& \wedge (\text{unlabel}(\text{get}(\text{offset}(pc), \\
& \quad \text{program-body}(\text{assoc}(\text{area-name}(pc), \\
& \quad \quad \text{p-prog-segment}(\text{lr->p}(l))))) \\
& \quad = \text{'(call cdr)})) \\
\rightarrow & (\text{p-pc}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \text{p-cdr-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
& \quad = \text{add-addr}(pc, 1))
\end{aligned}$$

THEOREM: p-pc-run-cons

$$\begin{aligned}
& (\text{proper-p-statep}(\text{lr->p}(l)) \\
& \wedge (\text{p-psw}(l) = \text{'run}) \\
& \wedge (\text{p-psw}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \text{p-cons-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
& \quad = \text{'run}) \\
& \wedge \text{lr-programs-properp}(l, table) \\
& \wedge (\text{unlabel}(\text{get}(\text{offset}(pc), \\
& \quad \text{program-body}(\text{assoc}(\text{area-name}(pc), \\
& \quad \quad \text{p-prog-segment}(\text{lr->p}(l))))) \\
& \quad = \text{'(call cons)})) \\
\rightarrow & (\text{p-pc}(\text{p}(\text{p-set-pc}(\text{lr->p}(l), pc), \text{p-cons-clock}(\text{p-set-pc}(\text{lr->p}(l), pc)))) \\
& \quad = \text{add-addr}(pc, 1))
\end{aligned}$$

THEOREM: p-pc-run-false
 (proper-p-statep (lr->p (l))
 $\wedge$ (p-psw (l) = 'run)
 $\wedge$ (p-psw (p (p-set-pc (lr->p (l), pc), p-false-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 false)))
 $\rightarrow$ (p-pc (p (p-set-pc (lr->p (l), pc), p-false-clock (p-set-pc (lr->p (l), pc))))
 $=$ add-addr (pc, 1))

THEOREM: p-pc-run-falsep
 (proper-p-statep (lr->p (l))
 $\wedge$ (p-psw (l) = 'run)
 $\wedge$ (p-psw (p (p-set-pc (lr->p (l), pc), p-falsep-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 falsep)))
 $\rightarrow$ (p-pc (p (p-set-pc (lr->p (l), pc), p-falsep-clock (p-set-pc (lr->p (l), pc))))
 $=$ add-addr (pc, 1))

THEOREM: p-pc-run-listp
 (proper-p-statep (lr->p (l))
 $\wedge$ (p-psw (l) = 'run)
 $\wedge$ (p-psw (p (p-set-pc (lr->p (l), pc), p-listp-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 listp)))
 $\rightarrow$ (p-pc (p (p-set-pc (lr->p (l), pc), p-listp-clock (p-set-pc (lr->p (l), pc))))
 $=$ add-addr (pc, 1))

THEOREM: p-pc-run-nlistp
 (proper-p-statep (lr->p (l))
 $\wedge$ (p-psw (l) = 'run)
 $\wedge$ (p-psw (p (p-set-pc (lr->p (l), pc), p-nlistp-clock (p-set-pc (lr->p (l), pc))))
 $=$ 'run)
 $\wedge$ lr-programs-properp (l, table)

$$\begin{aligned}
& \wedge \text{ (unlabel (get (offset (pc),} \\
& \quad \text{program-body (assoc (area-name (pc),} \\
& \quad \quad \text{p-prog-segment (lr->p (l))))))} \\
& = \text{ ' (call nlistp))} \\
\rightarrow & \text{ (p-pc (p (p-set-pc (lr->p (l), pc), p-nlistp-clock (p-set-pc (lr->p (l), pc))))} \\
& = \text{ add-addr (pc, 1))}
\end{aligned}$$

THEOREM: p-pc-run-true

$$\begin{aligned}
& \text{(proper-p-statep (lr->p (l))} \\
& \wedge \text{ (p-psw (l) = 'run)} \\
& \wedge \text{ (p-psw (p (p-set-pc (lr->p (l), pc), p-true-clock (p-set-pc (lr->p (l), pc))))} \\
& \quad = \text{ 'run)} \\
& \wedge \text{ lr-programs-properp (l, table)} \\
& \wedge \text{ (unlabel (get (offset (pc),} \\
& \quad \text{program-body (assoc (area-name (pc),} \\
& \quad \quad \text{p-prog-segment (lr->p (l))))))} \\
& = \text{ ' (call true))} \\
\rightarrow & \text{ (p-pc (p (p-set-pc (lr->p (l), pc), p-true-clock (p-set-pc (lr->p (l), pc))))} \\
& = \text{ add-addr (pc, 1))}
\end{aligned}$$

THEOREM: p-pc-run-truep

$$\begin{aligned}
& \text{(proper-p-statep (lr->p (l))} \\
& \wedge \text{ (p-psw (l) = 'run)} \\
& \wedge \text{ (p-psw (p (p-set-pc (lr->p (l), pc), p-truep-clock (p-set-pc (lr->p (l), pc))))} \\
& \quad = \text{ 'run)} \\
& \wedge \text{ lr-programs-properp (l, table)} \\
& \wedge \text{ (unlabel (get (offset (pc),} \\
& \quad \text{program-body (assoc (area-name (pc),} \\
& \quad \quad \text{p-prog-segment (lr->p (l))))))} \\
& = \text{ ' (call truep))} \\
\rightarrow & \text{ (p-pc (p (p-set-pc (lr->p (l), pc), p-truep-clock (p-set-pc (lr->p (l), pc))))} \\
& = \text{ add-addr (pc, 1))}
\end{aligned}$$

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

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

$$\begin{aligned}
& \text{p-set-pc (lr->p (lr-eval ('list,} \\
& \qquad \qquad \qquad \text{lr-set-pos (l, pos),} \\
& \qquad \qquad \qquad \text{c)),} \\
& \qquad \qquad \qquad \text{lr-return-pc (l))))) \\
& = \text{'run}) \\
\rightarrow & \text{(p-pc (p-run-subr (car (lr-expr (l)),} \\
& \qquad \qquad \text{p-set-pc (lr->p (lr-eval ('list,} \\
& \qquad \qquad \qquad \text{lr-set-pos (l, pos),} \\
& \qquad \qquad \qquad \text{c)),} \\
& \qquad \qquad \qquad \text{lr-return-pc (l))))) \\
& = \text{add-addr (lr-return-pc (l), 1)) endlet}
\end{aligned}$$

THEOREM: lr-eval-p-pc-equivalence-helper-11

$$\begin{aligned}
& ((\text{p-psw (l) = 'run}) \\
& \wedge (\text{flag} \neq \text{'list}) \\
& \wedge (c \neq 0) \\
& \wedge (c \in \mathbf{N}) \\
& \wedge \text{listp (lr-expr (l))} \\
& \wedge (\text{car (lr-expr (l))} \neq \text{'if}) \\
& \wedge (\text{car (lr-expr (l))} \neq \text{S-TEMP-EVAL}) \\
& \wedge (\text{car (lr-expr (l))} \neq \text{S-TEMP-TEST}) \\
& \wedge (\text{car (lr-expr (l))} \neq \text{S-TEMP-FETCH}) \\
& \wedge (\text{car (lr-expr (l))} \neq \text{'quote}) \\
& \wedge \text{subrp (car (lr-expr (l)))} \\
& \wedge (\text{p-psw (lr-eval ('list, lr-set-pos (l, pos), c)) = 'run}) \\
& \wedge (\text{p (lr->p (l), p-clock1 ('list, lr-set-pos (l, pos), c))} \\
& \quad = \text{p-set-pc (lr->p (lr-eval ('list, lr-set-pos (l, pos), c)),} \\
& \qquad \qquad \text{p-final-pc ('list, lr-set-pos (l, pos), 0))}) \\
& \wedge \text{proper-p-statep (lr->p (l))} \\
& \wedge \text{good-posp1 (offset (p-pc (l)), program-body (p-current-program (l)))} \\
& \wedge \text{lr-programs-properp (l, table)} \\
& \wedge (\text{p-psw (lr-apply-subr (l, lr-eval ('list, lr-set-pos (l, pos), c))} \\
& \quad = \text{'run}) \\
& \wedge (\text{pos} = \text{dv (offset (p-pc (l)), 1))) \\
\rightarrow & (\text{p-set-pc (lr->p (lr-apply-subr (l, lr-eval ('list, lr-set-pos (l, pos), c))}, \\
& \qquad \qquad \text{p-final-pc (flag, l, 0))} \\
& = \text{p (lr->p (l),} \\
& \quad \text{p-clock1 ('list, lr-set-pos (l, pos), c)} \\
& \quad + \text{p-run-subr-clock (l,} \\
& \qquad \qquad \text{lr-eval ('list, lr-set-pos (l, pos), c))})
\end{aligned}$$

THEOREM: p-set-pc-twice

$$\text{p-set-pc (p-set-pc (p, pc1), pc2) = p-set-pc (p, pc2)}$$

THEOREM: lr-eval-p-pc-equivalence-helper-12-helper-1

$$\begin{aligned}
& (\text{good-posp1} (\text{offset} (\text{p-pc} (l)), \text{program-body} (\text{p-current-program} (l))) \\
& \wedge \text{lr-programs-properp} (l, \text{table}) \\
& \wedge \text{listp} (\text{lr-expr} (l)) \\
& \wedge (\text{car} (\text{lr-expr} (l)) \neq \text{'if}) \\
& \wedge (\text{car} (\text{lr-expr} (l)) \neq \text{'quote}) \\
& \wedge (\neg \text{subrp} (\text{car} (\text{lr-expr} (l)))) \\
& \wedge \text{litatom} (\text{car} (\text{lr-expr} (l))) \\
& \wedge (\text{p-psw} (\text{lr-eval} (\mathbf{t}, \text{lr-funcall} (l, \text{new-l}), c - 1)) = \text{'run}) \\
& \wedge (\text{p-psw} (\text{new-l}) = \text{'run}) \\
& \wedge (\text{p-prog-segment} (l) = \text{p-prog-segment} (\text{new-l})) \\
\rightarrow & (\text{p} (\text{p-set-pc} (\text{lr->p} (\text{new-l}), \\
& \quad \text{add-addr} (\text{lr-p-pc} (l), \\
& \quad \quad \text{lr-p-c-size-list} (\text{arity} (\text{car} (\text{lr-expr} (l))), \\
& \quad \quad \text{lr-expr} (l))), \\
& \quad \mathbf{1}) \\
& = \text{lr->p} (\text{lr-funcall} (l, \text{new-l}))
\end{aligned}$$

THEOREM: lr-expr-funcall

$$\begin{aligned}
& ((\text{p-psw} (\text{lr-funcall} (l, \text{new-l})) = \text{'run}) \\
& \wedge (\text{p-prog-segment} (l) = \text{p-prog-segment} (\text{new-l}))) \\
\rightarrow & (\text{lr-expr} (\text{lr-funcall} (l, \text{new-l})) \\
& = \text{program-body} (\text{assoc} (\text{user-fname} (\text{car} (\text{lr-expr} (l))), \\
& \quad \text{p-prog-segment} (l))))
\end{aligned}$$

THEOREM: unlabel-car-last-comp-body

$$\text{unlabel} (\text{car} (\text{last} (\text{comp-body} (\text{body})))) = \text{'(ret)}$$

THEOREM: unlabel-get-last-funcall-body-assoc-comp-programs

$$\begin{aligned}
& (\text{listp} (\text{lr-expr} (l)) \\
& \wedge (\neg \text{subrp} (\text{car} (\text{lr-expr} (l)))) \\
& \wedge (\text{car} (\text{lr-expr} (l)) \neq \text{'quote}) \\
& \wedge \text{litatom} (\text{car} (\text{lr-expr} (l))) \\
& \wedge \text{good-posp1} (\text{offset} (\text{p-pc} (l)), \text{program-body} (\text{p-current-program} (l))) \\
& \wedge \text{lr-programs-properp} (l, \text{table}) \\
& \wedge (\text{prog-seg} = \text{p-prog-segment} (l)) \\
\rightarrow & (\text{unlabel} (\text{get} (\text{lr-p-c-size} (\mathbf{t}, \\
& \quad \text{program-body} (\text{assoc} (\text{user-fname} (\text{car} (\text{lr-expr} (l))), \\
& \quad \quad \text{prog-seg}))), \\
& \quad \text{program-body} (\text{assoc} (\text{user-fname} (\text{car} (\text{lr-expr} (l))), \\
& \quad \quad \text{comp-programs} (\text{prog-seg})))))) \\
& = \text{'(ret)}
\end{aligned}$$

THEOREM: lr-eval-preserves-cdr-p-ctrl-stk-lr-funcall

$$\begin{aligned}
& (\text{listp} (\text{lr-expr} (l)) \\
& \wedge (\neg \text{subrp} (\text{car} (\text{lr-expr} (l))))
\end{aligned}$$

$\wedge$ (car (lr-expr (l)) $\neq$ 'if)
 $\wedge$ (car (lr-expr (l)) $\neq$ 'quote)
 $\wedge$ litatom (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 (lr-eval (**t**, lr-funcall (l , new- l), $c - 1$)) = 'run)
 $\wedge$ (p-psw (new- l) = 'run)
 $\wedge$ (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), $c - 1$)))
 $\quad$ = p-ctrl-stk (new- l))

THEOREM: lr-eval-preserves-ret-pc-car-p-ctrl-stk
 (proper-p-statep (lr->p (l))
 $\wedge$ good-posp1 ($flag$, 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))
 $\rightarrow$ (ret-pc (car (p-ctrl-stk (lr-eval ($flag$, l , c))))
 $\quad$ = ret-pc (car (p-ctrl-stk (l))))

THEOREM: lr-eval-preserves-ret-pc-car-p-ctrl-stk-lr-funcall
 (listp (lr-expr (l))
 $\wedge$ ($\neg$ subrp (car (lr-expr (l))))
 $\wedge$ (car (lr-expr (l)) $\neq$ 'if)
 $\wedge$ (car (lr-expr (l)) $\neq$ 'quote)
 $\wedge$ litatom (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 (lr-eval (**t**, lr-funcall (l , new- l), $c - 1$)) = 'run)
 $\wedge$ (p-psw (new- l) = 'run)
 $\wedge$ (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), $c - 1$))))
 $\quad$ = 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)))
 $\wedge$ lr-programs-properp (l , table)
 $\wedge$ 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))))
 $\wedge$ proper-p-statep (lr->p (l))


```

^ litatom (car (lr-expr (l)))
^ lr-proper-formalsp (cdr (p-prog-segment (l)))
^ (p-psw (lr-eval (t, lr-funcall (l, new-l), c - 1)) = 'run)
^ (p-psw (new-l) = 'run)
→ (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)
^ (flag ≠ 'list)
^ (c ≠ 0)
^ (c ∈ ℕ)
^ 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)
^ (¬ subrp (car (lr-expr (l))))
^ litatom (car (lr-expr (l)))
^ (p-psw (lr-eval ('list, lr-set-pos (l, pos), c)) = 'run)
^ (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)))
^ (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)))
^ proper-p-statep (lr->p (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)))
^ (p-psw (lr-eval (t, fs, c - 1)) = 'run)

```

$$\begin{aligned}
& \wedge (pos = dv(\text{offset}(\text{p-pc}(l)), 1))) \\
\rightarrow & (\text{p-set-pc}(\text{lr->p}(\text{lr-pop-cstk}(\text{lr-eval}(\mathbf{t}, fs, c - 1))), \\
& \quad \text{p-final-pc}(flag, l, 0)) \\
& = \text{p}(\text{lr->p}(l), \\
& \quad \text{p-clock1}('list, \text{lr-set-pos}(l, pos), c) \\
& \quad + 1 \\
& \quad + \text{p-clock1}(\mathbf{t}, fs, c - 1) \\
& \quad + 1)) \mathbf{endlet}
\end{aligned}$$

THEOREM: lr-eval-p-pc-equivalence

$$\begin{aligned}
& (\text{proper-p-statep}(\text{lr->p}(l)) \\
& \wedge \text{good-posp}(flag, \text{offset}(\text{p-pc}(l)), \text{program-body}(\text{p-current-program}(l))) \\
& \wedge ((flag \neq 'list) \\
& \quad \vee (\text{car}(\text{cur-expr}(\text{butlast}(\text{offset}(\text{p-pc}(l))), \\
& \quad \quad \text{program-body}(\text{p-current-program}(l)))) \\
& \quad \neq 'if)) \\
& \wedge \text{lr-programs-properp}(l, table) \\
& \wedge \text{lr-p-proper-statep}(\text{p-temp-stk}(l), \\
& \quad \text{p-ctrl-stk}(l), \\
& \quad \text{p-data-segment}(l), \\
& \quad table) \\
& \wedge \text{lr-proper-formalsp}(\text{cdr}(\text{p-prog-segment}(l))) \\
& \wedge (\text{p-psw}(\text{lr-eval}(flag, l, c)) = 'run)) \\
\rightarrow & (\text{p}(\text{lr->p}(l), \text{p-clock1}(flag, l, c)) \\
& \quad = \text{p-set-pc}(\text{lr->p}(\text{lr-eval}(flag, l, c)), \text{p-final-pc}(flag, l, 0)))
\end{aligned}$$

EVENT: Disable lr-expr-cur-expr-if-same.

DEFINITION:

$$\begin{aligned}
& \text{logic->p-clock}(expr, \\
& \quad alist, \\
& \quad program-names, \\
& \quad heap-size, \\
& \quad max-temp-stk-size, \\
& \quad max-ctrl-stk-size, \\
& \quad word-size) \\
& = (\text{p-clock1}(\mathbf{t}, \\
& \quad \text{s->lr}(\text{logic->s}(expr, alist, program-names), \\
& \quad \quad heap-size, \\
& \quad \quad max-temp-stk-size, \\
& \quad \quad max-ctrl-stk-size, \\
& \quad \quad word-size), \\
& \quad \text{cdr}(\text{v\&c\$}(\mathbf{t}, expr, alist)) + 1) \\
& \quad + 2)
\end{aligned}$$

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)

```

$$\begin{aligned}
& \wedge \text{lr-proper-formalsp}(\text{cdr}(\text{p-prog-segment}(l))) \\
& \wedge (\text{p-psw}(\text{lr-eval}(\mathbf{t}, l, c)) = \text{'run}) \\
& \wedge \text{adpp}(\text{untag}(\text{LR-ANSWER-ADDR}), \text{p-data-segment}(l)) \\
& \wedge (\text{area-name}(\text{p-pc}(l)) = \text{name}(\text{car}(\text{p-prog-segment}(l)))) \\
& \wedge (\text{offset}(\text{p-pc}(l)) = \mathbf{nil}) \\
& \wedge \text{lr-valp}(\text{value}, \\
& \quad \text{car}(\text{p-temp-stk}(\text{lr-eval}(\mathbf{t}, l, c))), \\
& \quad \text{p-data-segment}(\text{lr-eval}(\mathbf{t}, l, c))) \\
\rightarrow & \text{lr-valp}(\text{value}, \\
& \quad \text{fetch}(\text{LR-ANSWER-ADDR}, \\
& \quad \text{p-data-segment}(\text{p}(\text{p}(\text{lr->p}(l), \text{p-clock1}(\mathbf{t}, l, c)), 2))), \\
& \quad \text{p-data-segment}(\text{p}(\text{p}(\text{lr->p}(l), \text{p-clock1}(\mathbf{t}, l, c)), 2)))
\end{aligned}$$

THEOREM: name-car-p-prog-segment-s->lr
 $\text{name}(\text{car}(\text{p-prog-segment}(\text{s->lr}(s, \text{heap-size}, \text{max-ctrl}, \text{max-temp}, \text{word-size}))))$
 $= \text{name}(\text{car}(\text{s-progs}(s)))$

THEOREM: p-pc-s->lr
 $\text{p-pc}(\text{s->lr}(s, \text{heap-size}, \text{max-temp}, \text{max-ctrl}, \text{word-size}))$
 $= \text{tag}(\text{'pc}, \text{cons}(\text{s-pname}(s), \text{s-pos}(s)))$

THEOREM: adpp-untag-lr-answer-addr-s->lr
 $\text{adpp}(\text{identity}(\text{untag}(\text{LR-ANSWER-ADDR})),$
 $\text{p-data-segment}(\text{s->lr}(s, \text{heap-size}, \text{max-ctrl}, \text{max-temp}, \text{word-size})))$

THEOREM: lr-s-similar-statesp-s->lr-logic->s-lr-data-seg-table
 $((\text{heap-size} \not\leq \text{s-total-heap-reqs}(\text{s-progs}(\text{logic->s}(\text{expr}, \text{alist}, \text{pnames})),$
 $\quad \text{alist},$
 $\quad \text{heap-size}))$
 $\wedge \text{l-restrictedp}(\text{pnames}, \text{alist})$
 $\wedge \text{l-data-seg-body-restrictedp}(\mathbf{t}, \text{expr}))$
 $\rightarrow \text{lr-s-similar-statesp}(\text{alist},$
 $\quad \mathbf{nil},$
 $\quad \text{s->lr}(\text{logic->s}(\text{expr}, \text{alist}, \text{pnames}),$
 $\quad \text{heap-size},$
 $\quad \text{max-ctrl},$
 $\quad \text{max-temp},$
 $\quad \text{word-size}),$
 $\quad \text{cdr}(\text{lr-data-seg-table}(\text{s-progs}(\text{logic->s}(\text{expr},$
 $\quad \text{alist},$
 $\quad \text{pnames}))),$
 $\quad \text{alist},$
 $\quad \text{heap-size}))$

THEOREM: s-eval-flag-run-v&c\$-not-f-flag-t

$(v\&c\$(\mathbf{t}, \text{expr}, \text{alist})$
 $\wedge (\text{cdr}(v\&c\$(\mathbf{t}, \text{expr}, \text{alist})) < c)$
 $\wedge \text{l-proper-programsp}(\text{pnames})$
 $\wedge \text{l-proper-exprp}(\mathbf{t}, \text{expr}, \text{pnames}, \text{strip-cars}(\text{alist}))$
 $\wedge \text{all-litatoms}(\text{strip-cars}(\text{alist}))$
 $\rightarrow (\text{s-err-flag}(\text{s-eval}(\mathbf{t}, \text{logic->s}(\text{expr}, \text{alist}, \text{pnames}), c)) = \text{'run})$

THEOREM: $\text{lr-proper-formalsp-cdr-p-prog-segment-s->lr-logic->s}$
 $(\text{l-proper-programsp}(\text{pnames})$
 $\wedge \text{l-proper-exprp}(\mathbf{t}, \text{expr}, \text{pnames}, \text{strip-cars}(\text{alist}))$
 $\wedge \text{all-litatoms}(\text{strip-cars}(\text{alist}))$
 $\rightarrow \text{lr-proper-formalsp}(\text{cdr}(\text{p-prog-segment}(\text{s->lr}(\text{logic->s}(\text{expr},$
 $\text{alist},$
 $\text{pnames}),$
 $\text{heap-size},$
 $\text{max-ctrl},$
 $\text{max-temp},$
 $\text{word-size}))))$

THEOREM: $\text{lr-s-similar-params-lr-good-pointerps-strip-cdrs}$
 $\text{lr-s-similar-params}(\text{s-params}, \text{lr-params}, \text{data-seg})$
 $\rightarrow \text{lr-good-pointerps}(\text{strip-cdrs}(\text{lr-params}), \text{data-seg})$

THEOREM: $\text{lr-p-proper-statep-s->lr}$
 $((\text{heap-size} \not\leq \text{s-total-heap-reqs}(\text{s-progs}(s), \text{s-params}(s), \text{heap-size}))$
 $\wedge \text{s-restrictedp}(\text{s-progs}(s), \text{s-params}(s))$
 $\wedge (\text{params} = \text{s-params}(s)))$
 $\rightarrow \text{lr-p-proper-statep}(\text{p-temp-stk}(\text{s->lr}(s,$
 $\text{heap-size},$
 $\text{max-ctrl},$
 $\text{max-temp},$
 $\text{word-size})),$
 $\text{p-ctrl-stk}(\text{s->lr}(s,$
 $\text{heap-size},$
 $\text{max-ctrl},$
 $\text{max-temp},$
 $\text{word-size})),$
 $\text{p-data-segment}(\text{s->lr}(s,$
 $\text{heap-size},$
 $\text{max-ctrl},$
 $\text{max-temp},$
 $\text{word-size})),$
 $\text{cdr}(\text{lr-data-seg-table}(\text{s-progs}(s), \text{params}, \text{heap-size})))$

THEOREM: $\text{proper-p-statep-lr->p-s->lr-logic->s}$


```

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-code-mpile-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-listp, 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-addr, 35, 284, 292, 324,
344
- all-undef-addr-strip-cadrs-lr-
make-temp-var-dcls, 292
- all-undef-addr-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
- caddr-max-r, 190
- car-append, 57
- car-car-lr-compile-programs-progs, 132
- car-comp-body-1-litatom, 347
- car-comp-body-lr-expr-3-temp-test, 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-current-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-pointers, 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-properp-p-areasp, 102
- definedp-litatom-lr-properp-p-areas, 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-data-seg, 234
- definedp-lr-make-temp-name-alist, 151
 - 1, 151
- definedp-name-p-objectp-tag-0-lr-properp-p-areasp, 237
- definedp-object-cdr-lr-compile-quote-list, 254
- definedp-pairlist, 151
- definedp-s-temps-s-eval, 147
- definedp-strip-cars-append-member-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-init-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-init-data-seg-table, 221
- definedp-table-lr-compile-quote-self, 240
- definedp-user-fname-s-construct-programs, 331
- definition, 37
- definitions-subrps-lr-programs-properp, 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-lr-valp, 379
- deposit-cons, 218
- deposit-deposit, 109
- deposit-free-ptr-preserves-lr-valp, 107
- deposit-good-node-preserves-lr-proper-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-bo
dy-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-c
adr-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-pr
ograms-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-progra
m-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-p
roper-exprp, 62
- good-posp1-cons-lessp-4-if-s-pr
oper-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-
caddr, 68
- ihint-2, 110–112
- increment-num-list-count-code-li
st1, 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-d
ata-seg-body-restrictedp, 331
- l-data-seg-list-restrictedp, 330, 331
- l-data-seg-list-restrictedp-s-d
ata-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-lr-eval-flag-t, 192
- length-p-temp-stk-p-run-subr, 168
- length-p-temp-stk-p-run-subr-helper-1, 167
- length-pair-formals-with-addresses, 281
- length-pair-temps-with-initial-values, 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-subaddr, 105
 - addr-2, 228
 - addr-3, 230
- lessp-index-lessp-lr-p-c-size-list, 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-sub1-length, 59
- lessp-lr-p-pc-1-lr-p-c-size, 60
- lessp-lr-p-pc-1-lr-p-c-size-helper-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-cadr-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-fact, 113
- ct-zero, 113
- lr-boundary-nodep-lessp-lr-node
 - size-0, 225
 - size-1, 225
 - size-2, 225
- lr-boundary-nodep-lessp-plus-fact, 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-node-size-anything, 102
- lr-boundaryp-nodep-difference-node-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-anything-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-deposit, 118
- lr-check-free-nodes-deposit-a-list-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-nodelist, 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-expr-nx, 191
- lr-check-resourcesp-listp-s-expr-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-tstk-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-a
 ll-nodes-fact, 273
 lr-count-free-nodes-at-most, 248
 lr-count-free-nodes-delete-depo
 sit, 186
 lr-count-free-nodes-deposit-a-li
 st-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-node
 s, 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-ope
 ner, 301
 lr-data-seg-table-body-flag-t-f
 lag-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-resou
 rcesp-temp-test, 194
 lr-do-temp-fetch-run-lr-eval-te
 mp-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-pos, 123
- lr-eval-preserves-cdr-p-ctrl-stk, 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-bindings-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-statep, 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-params, 172
- lr-eval-preserves-lr-s-similar-temp, 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-bindings-car-p-ctrl-stk, 122
 - bindings-car-p-ctrl-stk-lr-set-pos, 123
- lr-eval-preserves-strip-cars-lr-temp-car-p-ctrl-stk, 148
- lr-eval-s->lr1-flag-list-opener
 - 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-ctrl-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-flr-set-pos, 185
- lr-eval-s-eval-flag-t-s-ans-non-flr-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
- lr-free-list-nodes-lr-init-heap-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
- lr-good-pointerp-tablep-deposit-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-pointers, 336–345, 366, 382
- lr-good-pointers-all-undef-addrs, 344
- lr-good-pointers-append, 344
- lr-good-pointers-cdr, 340
- lr-good-pointers-cons-fetch-car-temp-stk-cdr-car, 341
- lr-good-pointers-cons-fetch-car-temp-stk-cdr-cdr, 341
- lr-good-pointers-cons-fetch-fp-addr-deposit-a-list-cons, 341
- lr-good-pointers-cons-lr-0-add-r-lr-proper-heap, 340
- lr-good-pointers-cons-lr-f-add-r-lr-proper-heap, 340
- lr-good-pointers-cons-lr-t-add-r-lr-proper-heap, 340
- lr-good-pointers-deposit-a-list-node, 339
- lr-good-pointers-deposit-free-ptr, 338
- lr-good-pointers-first-n, 344
- lr-good-pointers-popn, 345
- lr-good-pointers-put-assoc, 337
- lr-good-pointers-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
- lr-init-data-seg-table-preserve-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-noddep, 7, 55, 97, 99, 103, 105, 106, 112–115, 118, 119, 128, 154, 155, 161, 162, 185, 186, 254, 257–260, 342
- lr-noddep-car-lr-compile-quote, 254
- lr-noddep-deposit-a-list, 254
- lr-noddep-lr-proper-heapp-noddep, 105
- lr-noddep-member-lr-node-listp, 103
- lr-noddep-member-lr-node-listp-a
 - dpp-untag-listp, 115
 - dpp-untag-numberp-offset, 115
- lr-noddep-member-lr-node-listp-l
 - r-boundaryp-offsetp, 115
- lr-noddep-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-cstk, 172
- lr-params-lr-set-pos, 136
- lr-params-lr-set-temp, 140
- lr-params-p-frame-not-definedp-put-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-fnamesp, 51
- lr-programs-properp-1-lr-compile-programs, 305
- lr-programs-properp-1-lr-proper-exprp, 56
- lr-programs-properp-all-user-fnamesp-strip-cars-cdr, 309
- lr-programs-properp-caar-main, 309
- lr-programs-properp-definedp-car-untag-p-pc, 61
- lr-programs-properp-definedp-subrp-runtime-support, 206
- lr-programs-properp-expr-quote-type-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-exprp-lr-expr, 57
- lr-programs-properp-lr-set-error, 175
- lr-programs-properp-lr-set-expr, 94
- lr-programs-properp-lr-set-pos, 90
- lr-programs-properp-member-lr-expr-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-opened, 306
- lr-programs-properp-s->lr1-definedp-cdr-s-progs, 208
- lr-programs-properp-s->lr1-definedp-s-progs, 212
- lr-proper-ctrl-stkp, 336, 339, 340
- lr-proper-ctrl-stkp-deposit-a-list-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-caddr, 58
 lr-proper-exprp-car-if-caddr, 58
 lr-proper-exprp-car-if-cadr, 58
 lr-proper-exprp-flag-list-cdr-funcall, 290
 lr-proper-exprp-flag-list-cons, 296
 lr-proper-exprp-flag-list-nil, 296
 lr-proper-exprp-flag-not-list-cons-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-not-listp, 304
 lr-proper-exprp-length-cur-expr, 57
 lr-proper-exprp-list-lr-proper-get-t, 56
 lr-proper-exprp-list-quote-opener, 152
 lr-proper-exprp-lr-proper-exprp-cur-expr, 56
 lr-proper-exprp-p-lr-compile-programs, 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-segment-s->lr-logic->s, 382
 lr-proper-formalsp-lr-compile-programs, 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-compile-quote, 222
 lr-proper-free-listp-car-lr-init-data-seg-table, 224
 lr-proper-free-listp-length-sub1-not-lessp, 242
 lr-proper-free-listp-lr-count-freee-nodes-max-addr, 249
 ree-nodes-max-addr-alt, 250
 lr-proper-free-listp-lr-init-data-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-lr-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-helper, 161
- lr-proper-heapp-lr-valp-lr-f-addr, 161
- lr-proper-heapp-lr-valp-lr-t-addr, 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-free-pte, 125
- lr-proper-heapp-nodep-lr-init-data-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-pte-0, 125
- lr-proper-heapp2-lr-init-data-seg, 270
- lr-proper-heapp2-lr-init-data-seg-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-compile-quote, 219
- lr-proper-p-areasp-car-lr-data-seg-table, 281
- lr-proper-p-areasp-car-lr-data-seg-table-body, 223
- lr-proper-p-areasp-car-lr-data-seg-table-list, 234
- lr-proper-p-areasp-car-lr-init-data-seg-table, 234
- lr-proper-p-areasp-deposit-a-list, 120
- lr-proper-p-areasp-deposit-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-pointer-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-pointer-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-similar-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-subr, 163
- lr-valp-car-p-temp-stk-p-run-subr, 163
 - br-cons-helper, 162
- lr-valp-cdr-assoc-firstn-cdr-assoc, 135
- lr-valp-cons, 162
- lr-valp-deposit-a-list-cons, 162
- lr-valp-deposit-a-list-cons-cons, 260
- lr-valp-deposit-a-list-cons-numberp, 260
- lr-valp-deposit-a-list-cons-truep, 261
- lr-valp-deposit-fetch-free-pointer, 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-valp-car-cdr-0, 161
- alp-listp, 161
- lr-valp-fetch-tag-not-true-lr-valp-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-temps, 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-programs-properp, 347
- member-cdr-assoc-strip-cdrs-definedp, 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-pointers-type-addr-untag-whole, 366
- member-make-symbol-max-count-code-list, 13
- member-no-duplicatesp-assoc-equal, 307
- member-strip-cdrs-lr-good-pointerp-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-programs, 132
- name-car-p-prog-segment-s->lr, 381
- name-car-s-progs-logic->s, 331
- name-formal-vars-temp-var-dcls-program-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-occurrences-1, 117
- no-duplicatesp-remove-duplicate-s, 310
- no-duplicatesp-strip-cars-s-construct-programs, 332
- no-duplicatesp-strip-cars-s-progs-logic->s, 332
- no-duplicatesp-strip-cdrs-lr-make-temp-name-alist, 147
- ke-temp-name-alist-1, 147
- not-adpp-untag-add-addr-adpp-untag, 120
- not-adpp-untag-node-not-definedp-lr-heap-name, 116
- not-definedp-not-listp, 291
- not-definedp-user-fname-p-runtime-support-programs, 210
- not-disjointp-member-arg1-cons-arg2, 144
- not-equal-0-count-list, 16
- not-equal-lr-s-eval-temp-setp-not-lr-s-similar-temps, 149
- not-equal-make-symbol-car-gensym

- 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-boundary-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-lr-data-seg-table-heap-r, 323
- not-lessp-lr-set-pos, 205
- not-lessp-lr-p-c-size-flag-t-1, 71
- not-lessp-max-r-caddr, 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-lr-funcall, 92
- not-lessp-p-max-temp-stk-size-lr-funcall, 93
- not-lessp-p-push-tstk, 60
- not-lessp-plus-arity-length-for-mals, 208
- not-lessp-plus-arity-length-for-mals-alt, 211
- not-lessp-x-x, 71
- not-listp-p-prog-segment-lr-expr, 84
- not-listp-p-temp-stk-not-lr-check-result1, 159
- not-listp-p-prog-segment-not-lr-programs-properp, 347
- not-listp-s-progs-not-s-good-statep, 217

- not-lr-check-resourcesp-temp-test-bad-max-temp-stk-size, 215
- not-lr-valp-lr-undef-addr, 143
- not-member-car-gensym-lr-make-temp-name-alist-1-cdr, 147
- not-member-lr-all-nodes-too-small-addr, 228
- not-member-make-symbol-lr-make-temp-name-alist-1-incr, 147
- not-member-no-duplicates-cdr-as-soc, 149
- not-member-no-duplicates-cdr-as-soc-helper, 149
- not-member-occurrences-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-list-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-caddr-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
- numberp-s-eval-temp-ctrl-ws-heap-r-opened, 195
- nx, 40, 41, 56, 58, 110, 131, 133, 155, 175, 179, 191, 295, 334, 349, 352
- object-addr, 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-pr-oper-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-check-
 k-resourcep, 207
 p-psw-run-lr-if-ok-p-psw-run, 176
 p-psw-run-p-psw-lr-if-ok-not-ru-
 n-check-resourcep, 215
 p-psw-run-p-run-subr-lr-check-re-
 sourcep, 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-
 sourcep, 200
 p-psw-run-run-falsep-lr-check-re-
 sourcep, 201
 p-psw-run-run-listp-lr-check-re-
 sourcep, 202
 p-psw-run-run-nlistp-lr-check-re-
 sourcep, 203
 p-psw-run-run-true-lr-check-res-
 ourcesp, 203
 p-psw-run-run-truep-lr-check-re-
 sourcep, 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-addresses,
 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-compile-programs, 318
- program-body-assoc-comp-programs, 59
 - s-1, 59
- program-body-assoc-lr-compile-programs, 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-similar-states, 50
- proper-p-ctrl-stkp-lr-apply-subr, 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-label, 283
- proper-p-instructionp-pop-global-free-ptr, 281
- proper-p-instructionp-push-constant-opener, 64
- proper-p-instructionp-push-global-free-ptr, 282
- proper-p-instructionp-push-local-temp-car, 282
- proper-p-instructionp-push-local-temp-cdr, 282
- proper-p-instructionp-push-local-temp-cons, 282
- proper-p-instructionp-ret, 281
- proper-p-instructionp-set-global, 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-programs, 310
- proper-p-prog-segmentp-grams-1, 308
- proper-p-prog-segmentp-grams-1-helper, 307
- proper-p-prog-segmentp-length-program-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-program-segment, 308
- proper-p-programp-car-code, 292
- proper-p-programp-cdr-code, 292
- proper-p-programp-cons-code, 293
- proper-p-programp-false-code, 293
- proper-p-programp-falsep-code, 293
- proper-p-programp-listp-code, 293
- proper-p-programp-nlistp-code, 293
- proper-p-programp-true-code, 293
- proper-p-programp-truep-code, 294
- proper-p-push-constant-instructionp, 64
- proper-p-state-p-p-run-subr-opener-1, 89
- proper-p-state-p-p-run-subr-opener-2, 89
- proper-p-state-p-p-run-subr-opener-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-programp-stk-max-temp-stk-size, 353
- proper-p-statep-lr->p-equal-word-size-0, 88
- proper-p-statep-lr->p-implies-lr-proper-p-areasp, 109
- proper-p-statep-lr->p-lessp-ctrl-stk-size, 87
- proper-p-statep-lr->p-lessp-max-ctrl-stk-size, 88
- proper-p-statep-lr->p-lessp-max-temp-stk-size, 88
- proper-p-statep-lr->p-lr-eval-list, 158
- proper-p-statep-lr->p-lr-eval-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-addr, 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-opene-r, 252
- s-heap-reqs-flag-list-nil-opene-r, 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-prope
 r-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-su
 brps-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-q
 uote, 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-temp
 - s, 259
- strip-cars-lr-make-temp-name-ali
 - st, 148
 - st-1, 148
- strip-cars-lr-make-temp-var-dcl
 - s, 135
- strip-cars-lr-temps-strip-cars-te
 - mp-var-dcls, 146
- strip-cars-nil-fact, 140
- strip-cars-pair-formals-with-ad
 - resses, 259
- strip-cars-pair-temps-with-initi
 - al-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-initi
 - al-values, 344
- strip-cdrs-pairlist, 343
- strip-logic-fnames, 35, 56, 57, 208,
 - 289–292, 307, 308
- strip-logic-fnames-cdr-lr-compi
 - le-programs, 208
- strip-logic-fnames-lr-compile-p
 - rograms, 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-progra
 - ms, 61
 - ms-1, 60
 - ms-lr-programs-properp, 91
- temp-var-dcls-assoc-p-current-p
 - rogram-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-a
 - ssoc-comp-programs, 375
- unlabel-get-lr-p-pc-program-bod
 - y-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