Event: Start with the initial **thm** theory.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                    ;;
;;                        SUBSIDIARY FUNCTIONS                        ;;
;;                                                                    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; I suspect that many of these lemmas are not necessary but I'm keeping
;; them for now.  I'll eliminate many of them later as I see that they
;; are not needed.

;; I'm adopting the convention that all alist components are cons
;; pairs.  That is, of the form (x . y) not of the form (x y).
```

Definition: $\text{name}(exp) = \text{car}(exp)$

Event: Disable name.

Theorem: name-expansion
$\text{name}(\text{cons}(x,\ y)) = x$

Definition:
$\text{assoc}(x,\ y)$
= **if** listp (y)
then if $x = \text{caar}(y)$ **then** car (y)
else assoc (x, cdr (y)) **endif**
else f endif

Theorem: assoc-value1
$\text{assoc}(x, \text{cons}(\text{cons}(x,\ y),\ z)) = \text{cons}(x,\ y)$

Definition: $\text{value}(name,\ alist) = \text{cdr}(\text{assoc}(name,\ alist))$

Theorem: value-expansion3
$\text{value}(x, \text{cons}(\text{cons}(x,\ z),\ alist)) = z$

Theorem: value-expansion2
$(x \neq y) \rightarrow (\text{value}(x, \text{cons}(\text{cons}(y,\ z),\ alist)) = \text{value}(x,\ alist))$

Event: Disable value.

Definition:

$\text{max}(x,\ y)$
$=$ **if** $x < y$ **then** y
else x **endif**

DEFINITION:
$\text{append}(x,\ y)$
$=$ **if** $\text{listp}(x)$ **then** $\text{cons}(\text{car}(x),\ \text{append}(\text{cdr}(x),\ y))$
else y **endif**

THEOREM: append-cons-rewrite
$\text{append}(\text{cons}(x,\ y),\ z) = \text{cons}(x,\ \text{append}(y,\ z))$

THEOREM: append-nil-lemma
$\text{append}(\textbf{nil},\ x) = x$

THEOREM: assoc-not-affected-by-extra-element
$(y \neq \text{car}(x))$
$\rightarrow \quad (\text{assoc}(y,\ \text{append}(\textit{lst1},\ \text{cons}(x,\ \textit{lst2}))) = \text{assoc}(y,\ \text{append}(\textit{lst1},\ \textit{lst2})))$

DEFINITION:
$\text{plistp}(x)$
$=$ **if** $x \simeq \textbf{nil}$ **then** $x = \textbf{nil}$
else $\text{plistp}(\text{cdr}(x))$ **endif**

DEFINITION:
$\text{length-plistp}(\textit{lst},\ n) = (\text{plistp}(\textit{lst}) \wedge (\text{length}(\textit{lst}) = n))$

THEOREM: associativity-of-append
$\text{append}(\text{append}(x,\ y),\ z) = \text{append}(x,\ \text{append}(y,\ z))$

THEOREM: length-distributes
$\text{length}(\text{append}(\textit{lst1},\ \textit{lst2})) = (\text{length}(\textit{lst1}) + \text{length}(\textit{lst2}))$

THEOREM: not-zerop-length-listp
$(\text{length}(x) \not\simeq \mathtt{0}) \rightarrow \text{listp}(x)$

EVENT: Disable not-zerop-length-listp.

THEOREM: length-cons
$\text{length}(\text{cons}(x,\ y)) = (1 + \text{length}(y))$

EVENT: Disable length-cons.

THEOREM: append-rewrite2
$(\text{append}(x,\ y) = \text{append}(x,\ z)) = (y = z)$

Theorem: append-cons-rewrite2
$(\text{cons}(x, y) = \text{cons}(x, z)) = (y = z)$

Theorem: member-distributes
$(x \in \text{append}(y, z)) = ((x \in y) \vee (x \in z))$

Definition:
$\text{reverse}(x)$
$= \;$ **if** $\text{listp}(x)$ **then** $\text{append}(\text{reverse}(\text{cdr}(x)), \text{list}(\text{car}(x)))$
else nil endif

Theorem: reverse-preserves-length
$\text{length}(\text{reverse}(lst)) = \text{length}(lst)$

Theorem: reverse-plistp
$\text{plistp}(\text{reverse}(lst))$

Theorem: reverse-reverse
$\text{plistp}(lst) \rightarrow (\text{reverse}(\text{reverse}(lst)) = lst)$

Theorem: listp-implies-non-zero-length
$\text{listp}(x) \rightarrow (\text{length}(x) \neq 0)$

Definition:
$\text{subset}(x, y)$
$= \;$ **if** $x \simeq$ **nil then t**
else $(\text{car}(x) \in y) \wedge \text{subset}(\text{cdr}(x), y)$ **endif**

Theorem: superset-preserves-membership
$(\text{subset}(x, y) \wedge (z \in x)) \rightarrow (z \in y)$

Event: Disable superset-preserves-membership.

Theorem: cons-preserves-subset
$\text{subset}(x, y) \rightarrow \text{subset}(x, \text{cons}(z, y))$

Event: Disable cons-preserves-subset.

Theorem: subset-reflexive
$\text{subset}(y, y)$

Theorem: cons-preserves-subset2
$\text{subset}(y, \text{cons}(x, y))$

Theorem: subset-distributes
$\text{subset}(\text{append}(x, y), z) = (\text{subset}(x, z) \wedge \text{subset}(y, z))$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                        ;;
;;                           ALIST FUNCTIONS                              ;;
;;                                                                        ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
listcars (*lst*)
= **if** *lst* $\simeq$ **nil then nil**
else cons (car (car (*lst*)), listcars (cdr (*lst*))) **endif**

THEOREM: listcars-plistp
plistp (listcars (x))

THEOREM: cons-distributes-over-listcars
listcars (cons (x, y)) = cons (car (x), listcars (y))

THEOREM: listcars-distributes
listcars (append (x, y)) = append (listcars (x), listcars (y))

THEOREM: listcars-preserves-length
length (listcars (*lst*)) = length (*lst*)

THEOREM: not-member-listcars-not-assoc
($x \notin$ listcars (*lst*)) $\rightarrow$ (assoc (x, *lst*) = **f**)

THEOREM: member-listcars-assoc
($x \notin$ listcars (*lst2*)) $\rightarrow$ (assoc (x, append (*lst2*, *lst3*)) = assoc (x, *lst3*))

DEFINITION:
no-duplicates (*lst*)
= **if** *lst* $\simeq$ **nil then t**
elseif car (*lst*) $\in$ cdr (*lst*) **then f**
else no-duplicates (cdr (*lst*)) **endif**

THEOREM: no-duplicates-append
no-duplicates (append (x, y)) $\rightarrow$ (no-duplicates (x) $\wedge$ no-duplicates (y))

THEOREM: no-duplicates-append-append
no-duplicates (append (x, append (y, z))) $\rightarrow$ no-duplicates (append (x, z))

EVENT: Disable no-duplicates-append-append.

THEOREM: no-duplicates-member2
(no-duplicates (append (*lst1*, *lst2*)) $\wedge$ ($x \in$ *lst1*))
$\rightarrow$ (($x \in$ *lst2*) = **f**)

EVENT: Disable no-duplicates-member2.

THEOREM: subset-preserves-no-duplicates
$(\text{no-duplicates}(\text{append}(x, y)) \land \text{no-duplicates}(z) \land \text{subset}(z, y))$
$\rightarrow \quad \text{no-duplicates}(\text{append}(x, z))$

EVENT: Disable subset-preserves-no-duplicates.

THEOREM: no-duplicates-cons-append2
$\text{no-duplicates}(\text{cons}(x, \text{append}(y, z))) \rightarrow \text{no-duplicates}(\text{cons}(x, z))$

EVENT: Disable no-duplicates-cons-append2.

THEOREM: no-duplicates-member-append2
$\text{no-duplicates}(\text{append}(a, \text{cons}(x, b))) \rightarrow \text{no-duplicates}(\text{append}(a, b))$

THEOREM: no-duplicates-duplication
$((x \in \mathit{lst1}) \land (x \in \mathit{lst2}))$
$\rightarrow \quad (\text{no-duplicates}(\text{append}(\mathit{lst1}, \mathit{lst2})) = \mathbf{f})$

THEOREM: no-duplicates-cons-append
$((x \notin \mathit{lst1}) \land (x \notin \mathit{lst2}) \land \text{no-duplicates}(\text{append}(\mathit{lst1}, \mathit{lst2})))$
$\rightarrow \quad \text{no-duplicates}(\text{append}(\mathit{lst1}, \text{cons}(x, \mathit{lst2})))$

EVENT: Disable no-duplicates-cons-append.

THEOREM: append-plistp-nil-lemma
$\text{plistp}(x) \rightarrow (\text{append}(x, \mathbf{nil}) = x)$

THEOREM: no-duplicates-commutes-append
$(\text{plistp}(x) \land \text{plistp}(y) \land \text{no-duplicates}(\text{append}(x, y)))$
$\rightarrow \quad \text{no-duplicates}(\text{append}(y, x))$

EVENT: Disable no-duplicates-commutes-append.

THEOREM: member-cons
$(x \neq y) \rightarrow ((x \in \text{cons}(y, z)) = (x \in z))$

DEFINITION: $\text{all-cars-unique}(\mathit{lst}) = \text{no-duplicates}(\text{listcars}(\mathit{lst}))$

THEOREM: cars-unique-cars-unique
$(\text{all-cars-unique}(\text{cons}(y, \mathit{lst})) \land (x \in \mathit{lst})) \rightarrow (\text{car}(x) \neq \text{car}(y))$

EVENT: Disable cars-unique-cars-unique.

THEOREM: assoc-unique-member
$((x \in lst) \wedge \text{all-cars-unique}(lst)) \rightarrow (\text{assoc}(\text{car}(x),\ lst) = x)$

THEOREM: no-duplicates-member-cons
$\text{no-duplicates}(\text{append}(x,\ \text{cons}(y,\ z))) \rightarrow (y \notin z)$

EVENT: Disable no-duplicates-member-cons.

THEOREM: all-cars-unique-commutes-append
$(\text{plistp}(x) \wedge \text{plistp}(y) \wedge \text{all-cars-unique}(\text{append}(x,\ y)))$
$\rightarrow \quad \text{all-cars-unique}(\text{append}(y,\ x))$

EVENT: Disable all-cars-unique-commutes-append.

THEOREM: cars-unique-cons
$\text{all-cars-unique}(\text{cons}(x,\ \text{cons}(y,\ z))) \rightarrow \text{all-cars-unique}(\text{cons}(x,\ z))$

EVENT: Disable cars-unique-cons.

THEOREM: cons-car-append-all-cars-unique
$(\text{listp}(x) \wedge \text{all-cars-unique}(\text{append}(x,\ y)))$
$\rightarrow \quad \text{all-cars-unique}(\text{cons}(\text{car}(x),\ y))$

THEOREM: member-car-listcars
$(x \in y) \rightarrow (\text{car}(x) \in \text{listcars}(y))$

THEOREM: cars-unique-names-unique
$(\text{all-cars-unique}(\text{append}(z,\ w)) \wedge (x \in z) \wedge (y \in w))$
$\rightarrow \quad (\text{car}(x) \neq \text{car}(y))$

EVENT: Disable cars-unique-names-unique.

EVENT: Disable all-cars-unique.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                    ;;
;;                     ARITHMETIC FUNCTIONS                           ;;
;;                                                                    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
$\exp(x,\ y)$
$= \quad$ **if** $y \simeq 0$ **then** 1
else $x * \exp(x,\ y - 1)$ **endif**

DEFINITION:
integerp (x) = ((negativep (x) ∧ (negative-guts $(x) \neq \texttt{0}$)) ∨ $(x \in \mathbf{N})$)

DEFINITION:
ilessp $(x,\ y)$
= **if** negativep (x)
then if negativep (y) **then** negative-guts (y) < negative-guts (x)
else t endif
elseif negativep (y) **then f**
else $x < y$ **endif**

DEFINITION:
iplus $(i,\ j)$
= **if** negativep (i)
then if negativep (j) **then** − (negative-guts (i) + negative-guts (j))
elseif j < negative-guts (i) **then** − (negative-guts $(i) - j$)
else j − negative-guts (i) **endif**
elseif negativep (j)
then if i < negative-guts (j) **then** − (negative-guts $(j) - i$)
else i − negative-guts (j) **endif**
else $i + j$ **endif**

DEFINITION:
inegate (i)
= **if** negativep (i) **then** negative-guts (i)
elseif $i \simeq \texttt{0}$ **then** $\texttt{0}$
else $-\ i$ **endif**

DEFINITION: idifference $(i,\ j)$ = iplus $(i$, inegate $(j))$

DEFINITION: ileq $(x,\ y)$ = (¬ ilessp $(y,\ x)$)

THEOREM: zero-iplus-identity
integerp (n) → (iplus $(\texttt{0},\ n) = n$)

THEOREM: iplus-integerp
(integerp (n) ∧ integerp (m)) → integerp (iplus $(n,\ m)$)

EVENT: Disable iplus-integerp.

THEOREM: plus-commutes
$(x + y) = (y + x)$

EVENT: Disable plus-commutes.

THEOREM: plus-0-rewrite
$(x + 0) = \text{fix}(x)$

THEOREM: plus-add1
$(x + 1) = (1 + x)$

EVENT: Disable plus-add1.

THEOREM: times-m-rewrite
$(m * (1 + n)) = (m + (m * n))$

EVENT: Disable times-m-rewrite.

THEOREM: plus-add1-commute
$(x + (1 + n)) = (1 + (x + n))$

EVENT: Disable plus-add1-commute.

THEOREM: add1-preserves-lessp
$(n < m) \rightarrow ((n < (1 + m)) = \mathbf{t})$

EVENT: Disable add1-preserves-lessp.

THEOREM: difference-x-x
$(x - x) = 0$

THEOREM: plus-equality-lemma1
$(m = k) \rightarrow (((n + m) = (n + k)) = \mathbf{t})$

EVENT: Disable plus-equality-lemma1.

THEOREM: difference-rewrite
$(y < x) \rightarrow ((x - y) = (1 + (x - (1 + y))))$

EVENT: Disable difference-rewrite.

THEOREM: sub1-preserves-lessp
$(x < y) \rightarrow (((x - 1) < y) = \mathbf{t})$

THEOREM: add1-difference
$(j \leq k) \rightarrow (((1 + k) - j) = (1 + (k - j)))$

THEOREM: add1-sub1-difference
$(((1 + x) - y) - 1) = (x - y)$

THEOREM: difference-add1
$((1 + n) - n) = \mathtt{1}$

THEOREM: difference-plus-rewrite
$((x + y) - x) = \text{fix}\,(y)$

THEOREM: plus-0-rewrite2
$(n \simeq \mathtt{0}) \rightarrow ((m + n) = \text{fix}\,(m))$

THEOREM: plus-add1-sub1
$(m \not\simeq \mathtt{0}) \rightarrow (((1 + n) + (m - 1)) = (n + m))$

THEOREM: associativity-of-plus
$((x + y) + z) = (x + (y + z))$

THEOREM: difference-n-leq
$(n \not\simeq \mathtt{0}) \rightarrow ((((n - m) - 1) < n) = \mathbf{t})$

EVENT: Disable difference-n-leq.

THEOREM: plus-preserves-lessp
$((x + y) < z) \rightarrow ((x < z) = \mathbf{t})$

THEOREM: plus-preserves-lessp2
$(x < y) \rightarrow ((x < (n + y)) = \mathbf{t})$

THEOREM: lessp-transitive
$((x < \mathit{y1}) \land (\mathit{y2} \not< \mathit{y1})) \rightarrow ((x < \mathit{y2}) = \mathbf{t})$

EVENT: Disable lessp-transitive.

THEOREM: difference-lessp
$(x < (y - z)) \rightarrow ((z < y) = \mathbf{t})$

EVENT: Disable difference-lessp.

THEOREM: plus-preserves-lessp3
$(n \not\simeq \mathtt{0}) \rightarrow ((k < (n + k)) = \mathbf{t})$

EVENT: Disable plus-preserves-lessp3.

THEOREM: difference-difference-plus
$((n \not< l) \land (k \not< n)) \rightarrow ((k - (n - l)) = (l + (k - n)))$

EVENT: Disable difference-difference-plus.

Theorem: zerop-difference-lessp
$(m < n) \rightarrow (((n - m) = 0) = \mathbf{f})$

Event: Disable zerop-difference-lessp.

Theorem: add1-difference2
$(y < x) \rightarrow ((x - (1 + y)) = ((x - y) - 1))$

Event: Disable add1-difference2.

Theorem: sub1-plus5
$(y \not\simeq 0) \rightarrow ((x + (y - 1)) = ((x + y) - 1))$

Event: Disable sub1-plus5.

Theorem: sub1-plus4
$((x \not\simeq 0) \land (y \not\simeq 0)) \rightarrow (((x - 1) + y) = (x + (y - 1)))$

Theorem: difference-lessp2
$((x + y + z) < (m - n)) \rightarrow ((m < (x + y + z + n)) = \mathbf{f})$

Theorem: plus-times-3
$(k \not\simeq 0)$
$\rightarrow \quad (((k * 3) + n) = (1 + (1 + (1 + (((k - 1) * 3) + n)))))$

Theorem: zero-iplus-right-identity
$\text{integerp}(x) \rightarrow (\text{iplus}(x, 0) = x)$

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                 ;;
;;                    MISCELLANEOUS STUFF                          ;;
;;                                                                 ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Definition:
$\text{nth}(x, n)$
$= \quad$ **if** $n \simeq 0$ **then** x
else $\text{nth}(\text{cdr}(x), n - 1)$ **endif**

```
;; Index is used for converting the current condition to a numeric
;; value for the lower level state.
```

DEFINITION:
index $(y,\ lst)$
$=$ **if** $lst \simeq$ **nil** **then** `0`
elseif $y = \text{car}(lst)$ **then** `1`
else $1 + \text{index}(y, \text{cdr}(lst))$ **endif**

THEOREM: member-implies-nonzero-index
$(c \in \textit{cond-list}) \rightarrow (\text{index}(c,\ \textit{cond-list}) \neq \texttt{0})$

EVENT: Disable member-implies-nonzero-index.

THEOREM: member-index-lessp-length
$\text{index}(c,\ \textit{cond-list}) < (1 + \text{length}(\textit{cond-list}))$

EVENT: Disable member-index-lessp-length.

THEOREM: lessp-member-index-length
$(x \in lst) \rightarrow ((\text{index}(x,\ lst) < (1 + \text{length}(lst))) = \mathbf{t})$

THEOREM: index-length
$(\text{length}(lst) < (n - 1)) \rightarrow ((\text{index}(x,\ lst) < n) = \mathbf{t})$

EVENT: Disable index-length.

THEOREM: nth-index-elimination
$(x \in lst) \rightarrow (\text{car}(\text{nth}(lst,\ \text{index}(x,\ lst) - 1)) = x)$

EVENT: Disable nth-index-elimination.

THEOREM: subset-append1
$\text{subset}(x,\ y) \rightarrow (\text{subset}(x,\ \text{append}(y,\ z)) \land \text{subset}(x,\ \text{append}(z,\ y)))$

THEOREM: reorder-subset
$\text{subset}(\text{cons}(x,\ \text{append}(y,\ z)),\ \text{append}(y,\ \text{cons}(x,\ z)))$

EVENT: Disable reorder-subset.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                ;;
;;                         PLISTP LEMMAS                          ;;
;;                                                                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

THEOREM: plistp-cons
$\text{plistp}(\text{cons}(x,\ y)) = \text{plistp}(y)$

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

THEOREM: plistp-atom
$(x \simeq \textbf{nil}) \rightarrow (\text{plistp}(x) = (x = \textbf{nil}))$

THEOREM: zero-length-plistp-nil
$(\text{plistp}(x) \land (\text{length}(x) = 0)) \rightarrow ((x = \textbf{nil}) = \textbf{t})$

EVENT: Disable zero-length-plistp-nil.

THEOREM: cddr-length-plistp-2
$\text{length-plistp}(x,\ 2) \rightarrow (\text{cddr}(x) = \textbf{nil})$

THEOREM: reverse-append-reverse1
$(\text{plistp}(x) \land \text{plistp}(y))$
$\rightarrow \quad (\text{reverse}(\text{append}(x,\ y)) = \text{append}(\text{reverse}(y),\ \text{reverse}(x)))$

THEOREM: reverse-append-reverse
$(\text{plistp}(\mathit{lst1}) \land \text{plistp}(\mathit{lst2}))$
$\rightarrow \quad (\text{reverse}(\text{append}(\text{reverse}(\mathit{lst1}),\ \text{reverse}(\mathit{lst2}))) = \text{append}(\mathit{lst2},\ \mathit{lst1}))$

EVENT: Disable reverse-append-reverse.

THEOREM: append-doesnt-affect-value4
$(x \notin \text{listcars}(z)) \rightarrow (\text{assoc}(x,\ \text{append}(y,\ z)) = \text{assoc}(x,\ y))$

EVENT: Disable append-doesnt-affect-value4.

EVENT: Disable length-plistp.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                   ;;
;;                        GET and PUT LEMMAS                         ;;
;;                                                                   ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
$\text{get}(n,\ \mathit{lst})$
$= \quad$ **if** $n \simeq 0$ **then** $\text{car}(\mathit{lst})$
else $\text{get}(n - 1,\ \text{cdr}(\mathit{lst}))$ **endif**

Theorem: get-0
$\text{get}(0, \text{cons}(x, y)) = x$

Theorem: get-add1
$\text{get}(1 + n, \text{cons}(x, y)) = \text{get}(n, y)$

Theorem: get-length-cons
$(n \not\simeq 0) \rightarrow (\text{get}(n, \text{cons}(x, \mathit{lst})) = \text{get}(n - 1, \mathit{lst}))$

Theorem: get-length-car1
$\text{get}(\text{length}(\mathit{lst}), \text{append}(\mathit{lst}, \mathit{lst2})) = \text{car}(\mathit{lst2})$

Event: Disable get-length-car1.

Theorem: get-length-car
$(n = \text{length}(\mathit{lst})) \rightarrow (\text{get}(n, \text{append}(\mathit{lst}, \mathit{lst2})) = \text{car}(\mathit{lst2}))$

Theorem: get-length-plus
$(n = \text{length}(\mathit{lst1})) \rightarrow (\text{get}(n + m, \text{append}(\mathit{lst1}, \mathit{lst2})) = \text{get}(m, \mathit{lst2}))$

Theorem: get-car
$(n \simeq 0) \rightarrow (\text{get}(n, \mathit{lst}) = \text{car}(\mathit{lst}))$

Theorem: get-sub1
$(n \not\simeq 0) \rightarrow (\text{get}(n, \mathit{lst}) = \text{get}(n - 1, \text{cdr}(\mathit{lst})))$

Theorem: get-append
$(k < \text{length}(\mathit{lst1})) \rightarrow (\text{get}(k, \text{append}(\mathit{lst1}, \mathit{lst2})) = \text{get}(k, \mathit{lst1}))$

Theorem: get-add1-add1-append
$\text{get}(n + (1 + (1 + m)), \text{cons}(x, \text{cons}(y, \mathit{lst}))) = \text{get}(n + m, \mathit{lst})$

Theorem: get-add1-length-plus
$((\text{length}(\mathit{lst1}) = (1 + n)) \wedge (m \not\simeq 0))$
$\rightarrow \quad (\text{get}(n + m, \text{append}(\mathit{lst1}, \mathit{lst2})) = \text{get}(m - 1, \mathit{lst2}))$

Theorem: get-add1-plus
$\text{get}((1 + n) + m, \text{cons}(x, y)) = \text{get}(n + m, y)$

Theorem: get-0-plus
$\text{get}(0 + n, y) = \text{get}(n, y)$

Definition:
$\text{put}(\mathit{val}, n, \mathit{lst})$
= **if** $n \simeq 0$
  then if $\text{listp}(\mathit{lst})$ **then** $\text{cons}(\mathit{val}, \text{cdr}(\mathit{lst}))$
    else $\text{list}(\mathit{val})$ **endif**
  else $\text{cons}(\text{car}(\mathit{lst}), \text{put}(\mathit{val}, n - 1, \text{cdr}(\mathit{lst})))$ **endif**

THEOREM: put-car-nlistp
$((n \simeq 0) \land (lst \simeq \textbf{nil})) \rightarrow (\text{put}(val, n, lst) = \text{list}(val))$

THEOREM: put-car-listp
$((n \simeq 0) \land \text{listp}(lst)) \rightarrow (\text{put}(val, n, lst) = \text{cons}(val, \text{cdr}(lst)))$

THEOREM: put-preserves-plistp
$\text{plistp}(lst) \rightarrow \text{plistp}(\text{put}(val, n, lst))$

THEOREM: get-inverts-put
$\text{get}(y, \text{put}(x, y, z)) = x$

DEFINITION:
put-assoc (val, $name$, $alist$)
= **if** $alist \simeq$ **nil** **then** $alist$
elseif $name = \text{caar}(alist)$ **then** cons (cons ($name$, val), cdr ($alist$))
else cons (car ($alist$), put-assoc (val, $name$, cdr ($alist$))) **endif**

THEOREM: put-assoc-expansion
$\text{put-assoc}(x, y, \text{cons}(\text{cons}(y, z), w)) = \text{cons}(\text{cons}(y, x), w)$

DEFINITION:
definedp ($name$, $alist$)
= **if** $alist \simeq$ **nil** **then f**
elseif $name = \text{caar}(alist)$ **then t**
else definedp ($name$, cdr ($alist$)) **endif**

THEOREM: member-defined-name
$(x \in y) \rightarrow \text{definedp}(\text{car}(x), y)$

THEOREM: definedp-car-assoc
$\text{definedp}(x, lst) \rightarrow (\text{car}(\text{assoc}(x, lst)) = x)$

THEOREM: definedp-car-cons
$\text{definedp}(x, \text{cons}(\text{cons}(x, y), z))$

THEOREM: definedp-implies-member
$(x \in \text{listcars}(y)) = \text{definedp}(x, y)$

EVENT: Disable definedp-implies-member.

THEOREM: definedp-member-assoc
$\text{definedp}(x, lst) \rightarrow (\text{assoc}(x, lst) \in lst)$

THEOREM: definedp-rewrites-to-member
$\text{definedp}(x, lst) = (x \in \text{listcars}(lst))$

EVENT: Disable definedp-rewrites-to-member.

THEOREM: definedp-distributes
$\text{definedp}(x, \text{append}(y, z)) = (\text{definedp}(x, y) \lor \text{definedp}(x, z))$

THEOREM: definedp-append-preserves-assoc
$\text{definedp}(x, y) \rightarrow (\text{assoc}(x, \text{append}(y, z)) = \text{assoc}(x, y))$

EVENT: Disable definedp-append-preserves-assoc.

THEOREM: definedp-once
$(\text{all-cars-unique}(\text{append}(y, z)) \land \text{definedp}(x, z)) \rightarrow (\neg\ \text{definedp}(x, y))$

EVENT: Disable definedp-once.

THEOREM: not-definedp-assoc-append
$(\neg\ \text{definedp}(x, y)) \rightarrow (\text{assoc}(x, \text{append}(y, z)) = \text{assoc}(x, z))$

THEOREM: put-preserves-length
$(y < \text{length}(z)) \rightarrow (\text{length}(\text{put}(x, y, z)) = \text{length}(z))$

THEOREM: multiple-puts-cancel
$\text{put}(x, y, \text{put}(z, y, w)) = \text{put}(x, y, w)$

THEOREM: puts-commute
$$\begin{array}{ll} ((y \in \mathbf{N}) & \\ \land & (v \in \mathbf{N}) \\ \land & (y < \text{length}(z)) \\ \land & (v < \text{length}(z)) \\ \land & (v \neq y)) \\ \rightarrow & (\text{put}(x, y, \text{put}(u, v, z)) = \text{put}(u, v, \text{put}(x, y, z))) \end{array}$$

EVENT: Disable puts-commute.

THEOREM: put-never-shrinks
$(\text{length}(\text{put}(x, y, z)) < \text{length}(z)) = \mathbf{f}$

THEOREM: put-value-length-rewrite
$$\begin{array}{ll} (n = \text{length}(\mathit{lst1})) & \\ \rightarrow & (\text{put}(\mathit{value}, n, \text{append}(\mathit{lst1}, \text{cons}(\mathit{value2}, \mathit{lst2}))) \\ & \quad = \quad \text{append}(\mathit{lst1}, \text{cons}(\mathit{value}, \mathit{lst2}))) \end{array}$$

EVENT: Disable put-value-length-rewrite.

THEOREM: put-length
$(n = \text{length}(lst))$
$\rightarrow \quad (\text{put}(x,\ n,\ \text{append}(lst,\ \text{cons}(y,\ lst2))) = \text{append}(lst,\ \text{cons}(x,\ lst2)))$

EVENT: Disable put-length.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                             ;;
;;                         RESTRICT                            ;;
;;                                                             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Given an alist, select the sub-alist with names in a given list.
;; This will allow me to select out, for example, the sub-alist relating to
;; the locals or formals.
```

DEFINITION:
restrict (*alist*, *names*)
= **if** *alist* $\simeq$ **nil then nil**
elseif caar (*alist*) $\in$ *names*
then cons (car (*alist*), restrict (cdr (*alist*), *names*))
else restrict (cdr (*alist*), *names*) **endif**

THEOREM: restriction-nil-names
$(names \simeq \textbf{nil}) \rightarrow (\text{restrict}(alist,\ names) = \textbf{nil})$

THEOREM: no-duplicates-restriction
$\text{no-duplicates}(\text{cons}(x,\ \text{listcars}(alist)))$
$\rightarrow \quad (\text{restrict}(alist,\ \text{cons}(x,\ names)) = \text{restrict}(alist,\ names))$

EVENT: Disable no-duplicates-restriction.

THEOREM: restriction-names-cdr
$(\text{no-duplicates}(names)$
$\land \quad \text{no-duplicates}(\text{listcars}(alist))$
$\land \quad (\text{caar}(alist) = \text{car}(names)))$
$\rightarrow \quad (\text{restrict}(\text{cdr}(alist),\ names) = \text{restrict}(\text{cdr}(alist),\ \text{cdr}(names)))$

EVENT: Disable restriction-names-cdr.

THEOREM: restrict-alist-listcars
$(\text{plistp}(alist) \land \text{no-duplicates}(\text{listcars}(alist)))$
$\rightarrow \quad (\text{restrict}(alist,\ \text{listcars}(alist)) = alist)$

EVENT: Disable restrict-alist-listcars.

THEOREM: restrict-listcars-member
$(x \notin y) \rightarrow (x \notin \text{listcars}(\text{restrict}(lst, y)))$

DEFINITION:
restriction-induction-hint (*alist*, *names1*, *names2*)
= **if** *alist* $\simeq$ **nil** **then t**
elseif caar (*alist*) $\in$ *names1*
then restriction-induction-hint (cdr (*alist*), cdr (*names1*), *names2*)
else restriction-induction-hint (cdr (*alist*), *names1*, cdr (*names2*)) **endif**

THEOREM: listcars-restriction-append
((listcars (*alist*) = append (*names1*, *names2*))
$\wedge$ no-duplicates (listcars (*alist*))
$\wedge$ plistp (*alist*))
$\rightarrow$ (append (restrict (*alist*, *names1*), restrict (*alist*, *names2*)) = *alist*)

EVENT: Disable listcars-restriction-append.

THEOREM: restriction-plistp
plistp (restrict (x, y))

THEOREM: assoc-restriction
$(x \in y) \rightarrow (\text{assoc}(x, \text{restrict}(z, y)) = \text{assoc}(x, z))$

EVENT: Disable assoc-restriction.

THEOREM: no-duplicates-append-restrict
no-duplicates (append (*names*, listcars (*lst*)))
$\rightarrow$ (restrict (append (x, *lst*), *names*) = restrict (x, *names*))

EVENT: Disable no-duplicates-append-restrict.

DEFINITION:
double-cdr-induction (x, y)
= **if** $x \simeq$ **nil** **then t**
else double-cdr-induction (cdr (x), cdr (y)) **endif**

THEOREM: restrict-matching-listcars
(plistp (*lst*) $\wedge$ (*names* = listcars (*lst*)) $\wedge$ no-duplicates (listcars (*lst*)))
$\rightarrow$ (restrict (*lst*, *names*) = *lst*)

EVENT: Disable restrict-matching-listcars.

THEOREM: restrict-append2
no-duplicates (append (listcars (*lst1*), *names*))
$\rightarrow$ (restrict (append (*lst1*, *lst2*), *names*) = restrict (*lst2*, *names*))

EVENT: Disable restrict-append2.

THEOREM: restrict-restricts-listcars
$(x \notin$ listcars $(y)) \rightarrow (x \notin$ listcars (restrict $(y, z)))$

THEOREM: restriction-cdr
(listp (y) $\land$ (car (y) $\notin$ listcars (x)))
$\rightarrow$ (restrict (x, y) = restrict $(x$, cdr $(y)))$

EVENT: Disable restriction-cdr.

THEOREM: restrict-append
(listp (y) $\land$ (listcars (x) = append (y, z)) $\land$ all-cars-unique (x))
$\rightarrow$ (restrict (x, y) = cons (car (x), restrict (cdr (x), cdr (y))))

EVENT: Disable restrict-append.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                   ;;
;;                           SIGNATURES                              ;;
;;                                                                   ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
signatures-match (*alist1*, *alist2*)
= **if** *alist1* $\simeq$ **nil then** *alist2* = **nil**
  else (caar (*alist1*) = caar (*alist2*))
    $\land$ (cadr (car (*alist1*)) = cadr (car (*alist2*)))
    $\land$ signatures-match (cdr (*alist1*), cdr (*alist2*)) **endif**

THEOREM: signatures-match-reflexive
plistp (*lst*) $\rightarrow$ signatures-match (*lst*, *lst*)

THEOREM: signatures-match-reflexive1
signatures-match (*lst*, *lst*) = plistp (*lst*)

THEOREM: signatures-match-symmetric
(plistp (*lst1*) $\land$ signatures-match (*lst1*, *lst2*))
$\rightarrow$ signatures-match (*lst2*, *lst1*)

EVENT: Disable signatures-match-symmetric.

THEOREM: signatures-match-transitive
(plistp (*lst1*)
 $\land$ signatures-match (*lst1*, *lst2*)
 $\land$ signatures-match (*lst2*, *lst3*))
$\rightarrow$ signatures-match (*lst1*, *lst3*)

EVENT: Disable signatures-match-transitive.

THEOREM: signatures-match-listp
(signatures-match (x, y) $\land$ listp (x)) $\rightarrow$ listp (y)

EVENT: Disable signatures-match-listp.

THEOREM: signatures-match-listcars
signatures-match (x, y) $\rightarrow$ (listcars (y) = listcars (x))

EVENT: Disable signatures-match-listcars.

THEOREM: signatures-match-append1
(signatures-match (*lst1*, *lst3*) $\land$ signatures-match (*lst2*, *lst4*))
$\rightarrow$ signatures-match (append (*lst1*, *lst2*), append (*lst3*, *lst4*))

EVENT: Disable signatures-match-append1.

THEOREM: signatures-match-reorder
(plistp (*alist1*)
 $\land$ signatures-match (*alist1*, *alist3*)
 $\land$ signatures-match (*alist1*, *alist2*))
$\rightarrow$ signatures-match (*alist2*, *alist3*)

EVENT: Disable signatures-match-reorder.

DEFINITION:
signature (*mg-vars-list*)
= **if** *mg-vars-list* $\simeq$ **nil then nil**
 else cons (list (caar (*mg-vars-list*), cadar (*mg-vars-list*)),
 signature (cdr (*mg-vars-list*))) **endif**

THEOREM: signatures-match-listcars-equal
signatures-match (x, y) $\rightarrow$ (listcars (y) = listcars (x))

Event: Disable signatures-match-listcars-equal.

Theorem: signatures-match-preserves-uniqueness-of-cars
(signatures-match $(x,\ y)$ ∧ all-cars-unique (x)) → all-cars-unique (y)

Event: Disable signatures-match-preserves-uniqueness-of-cars.

Theorem: signature-restrict-commute
signature (restrict (*alist*, *names*)) = restrict (signature (*alist*), *names*)

Event: Disable signature-restrict-commute.

Theorem: signatures-match-restrict
signatures-match $(x,\ y)$ → signatures-match (restrict $(x,\ z)$, restrict $(y,\ z)$)

Event: Disable signatures-match-restrict.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                             ;;
;;                          DISJOINT                           ;;
;;                                                             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Definition:
one-way-disjoint (*lst1*, *lst2*)
= **if** *lst1* ≃ **nil then t**
else (car (*lst1*) ∉ *lst2*) ∧ one-way-disjoint (cdr (*lst1*), *lst2*) **endif**

Definition:
disjoint (*lst1*, *lst2*)
= (one-way-disjoint (*lst1*, *lst2*) ∧ one-way-disjoint (*lst2*, *lst1*))

Theorem: disjoint-nil
disjoint (x, **nil**) ∧ disjoint (**nil**, x)

Theorem: cons-preserves-one-way-disjoint2
one-way-disjoint (*lst2*, cons (x, *lst*)) → one-way-disjoint (*lst2*, *lst*)

Theorem: cdr-preserves-disjoint
disjoint (cons (x, *lst*), *lst2*) → disjoint (*lst*, *lst2*)

Theorem: no-duplicates-append-implies-one-way-disjoint
no-duplicates (append $(x,\ y)$) → one-way-disjoint $(x,\ y)$

EVENT: Disable no-duplicates-append-implies-one-way-disjoint.

THEOREM: right-cdr-preserves-one-way-disjoint
(one-way-disjoint $(x,\ y) \land$ listp $(y)) \rightarrow$ one-way-disjoint $(x,$ cdr $(y))$

EVENT: Disable right-cdr-preserves-one-way-disjoint.

THEOREM: disjoint-preserves-no-duplicates
(no-duplicates (*lst1*) $\land$ no-duplicates (*lst2*) $\land$ disjoint (*lst1*, *lst2*))
$\rightarrow$ no-duplicates (append (*lst1*, *lst2*))

EVENT: Disable disjoint-preserves-no-duplicates.

THEOREM: one-way-disjoint-right-cons
(one-way-disjoint (*lst1*, *lst2*) $\land$ ($x \notin$ *lst1*))
$\rightarrow$ one-way-disjoint (*lst1*, cons (x, *lst2*))

EVENT: Disable one-way-disjoint-right-cons.

THEOREM: disjoint-right-cons
(disjoint (*lst1*, *lst2*) $\land$ ($x \notin$ *lst1*)) $\rightarrow$ disjoint (*lst1*, cons (x, *lst2*))

EVENT: Disable disjoint-right-cons.

THEOREM: one-way-disjoint-right-cdr
(listp (*lst2*) $\land$ one-way-disjoint (*lst1*, *lst2*))
$\rightarrow$ one-way-disjoint (*lst1*, cdr (*lst2*))

EVENT: Disable one-way-disjoint-right-cdr.

THEOREM: disjoint-right-cdr
(listp (*lst2*) $\land$ disjoint (*lst1*, *lst2*)) $\rightarrow$ disjoint (*lst1*, cdr (*lst2*))

EVENT: Disable disjoint-right-cdr.

THEOREM: disjoint-right-append
(disjoint (*lst1*, *lst2*) $\land$ disjoint (*lst1*, *lst3*))
$\rightarrow$ disjoint (*lst1*, append (*lst2*, *lst3*))

EVENT: Disable disjoint-right-append.

EVENT: Disable disjoint.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                  ;;
;;                         COND-SUBSETP                             ;;
;;                                                                  ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

DEFINITION:
cond-subsetp (*lst1*, *lst2*)
= **if** *lst1* $\simeq$ **nil** **then t**
else (car (*lst1*) $\in$ cons (`'leave`, cons (`'routineerror`, *lst2*)))
$\wedge$ cond-subsetp (cdr (*lst1*), *lst2*) **endif**

THEOREM: cond-subsetp-append
(cond-subsetp $(y,\ z)$ $\wedge$ cond-subsetp $(x,\ y)$) $\rightarrow$ cond-subsetp (append $(x,\ y)$, z)

EVENT: Disable cond-subsetp-append.

THEOREM: subsetp-implies-cond-subsetp
subset $(x,\ y)$ $\rightarrow$ cond-subsetp $(x,\ y)$

EVENT: Disable subsetp-implies-cond-subsetp.

EVENT: Make the library `"c1"`.

Index