EVENT: Start with the library "naturals".

```
;;; Several Useful Theorems
```

THEOREM: equal-iff
$((\text{truep}\,(a) \vee \text{falsep}\,(a)) \wedge (\text{truep}\,(b) \vee \text{falsep}\,(b)))$
$\rightarrow \quad ((a = b) = (a \leftrightarrow b))$

EVENT: Disable equal-iff.

DEFINITION:
length (*list*)
= **if** listp (*list*) **then** 1 + length (cdr (*list*))
else 0 **endif**

DEFINITION:
nth (*list*, *n*)
= **if** $n \simeq 0$ **then** car (*list*)
else nth (cdr (*list*), $n - 1$) **endif**

DEFINITION:
position (*prg*, *e*)
= **if** listp (*prg*)
then if car (*prg*) = *e* **then** 0
else 1 + position (cdr (*prg*), *e*) **endif**
else 0 **endif**

THEOREM: nth-member
$(n < \text{length}\,(list)) \rightarrow (\text{nth}\,(list, n) \in list)$

THEOREM: listp-not-zero-length
$(\text{length}\,(list) = 0) = (\neg\ \text{listp}\,(list))$

THEOREM: position-zero
$(e \notin prg) \rightarrow (\text{position}\,(prg, e) = \text{length}\,(prg))$

THEOREM: position-lessp
$(e \in prg) \rightarrow (\text{position}\,(prg, e) < \text{length}\,(prg))$

THEOREM: nth-position
nth (*prg*, position (*prg*, *e*))
= **if** $e \in prg$ **then** *e*
else 0 **endif**

THEOREM: position-nth
$((e \in \mathit{list}) \wedge (\mathrm{nth}\,(\mathit{list},\ n) \neq e)) \rightarrow ((\mathrm{position}\,(\mathit{list},\ e) = n) = \mathbf{f})$

THEOREM: member-append
$(e \in \mathrm{append}\,(j,\ k)) = ((e \in j) \vee (e \in k))$

THEOREM: append-is-associative
$\mathrm{append}\,(\mathrm{append}\,(x,\ y),\ z) = \mathrm{append}\,(x,\ \mathrm{append}\,(y,\ z))$

```
;;; The Witness function for the Computation
```

DEFINITION: $\mathrm{mchoose}\,(\mathit{prg},\ i) = \mathrm{nth}\,(\mathit{prg},\ i\ \mathbf{mod}\ \mathrm{length}\,(\mathit{prg}))$

DEFINITION:
$\mathrm{mnext}\,(\mathit{prg},\ e,\ i)$
$=\ \ (i + \mathbf{if}\ \mathrm{position}\,(\mathit{prg},\ e) < (i\ \mathbf{mod}\ \mathrm{length}\,(\mathit{prg}))$
$\quad \mathbf{then}\ \mathrm{position}\,(\mathit{prg},\ e)$
$\quad\quad +\ \ (\mathrm{length}\,(\mathit{prg}) - (i\ \mathbf{mod}\ \mathrm{length}\,(\mathit{prg})))$
$\quad \mathbf{else}\ \mathrm{position}\,(\mathit{prg},\ e) - (i\ \mathbf{mod}\ \mathrm{length}\,(\mathit{prg}))\ \mathbf{endif})$

THEOREM: mnext-fixes
$((e \in \mathit{prg}) \wedge (i \notin \mathbf{N})) \rightarrow (\mathrm{mnext}\,(\mathit{prg},\ e,\ i) = \mathrm{mnext}\,(\mathit{prg},\ e,\ 0))$

THEOREM: numberp-mnext
$(e \in \mathit{prg}) \rightarrow (\mathrm{mnext}\,(\mathit{prg},\ e,\ i) \in \mathbf{N})$

THEOREM: mchoose-chooses
$\mathrm{listp}\,(\mathit{prg}) \rightarrow (\mathrm{mchoose}\,(\mathit{prg},\ i) \in \mathit{prg})$

THEOREM: mchoose-fixes
$(\mathrm{listp}\,(\mathit{prg}) \wedge (i \notin \mathbf{N})) \rightarrow (\mathrm{mchoose}\,(\mathit{prg},\ i) = \mathrm{mchoose}\,(\mathit{prg},\ 0))$

THEOREM: mnext-choice-1
$(e \in \mathit{prg}) \rightarrow (\mathrm{mnext}\,(\mathit{prg},\ e,\ i) \not< i)$

THEOREM: mnext-choice-2-simplified
$(e < n)$
$\rightarrow\ \ (((i + \mathbf{if}\ e < (i\ \mathbf{mod}\ n)\ \ \mathbf{then}\ e + (n - (i\ \mathbf{mod}\ n))$
$\quad\quad\quad \mathbf{else}\ e - (i\ \mathbf{mod}\ n)\ \mathbf{endif})$
$\quad\ \ \mathbf{mod}\ \ \ n)$
$\quad =\ \ \mathrm{fix}\,(e))$

THEOREM: mnext-choice-2
$(e \in \mathit{prg}) \rightarrow (\mathrm{mchoose}\,(\mathit{prg},\ \mathrm{mnext}\,(\mathit{prg},\ e,\ i)) = e)$

THEOREM: remainder-of-add1
$((x < (a \bmod b)) \wedge (x < ((1 + a) \bmod b)))$
$\rightarrow \quad (((1 + a) \bmod b) = (1 + (a \bmod b)))$

THEOREM: remainder-of-add1-1
$((x \not< (a \bmod b)) \wedge (x < ((1 + a) \bmod b))) \rightarrow ((a \bmod b) = \text{fix}(x))$

THEOREM: remainder-of-add1-2
$((x < (a \bmod b)) \wedge (x \not< ((1 + a) \bmod b)))$
$\rightarrow \quad ((((1 + a) \bmod b) = \texttt{0}) \wedge ((a \bmod b) = (b - 1)))$

THEOREM: remainder-of-add1-3
$((x \not< (a \bmod b))$
$\wedge \quad (x < b)$
$\wedge \quad (x \in \mathbf{N})$
$\wedge \quad (x \neq (a \bmod b))$
$\wedge \quad (x \not< ((1 + a) \bmod b)))$
$\rightarrow \quad (((1 + a) \bmod b) = (1 + (a \bmod b)))$

THEOREM: remainder-of-add1-3-1
$((\text{position}(prg, e) \not< (i \bmod \text{length}(prg)))$
$\wedge \quad (\text{position}(prg, e) \not< ((1 + i) \bmod \text{length}(prg)))$
$\wedge \quad (\text{nth}(prg, i \bmod \text{length}(prg)) \neq e)$
$\wedge \quad (e \in prg))$
$\rightarrow \quad (((1 + i) \bmod \text{length}(prg)) = (1 + (i \bmod \text{length}(prg))))$

EVENT: Disable mchoose.

EVENT: Disable mnext.

```
;;; The Statement Interpreter
```

DEFINITION:
$\text{n}(old, new, e) = \text{apply\$}(\text{car}(e), \text{append}(\text{list}(old, new), \text{cdr}(e)))$

DEFINITION: $\text{exists-successor}(old, e) \leftrightarrow \exists\, new\ \text{n}(old, new, e)$

THEOREM: exists-successor-implies
$\text{exists-successor}(old, e) \rightarrow \text{n}(old, \text{newx}(e, old), e)$

THEOREM: prove-exists-successor
$\text{n}(old, new, e) \rightarrow \text{exists-successor}(old, e)$

EVENT: Disable exists-successor.

DEFINITION:
ms (*prg*, *i*)
= **if** *i* ≃ 0 **then nil**
elseif n (ms (*prg*, *i* − 1),
newx (mchoose (*prg*, *i* − 1), ms (*prg*, *i* − 1)),
mchoose (*prg*, *i* − 1))
then newx (mchoose (*prg*, *i* − 1), ms (*prg*, *i* − 1))
else ms (*prg*, *i* − 1) **endif**

THEOREM: ms-transition-successful
(listp (*prg*) ∧ n (ms (*prg*, *i*), *new*, mchoose (*prg*, *i*)))
→ n (ms (*prg*, *i*), ms (*prg*, 1 + *i*), mchoose (*prg*, *i*))

THEOREM: ms-transition-idle
(listp (*prg*)
∧ (¬ n (ms (*prg*, *i*), newx (mchoose (*prg*, *i*), ms (*prg*, *i*)), mchoose (*prg*, *i*))))
→ (ms (*prg*, 1 + *i*) = ms (*prg*, *i*))

```
;;; Characterizing an Arbitrary Computation
```

CONSERVATIVE AXIOM: computation
(listp (*prg*) → (choose (*prg*, *i*) ∈ *prg*))
∧ ((*e* ∈ *prg*) → (next (*prg*, *e*, *i*) ≮ *i*))
∧ ((*e* ∈ *prg*) → (choose (*prg*, next (*prg*, *e*, *i*)) = *e*))
∧ ((listp (*prg*) ∧ n (s (*prg*, *i*), *new*, choose (*prg*, *i*)))
→ n (s (*prg*, *i*), s (*prg*, 1 + *i*), choose (*prg*, *i*)))
∧ ((listp (*prg*)
∧ (¬ n (s (*prg*, *i*), newx (choose (*prg*, *i*), s (*prg*, *i*)), choose (*prg*, *i*))))
→ (s (*prg*, 1 + *i*) = s (*prg*, *i*)))
∧ ((*e* ∈ *prg*) → (next (*prg*, *e*, *i*) ∈ **N**))
∧ ((listp (*prg*) ∧ (*i* ∉ **N**)) → (s (*prg*, *i*) = s (*prg*, 0)))
∧ ((listp (*prg*) ∧ (*i* ∉ **N**)) → (choose (*prg*, *i*) = choose (*prg*, 0)))
∧ (((*e* ∈ *prg*) ∧ (*i* ∉ **N**)) → (next (*prg*, *e*, *i*) = next (*prg*, *e*, 0)))

Simultaneously, we introduce the new function symbols *choose*, *next*, and *s*.
EVENT: Disable mnext-fixes.

EVENT: Disable numberp-mnext.

EVENT: Disable mchoose-chooses.

EVENT: Disable mchoose-fixes.

EVENT: Disable mnext-choice-1.

EVENT: Disable mnext-choice-2-simplified.

EVENT: Disable mnext-choice-2.

EVENT: Disable remainder-of-add1.

EVENT: Disable remainder-of-add1-1.

EVENT: Disable remainder-of-add1-2.

EVENT: Disable remainder-of-add1-3.

EVENT: Disable remainder-of-add1-3-1.

EVENT: Disable exists-successor-implies.

EVENT: Disable prove-exists-successor.

EVENT: Disable ms-transition-successful.

EVENT: Disable ms-transition-idle.

EVENT: Disable n.

```
;;; The Predicate Interpreter
```

DEFINITION:
eval (*pred*, *state*) = eval$ (**t**, *pred*, list (cons (’state, *state*)))

THEOREM: eval-not
eval (list (’not, *p*), *state*) = (¬ eval (*p*, *state*))

THEOREM: eval-and
eval (list (’and, *p*, *q*), *state*) = (eval (*p*, *state*) ∧ eval (*q*, *state*))

THEOREM: eval-or
$\text{eval}\,(\text{list}\,(\texttt{'or},\ p,\ q),\ \mathit{state}) = (\text{eval}\,(p,\ \mathit{state}) \vee \text{eval}\,(q,\ \mathit{state}))$

THEOREM: eval-implies
$\text{eval}\,(\text{list}\,(\texttt{'implies},\ p,\ q),\ \mathit{state}) = (\text{eval}\,(p,\ \mathit{state}) \rightarrow \text{eval}\,(q,\ \mathit{state}))$

THEOREM: eval-iff
$\text{eval}\,(\text{list}\,(\texttt{'iff},\ p,\ q),\ \mathit{state}) = (\text{eval}\,(p,\ \mathit{state}) \leftrightarrow \text{eval}\,(q,\ \mathit{state}))$

THEOREM: eval-equal
$\text{eval}\,(\text{list}\,(\texttt{'equal},\ p,\ q),\ \mathit{state}) = (\text{eval}\,(p,\ \mathit{state}) = \text{eval}\,(q,\ \mathit{state}))$

THEOREM: eval-true
$\text{eval}\,(\texttt{'(true)},\ \mathit{state}) = \mathbf{t}$

THEOREM: eval-false
$\text{eval}\,(\texttt{'(false)},\ \mathit{state}) = \mathbf{f}$

EVENT: Disable eval.

```
;;; The Stability Operator:  Unless
```

DEFINITION:
$$\begin{aligned}
&\text{unless}\,(p,\ q,\ \mathit{prg})\\
\leftrightarrow\quad &\forall\ \mathit{old},\ \mathit{new},\ e\ (((e \in \mathit{prg})\\
&\qquad\qquad \wedge\quad (\text{n}\,(\mathit{old},\ \mathit{new},\ e)\\
&\qquad\qquad\qquad \wedge\quad \text{eval}\,(\text{list}\,(\texttt{'and},\ p,\ \text{list}\,(\texttt{'not},\ q)),\ \mathit{old})))\\
&\qquad\qquad \rightarrow\quad \text{eval}\,(\text{list}\,(\texttt{'or},\ p,\ q),\ \mathit{new}))
\end{aligned}$$

EVENT: Disable unless.

THEOREM: prove-unless
$$\begin{aligned}
&(((\text{eu}\,(p,\ \mathit{prg},\ q) \in \mathit{prg})\\
&\quad \wedge\quad \text{n}\,(\text{oldu}\,(p,\ \mathit{prg},\ q),\ \text{newu}\,(p,\ \mathit{prg},\ q),\ \text{eu}\,(p,\ \mathit{prg},\ q))\\
&\quad \wedge\quad \text{eval}\,(p,\ \text{oldu}\,(p,\ \mathit{prg},\ q))\\
&\quad \wedge\quad (\neg\ \text{eval}\,(q,\ \text{oldu}\,(p,\ \mathit{prg},\ q))))\\
&\quad \rightarrow\quad (\text{eval}\,(p,\ \text{newu}\,(p,\ \mathit{prg},\ q)) \vee \text{eval}\,(q,\ \text{newu}\,(p,\ \mathit{prg},\ q))))\\
&\rightarrow\quad \text{unless}\,(p,\ q,\ \mathit{prg})
\end{aligned}$$

THEOREM: unless-implies
$$\begin{aligned}
&(\text{unless}\,(p,\ q,\ \mathit{prg})\\
&\quad \wedge\quad (e \in \mathit{prg})\\
&\quad \wedge\quad \text{n}\,(\mathit{old},\ \mathit{new},\ e)\\
&\quad \wedge\quad \text{eval}\,(p,\ \mathit{old})\\
&\quad \wedge\quad (\neg\ \text{eval}\,(q,\ \mathit{old})))\\
&\rightarrow\quad \text{eval}\,(\text{list}\,(\texttt{'or},\ p,\ q),\ \mathit{new})
\end{aligned}$$

EVENT: Disable prove-unless.

EVENT: Disable unless-implies.

```
;;; The Progress Operator:  Leads-To
```

DEFINITION:
leads-to $(p,\ q,\ prg)$
$\leftrightarrow \quad \forall\, i\ (\text{eval}\,(p,\ \text{s}\,(prg,\ i)) \rightarrow \exists\, j\ ((j \not< i) \wedge \text{eval}\,(q,\ \text{s}\,(prg,\ j))))$

EVENT: Disable leads-to.

THEOREM: prove-leads-to
$(\text{eval}\,(p,\ \text{s}\,(prg,\ \text{ileads}\,(p,\ prg,\ q)))$
$\rightarrow \quad ((j \not< \text{ileads}\,(p,\ prg,\ q)) \wedge \text{eval}\,(q,\ \text{s}\,(prg,\ j))))$
$\rightarrow \quad \text{leads-to}\,(p,\ q,\ prg)$

THEOREM: leads-to-implies
$(\text{leads-to}\,(p,\ q,\ prg) \wedge \text{eval}\,(p,\ \text{s}\,(prg,\ i)))$
$\rightarrow \quad ((\text{jleads}\,(i,\ prg,\ q) \not< i) \wedge \text{eval}\,(q,\ \text{s}\,(prg,\ \text{jleads}\,(i,\ prg,\ q))))$

EVENT: Disable prove-leads-to.

EVENT: Disable leads-to-implies.

```
;;; The Most Effective Trace
```

DEFINITION:
schedulable $(prg) \leftrightarrow \forall\, i\ \exists\, new\ \text{n}\,(\text{s}\,(prg,\ i),\ new,\ \text{choose}\,(prg,\ i))$

EVENT: Disable schedulable.

THEOREM: prove-schedulable
$\text{n}\,(\text{s}\,(prg,\ \text{is}\,(prg)),\ new,\ \text{choose}\,(prg,\ \text{is}\,(prg))) \rightarrow \text{schedulable}\,(prg)$

THEOREM: schedulable-implies
$\text{schedulable}\,(prg) \rightarrow \text{n}\,(\text{s}\,(prg,\ i),\ \text{news}\,(i,\ prg),\ \text{choose}\,(prg,\ i))$

EVENT: Disable prove-schedulable.

EVENT: Disable schedulable-implies.

THEOREM: schedulable-implies-effective-computation
(schedulable (*prg*) ∧ listp (*prg*))
→ n (s (*prg*, *i*), s (*prg*, 1 + *i*), choose (*prg*, *i*))

THEOREM: computation-n
(schedulable (*prg*) ∧ (*e* ∈ *prg*))
→ n (s (*prg*, next (*prg*, *e*, *i*)), s (*prg*, 1 + next (*prg*, *e*, *i*)), *e*)

THEOREM: effective-idle
listp (*prg*)
→ ((s (*prg*, 1 + *i*) = s (*prg*, *i*))
∨ n (s (*prg*, *i*), s (*prg*, 1 + *i*), choose (*prg*, *i*)))

EVENT: Disable effective-idle.

```
;;; Leads-To Proof Rules
```

THEOREM: leads-to-transitive
(leads-to (*p*, *q*, *prg*) ∧ leads-to (*q*, *r*, *prg*)) → leads-to (*p*, *r*, *prg*)

EVENT: Disable leads-to-transitive.

THEOREM: leads-to-transitive-general
(leads-to (*p-1*, *q-1*, *prg*)
∧ leads-to (*q-2*, *r-1*, *prg*)
∧ (eval (*q-1*, s (*prg*, jleads (ileads (*p*, *prg*, *r*), *prg*, *q-1*)))
→ eval (*q-2*, s (*prg*, jleads (ileads (*p*, *prg*, *r*), *prg*, *q-1*))))
∧ (eval (*p*, s (*prg*, ileads (*p*, *prg*, *r*))) → eval (*p-1*, s (*prg*, ileads (*p*, *prg*, *r*))))
∧ (eval (*r-1*, s (*prg*, jleads (jleads (ileads (*p*, *prg*, *r*), *prg*, *q-1*), *prg*, *r-1*)))
→ eval (*r*,
s (*prg*, jleads (jleads (ileads (*p*, *prg*, *r*), *prg*, *q-1*), *prg*, *r-1*)))))
→ leads-to (*p*, *r*, *prg*)

EVENT: Disable leads-to-transitive-general.

THEOREM: q-leads-to-q
leads-to (*q*, *q*, *prg*)

EVENT: Disable q-leads-to-q.

THEOREM: false-leads-to-anything
leads-to (`'(false)`, *p*, *prg*)

EVENT: Disable false-leads-to-anything.

THEOREM: p-implies-q-leads-to
(eval (*p*, s (*prg*, ileads (*p*, *prg*, *q*))) → eval (*q*, s (*prg*, ileads (*p*, *prg*, *q*))))
→ leads-to (*p*, *q*, *prg*)

EVENT: Disable p-implies-q-leads-to.

THEOREM: leads-to-strengthen-left
((eval (*q*, s (*prg*, ileads (*q*, *prg*, *r*))) → eval (*p*, s (*prg*, ileads (*q*, *prg*, *r*))))
∧ leads-to (*p*, *r*, *prg*))
→ leads-to (*q*, *r*, *prg*)

EVENT: Disable leads-to-strengthen-left.

THEOREM: leads-to-weaken-right
((eval (*q*, s (*prg*, jleads (ileads (*p*, *prg*, *r*), *prg*, *q*)))
→ eval (*r*, s (*prg*, jleads (ileads (*p*, *prg*, *r*), *prg*, *q*))))
∧ leads-to (*p*, *q*, *prg*))
→ leads-to (*p*, *r*, *prg*)

EVENT: Disable leads-to-weaken-right.

THEOREM: leads-to-modify-both
((eval (*p*, s (*prg*, ileads (*p*, *prg*, *q*))) → eval (*p-1*, s (*prg*, ileads (*p*, *prg*, *q*))))
∧ (eval (*q-1*, s (*prg*, jleads (ileads (*p*, *prg*, *q*), *prg*, *q-1*)))
→ eval (*q*, s (*prg*, jleads (ileads (*p*, *prg*, *q*), *prg*, *q-1*))))
∧ leads-to (*p-1*, *q-1*, *prg*))
→ leads-to (*p*, *q*, *prg*)

EVENT: Disable leads-to-modify-both.

THEOREM: disjoin-left
(leads-to (*p*, *r*, *prg*) ∧ leads-to (*q*, *r*, *prg*))
→ leads-to (list (`'or`, *p*, *q*), *r*, *prg*)

EVENT: Disable disjoin-left.

THEOREM: disjoin-left-general
(leads-to (*p-1*, *q-1*, *prg*)
∧ leads-to (*p-2*, *q-2*, *prg*)
∧ (eval (*p*, s (*prg*, ileads (*p*, *prg*, *q*)))

$\rightarrow$ eval (list (`'or`, *p-1*, *p-2*), s (*prg*, ileads (*p*, *prg*, *q*))))
$\wedge$ (eval (*q-1*, s (*prg*, jleads (ileads (*p*, *prg*, *q*), *prg*, *q-1*)))
$\rightarrow$ eval (*q*, s (*prg*, jleads (ileads (*p*, *prg*, *q*), *prg*, *q-1*))))
$\wedge$ (eval (*q-2*, s (*prg*, jleads (ileads (*p*, *prg*, *q*), *prg*, *q-2*)))
$\rightarrow$ eval (*q*, s (*prg*, jleads (ileads (*p*, *prg*, *q*), *prg*, *q-2*)))))
$\rightarrow$ leads-to (*p*, *q*, *prg*)

EVENT: Disable disjoin-left-general.

THEOREM: cancellation-leads-to
(leads-to (*p*, list (`'or`, *q*, *b*), *prg*) $\wedge$ leads-to (*b*, *r*, *prg*))
$\rightarrow$ leads-to (*p*, list (`'or`, *q*, *r*), *prg*)

EVENT: Disable cancellation-leads-to.

THEOREM: cancellation-leads-to-general
(leads-to (*p-1*, *d*, *prg*)
$\wedge$ leads-to (*b*, *r-1*, *prg*)
$\wedge$ (eval (list (`'and`, *d*, list (`'not`, *b*)),
s (*prg*, jleads (ileads (*p*, *prg*, *r*), *prg*, *d*)))
$\rightarrow$ eval (*r*, s (*prg*, jleads (ileads (*p*, *prg*, *r*), *prg*, *d*))))
$\wedge$ (eval (*r-1*, s (*prg*, jleads (jleads (ileads (*p*, *prg*, *r*), *prg*, *d*), *prg*, *r-1*)))
$\rightarrow$ eval (*r*, s (*prg*, jleads (jleads (ileads (*p*, *prg*, *r*), *prg*, *d*), *prg*, *r-1*))))
$\wedge$ (eval (*p*, s (*prg*, ileads (*p*, *prg*, *r*))) $\rightarrow$ eval (*p-1*, s (*prg*, ileads (*p*, *prg*, *r*)))))
$\rightarrow$ leads-to (*p*, *r*, *prg*)

EVENT: Disable cancellation-leads-to-general.

DEFINITION:
ensures-interval (*prg*, *q*, *top*, *i*)
= **if** *top* $<$ *i* **then** fix (*top*)
elseif eval (*q*, s (*prg*, *i*)) **then** fix (*i*)
else ensures-interval (*prg*, *q*, *top*, 1 + *i*) **endif**

THEOREM: ensures-interval-fixes
(listp (*prg*) $\wedge$ (*i* $\notin$ **N**))
$\rightarrow$ (ensures-interval (*prg*, *q*, *top*, *i*) = ensures-interval (*prg*, *q*, *top*, 0))

THEOREM: ensures-interval-bigger
(ensures-interval (*prg*, *q*, *top*, *i*) $<$ *i*) = (*top* $<$ *i*)

THEOREM: psp-proves-something
(leads-to (*p*, *q*, *prg*)
$\wedge$ unless (*r*, *b*, *prg*)

$\wedge$ listp (prg)
$\wedge$ ($i \in \mathbf{N}$)
$\wedge$ eval (p, s (prg, $base$))
$\wedge$ eval (r, s (prg, i))
$\wedge$ (jleads ($base$, prg, q) $<$ top)
$\wedge$ (jleads ($base$, prg, q) $\not<$ i)
$\wedge$ ($final$ = list (`'or`, q, b)))
$\rightarrow$ eval (list (`'or`, list (`'and`, q, r), b),
s (prg, ensures-interval (prg, $final$, top, i)))

EVENT: Disable psp-proves-something.

THEOREM: psp
(leads-to (p, q, prg) $\wedge$ unless (r, b, prg) $\wedge$ listp (prg))
$\rightarrow$ leads-to (list (`'and`, p, r), list (`'or`, list (`'and`, q, r), b), prg)

EVENT: Disable psp.

THEOREM: psp-general
(leads-to (p, q, prg)
$\wedge$ unless (r, b, prg)
$\wedge$ (eval (pr, s (prg, ileads (pr, prg, qb)))
$\rightarrow$ eval (list (`'and`, p, r), s (prg, ileads (pr, prg, qb))))
$\wedge$ (eval (list (`'or`, list (`'and`, q, r), b),
s (prg,
jleads (ileads (pr, prg, qb), prg, list (`'or`, list (`'and`, q, r), b))))
$\rightarrow$ eval (qb,
s (prg,
jleads (ileads (pr, prg, qb),
prg,
list (`'or`, list (`'and`, q, r), b)))))
$\wedge$ listp (prg))
$\rightarrow$ leads-to (pr, qb, prg)

EVENT: Disable psp-general.

THEOREM: leads-to-true
leads-to (p, `'(true)`, prg)

```
;;; Initial Condition and Invariants
```

DEFINITION: initial-condition (ic, prg) = eval (ic, s (prg, `0`))

EVENT: Disable initial-condition.

DEFINITION: $\text{invariant}(inv, prg) \leftrightarrow \forall\, i\ \text{eval}(inv, \text{s}(prg, i))$

EVENT: Disable invariant.

THEOREM: prove-invariant
$\text{eval}(inv, \text{s}(prg, \text{ii}(inv, prg))) \rightarrow \text{invariant}(inv, prg)$

THEOREM: invariant-implies
$\text{invariant}(inv, prg) \rightarrow \text{eval}(inv, \text{s}(prg, i))$

EVENT: Disable prove-invariant.

THEOREM: invariant-consequence
$\text{invariant}(p, prg)$
$\wedge\quad (\text{eval}(p, \text{s}(prg, \text{ii}(q, prg))) \rightarrow \text{eval}(q, \text{s}(prg, \text{ii}(q, prg))))$
$\rightarrow\quad \text{invariant}(q, prg)$

THEOREM: invariants-persist-general
$(\text{unless}(p, \texttt{'(false)}, prg) \wedge \text{eval}(p, \text{s}(prg, i)) \wedge \text{listp}(prg) \wedge (j \not< i))$
$\rightarrow\quad \text{eval}(p, \text{s}(prg, j))$

EVENT: Disable invariants-persist-general.

THEOREM: unless-proves-invariant
$(\text{initial-condition}(ic, prg)$
$\wedge\quad \text{unless}(p, \texttt{'(false)}, prg)$
$\wedge\quad (\text{eval}(ic, \text{s}(prg, \texttt{0})) \rightarrow \text{eval}(p, \text{s}(prg, \texttt{0})))$
$\wedge\quad \text{listp}(prg))$
$\rightarrow\quad \text{invariant}(p, prg)$

THEOREM: leads-to-false-invariant
$(\text{leads-to}(p, \texttt{'(false)}, prg)$
$\wedge\quad (\text{eval}(\text{list}(\texttt{'not}, p), \text{s}(prg, \text{ii}(inv, prg)))$
$\qquad \rightarrow\quad \text{eval}(inv, \text{s}(prg, \text{ii}(inv, prg)))))$
$\rightarrow\quad \text{invariant}(inv, prg)$

```
;;; Eventual Stability
```

DEFINITION:
$\text{eventually-stable}(r, prg) \leftrightarrow \exists\, i\ \forall\, j\ ((j \not< i) \rightarrow \text{eval}(r, \text{s}(prg, j)))$

EVENT: Disable eventually-stable.

THEOREM: prove-eventually-stable
((jes (*i*, *prg*, *r*) $\not<$ *i*) $\rightarrow$ eval (*r*, s (*prg*, jes (*i*, *prg*, *r*))))
$\rightarrow$ eventually-stable (*r*, *prg*)

THEOREM: eventually-stable-implies
(eventually-stable (*r*, *prg*) $\wedge$ (*j* $\not<$ ies (*prg*, *r*))) $\rightarrow$ eval (*r*, s (*prg*, *j*))

EVENT: Disable prove-eventually-stable.

EVENT: Disable eventually-stable-implies.

THEOREM: not-leads-to-proves-eventually-stable
((¬ leads-to (*p*, *not-r*, *prg*))
$\wedge$ ((¬ eval (*not-r*, s (*prg*, jes (ileads (*p*, *prg*, *not-r*), *prg*, *r*))))
$\rightarrow$ eval (*r*, s (*prg*, jes (ileads (*p*, *prg*, *not-r*), *prg*, *r*)))))
$\rightarrow$ eventually-stable (*r*, *prg*)

EVENT: Disable not-leads-to-proves-eventually-stable.

THEOREM: not-eventually-stable-proves-leads-to
((¬ eventually-stable (*not-q*, *prg*))
$\wedge$ ((¬ eval (*not-q*, s (*prg*, jes (ileads (*p*, *prg*, *q*), *prg*, *not-q*))))
$\rightarrow$ eval (*q*, s (*prg*, jes (ileads (*p*, *prg*, *q*), *prg*, *not-q*)))))
$\rightarrow$ leads-to (*p*, *q*, *prg*)

EVENT: Disable not-eventually-stable-proves-leads-to.

THEOREM: true-leads-to-proves-not-eventually-stable
(leads-to (`'(true)`, *not-r*, *prg*)
$\wedge$ (eval (*not-r*, s (*prg*, jleads (ies (*prg*, *r*), *prg*, *not-r*)))
$\rightarrow$ (¬ eval (*r*, s (*prg*, jleads (ies (*prg*, *r*), *prg*, *not-r*))))))
$\rightarrow$ (¬ eventually-stable (*r*, *prg*))

EVENT: Disable true-leads-to-proves-not-eventually-stable.

THEOREM: eventually-stable-proves-not-true-leads-to
(eventually-stable (*not-q*, *prg*)
$\wedge$ (eval (*not-q*, s (*prg*, jleads (ies (*prg*, *not-q*), *prg*, *q*)))
$\rightarrow$ (¬ eval (*q*, s (*prg*, jleads (ies (*prg*, *not-q*), *prg*, *q*))))))
$\rightarrow$ (¬ leads-to (`'(true)`, *q*, *prg*))

EVENT: Disable eventually-stable-proves-not-true-leads-to.

THEOREM: eventually-stable-weaken
(eventually-stable (*p*, *prg*)
 ∧ (eval (*p*, s (*prg*, jes (ies (*prg*, *p*), *prg*, *r*)))
  → eval (*r*, s (*prg*, jes (ies (*prg*, *p*), *prg*, *r*)))))
→ eventually-stable (*r*, *prg*)

EVENT: Disable eventually-stable-weaken.

THEOREM: eventually-stable-conjunction
(eventually-stable (*p*, *prg*)
 ∧ eventually-stable (*q*, *prg*)
 ∧ (eval (list (`'and`, *p*, *q*),
 s (*prg*,
 jes (**if** ies (*prg*, *p*) < ies (*prg*, *q*) **then** ies (*prg*, *q*)
 else ies (*prg*, *p*) **endif**,
 prg,
 r)))
  → eval (*r*,
 s (*prg*,
 jes (**if** ies (*prg*, *p*) < ies (*prg*, *q*) **then** ies (*prg*, *q*)
 else ies (*prg*, *p*) **endif**,
 prg,
 r)))))
→ eventually-stable (*r*, *prg*)

EVENT: Disable eventually-stable-conjunction.

THEOREM: eventually-stable-false
(leads-to (*p*, *q*, *prg*)
 ∧ (eval (*q*, s (*prg*, jleads (ies (*prg*, *p*), *prg*, *q*)))
  → (¬ eval (*p*, s (*prg*, jleads (ies (*prg*, *p*), *prg*, *q*))))))
→ (¬ eventually-stable (*p*, *prg*))

EVENT: Disable eventually-stable-false.

THEOREM: stable-occurs-proves-eventually-stable
(listp (*prg*) ∧ unless (*p*, `'(false)`, *prg*) ∧ leads-to (`'(true)`, *p*, *prg*))
→ eventually-stable (*p*, *prg*)

EVENT: Disable stable-occurs-proves-eventually-stable.

```
;;; The Basic Ensures Operator
```

DEFINITION:
$$\begin{array}{l}\text{ensures}\,(p,\ q,\ prg)\\ \leftrightarrow\quad \exists\, e\ ((e \in prg)\\ \qquad\wedge\quad \forall\, old,\ new\ ((\text{n}\,(old,\ new,\ e)\\ \qquad\qquad\qquad\wedge\quad \text{eval}\,(\text{list}\,(\texttt{'and},\ p,\ \text{list}\,(\texttt{'not},\ q)),\ old))\\ \qquad\qquad\qquad\rightarrow\quad \text{eval}\,(q,\ new)))\end{array}$$

EVENT: Disable ensures.

THEOREM: prove-ensures
$$\begin{array}{l}((e \in prg)\\ \wedge\quad ((\text{n}\,(\text{olde}\,(e,\ p,\ q),\ \text{newe}\,(e,\ p,\ q),\ e)\\ \qquad\wedge\quad \text{eval}\,(p,\ \text{olde}\,(e,\ p,\ q))\\ \qquad\wedge\quad (\neg\ \text{eval}\,(q,\ \text{olde}\,(e,\ p,\ q))))\\ \qquad\rightarrow\quad \text{eval}\,(q,\ \text{newe}\,(e,\ p,\ q))))\\ \rightarrow\quad \text{ensures}\,(p,\ q,\ prg)\end{array}$$

THEOREM: ensures-implies
$$\begin{array}{l}(\text{ensures}\,(p,\ q,\ prg) \rightarrow (\text{ee}\,(p,\ prg,\ q) \in prg))\\ \wedge\quad ((\text{ensures}\,(p,\ q,\ prg)\\ \qquad\wedge\quad \text{n}\,(old,\ new,\ \text{ee}\,(p,\ prg,\ q))\\ \qquad\wedge\quad \text{eval}\,(p,\ old)\\ \qquad\wedge\quad (\neg\ \text{eval}\,(q,\ old)))\\ \qquad\rightarrow\quad \text{eval}\,(q,\ new))\end{array}$$

EVENT: Disable prove-ensures.

EVENT: Disable ensures-implies.

THEOREM: ensures-proves-something
$$\begin{array}{l}(\text{ensures}\,(p,\ q,\ prg)\\ \wedge\quad \text{unless}\,(p,\ q,\ prg)\\ \wedge\quad \text{schedulable}\,(prg)\\ \wedge\quad (\text{next}\,(prg,\ \text{ee}\,(p,\ prg,\ q),\ base) < top)\\ \wedge\quad (\text{next}\,(prg,\ \text{ee}\,(p,\ prg,\ q),\ base) \not< i)\\ \wedge\quad (i \not< base)\\ \wedge\quad (i \in \mathbf{N})\\ \wedge\quad \text{eval}\,(p,\ \text{s}\,(prg,\ i)))\\ \rightarrow\quad \text{eval}\,(q,\ \text{s}\,(prg,\ \text{ensures-interval}\,(prg,\ q,\ top,\ i)))\end{array}$$

THEOREM: the-interval-of-ensures
(ensures (*p*, *q*, *prg*)
 ∧ unless (*p*, *q*, *prg*)
 ∧ schedulable (*prg*)
 ∧ eval (*p*, s (*prg*, *i*)))
→ eval (*q*, s (*prg*, ensures-interval (*prg*, *q*, 1 + next (*prg*, ee (*p*, *prg*, *q*), *i*), *i*)))

THEOREM: ensures-proves-leads-to
(schedulable (*prg*) ∧ unless (*p*, *q*, *prg*) ∧ ensures (*p*, *q*, *prg*))
→ leads-to (*p*, *q*, *prg*)

EVENT: Disable ensures-proves-something.

EVENT: Disable the-interval-of-ensures.

EVENT: Disable ensures-proves-leads-to.

```
;;; Total Programs, Statements are Assignments
```

DEFINITION:
total (*prg*) ↔ ∀ *e* ((*e* ∈ *prg*) → ∀ *old* ∃ *new* n (*old*, *new*, *e*))

EVENT: Disable total.

THEOREM: prove-total
((et (*prg*) ∈ *prg*) → n (oldt (*prg*), *new*, et (*prg*))) → total (*prg*)

THEOREM: total-implies
(total (*prg*) ∧ (*e* ∈ *prg*)) → n (*old*, newt (*e*, *old*), *e*)

EVENT: Disable prove-total.

EVENT: Disable total-implies.

THEOREM: total-implies-schedulable
(total (*prg*) ∧ listp (*prg*)) → schedulable (*prg*)

```
;;; Enabled Transitions
```

DEFINITION:
enabling-condition $(c,\ e,\ prg)$
$\leftrightarrow$ $((e \in prg)$
$\wedge$ $\forall\ old,\ new\ (\mathrm{n}\,(old,\ new,\ e) \rightarrow \mathrm{eval}\,(c,\ old))$
$\wedge$ $\forall\ old\ (\mathrm{eval}\,(c,\ old) \rightarrow \exists\ new\ \mathrm{n}\,(old,\ new,\ e)))$

EVENT: Disable enabling-condition.

THEOREM: prove-enabling-condition
$((e \in prg)$
$\wedge$ $(\mathrm{n}\,(\mathrm{oldc}\,(c,\ e),\ \mathrm{newc}\,(c,\ e),\ e) \rightarrow \mathrm{eval}\,(c,\ \mathrm{oldc}\,(c,\ e)))$
$\wedge$ $(\mathrm{eval}\,(c,\ \text{oldc-1}\,(c,\ e)) \rightarrow \mathrm{n}\,(\text{oldc-1}\,(c,\ e),\ new,\ e)))$
$\rightarrow$ enabling-condition $(c,\ e,\ prg)$

THEOREM: enabling-condition-implies
$(\text{enabling-condition}\,(c,\ e,\ prg) \rightarrow (e \in prg))$
$\wedge$ $((\text{enabling-condition}\,(c,\ e,\ prg) \wedge \mathrm{n}\,(old,\ new,\ e)) \rightarrow \mathrm{eval}\,(c,\ old))$
$\wedge$ $((\text{enabling-condition}\,(c,\ e,\ prg) \wedge \mathrm{eval}\,(c,\ old))$
$\rightarrow$ $\mathrm{n}\,(old,\ \text{newc-1}\,(e,\ old),\ e))$

EVENT: Disable prove-enabling-condition.

EVENT: Disable enabling-condition-implies.

```
;;; Ensures with Enabling
```

DEFINITION:
e-ensures $(p,\ q,\ c,\ prg)$
$\leftrightarrow$ $\exists\ e\ ((e \in prg)$
$\wedge$ enabling-condition $(c,\ e,\ prg)$
$\wedge$ $\forall\ old,\ new\ ((\mathrm{n}\,(old,\ new,\ e)$
$\wedge$ $\mathrm{eval}\,(\mathrm{list}\,(\texttt{'and},\ p,\ \mathrm{list}\,(\texttt{'not},\ q)),\ old))$
$\rightarrow$ $\mathrm{eval}\,(q,\ new)))$

EVENT: Disable e-ensures.

THEOREM: prove-e-ensures
$((e \in prg)$
$\wedge$ enabling-condition $(c,\ e,\ prg)$
$\wedge$ $((\mathrm{n}\,(\mathrm{oldee}\,(e,\ p,\ q),\ \mathrm{newee}\,(e,\ p,\ q),\ e)$
$\wedge$ $\mathrm{eval}\,(\mathrm{list}\,(\texttt{'and},\ p,\ \mathrm{list}\,(\texttt{'not},\ q)),\ \mathrm{oldee}\,(e,\ p,\ q)))$
$\rightarrow$ $\mathrm{eval}\,(q,\ \mathrm{newee}\,(e,\ p,\ q))))$
$\rightarrow$ e-ensures $(p,\ q,\ c,\ prg)$

EVENT: Disable prove-e-ensures.

DEFINITION:
e-ensures-enabling $(p,\ q,\ c,\ prg)$
$\leftrightarrow\quad \exists\ e\ ((e \in prg)$
$\quad \wedge\quad \forall\ old,\ new\ ((\mathrm{n}\,(old,\ new,\ e) \rightarrow \mathrm{eval}\,(c,\ old))$
$\quad\quad \wedge\quad ((\mathrm{n}\,(old,\ new,\ e)$
$\quad\quad\quad \wedge\quad \mathrm{eval}\,(\mathrm{list}\,(\texttt{'and},$
$\quad\quad\quad\quad\quad\quad\quad\quad p,$
$\quad\quad\quad\quad\quad\quad\quad\quad \mathrm{list}\,(\texttt{'not},\ q)),$
$\quad\quad\quad\quad\quad\quad\quad old))$
$\quad\quad\quad\quad\quad\quad \rightarrow\quad \mathrm{eval}\,(q,\ new)))$
$\quad \wedge\quad \forall\ old\ (\mathrm{eval}\,(c,\ old) \rightarrow \exists\ new\ \mathrm{n}\,(old,\ new,\ e)))$

EVENT: Disable e-ensures-enabling.

THEOREM: help-prove-e-ensures
$((e \in prg)$
$\wedge\quad (\mathrm{n}\,(\mathrm{oldeee}\,(c,\ e,\ p,\ q),\ \mathrm{neweee}\,(c,\ e,\ p,\ q),\ e)$
$\quad\quad \rightarrow\quad (\mathrm{eval}\,(c,\ \mathrm{oldeee}\,(c,\ e,\ p,\ q)) \wedge \mathrm{eval}\,(q,\ \mathrm{neweee}\,(c,\ e,\ p,\ q))))$
$\wedge\quad (\mathrm{eval}\,(c,\ \mathrm{oldeee\text{-}1}\,(c,\ e)) \rightarrow \mathrm{n}\,(\mathrm{oldeee\text{-}1}\,(c,\ e),\ new,\ e))$
$\wedge\quad (\mathrm{eval}\,(c,\ \mathrm{oldeee}\,(c,\ e,\ p,\ q))$
$\quad\quad \rightarrow\quad \mathrm{eval}\,(\mathrm{list}\,(\texttt{'and},\ p,\ \mathrm{list}\,(\texttt{'not},\ q)),\ \mathrm{oldeee}\,(c,\ e,\ p,\ q))))$
$\rightarrow\quad \mathrm{e\text{-}ensures}\,(p,\ q,\ c,\ prg)$

THEOREM: e-ensures-implies
$(\mathrm{e\text{-}ensures}\,(p,\ q,\ c,\ prg) \rightarrow (\mathrm{eee}\,(c,\ p,\ prg,\ q) \in prg))$
$\wedge\quad (\mathrm{e\text{-}ensures}\,(p,\ q,\ c,\ prg) \rightarrow \mathrm{enabling\text{-}condition}\,(c,\ \mathrm{eee}\,(c,\ p,\ prg,\ q),\ prg))$
$\wedge\quad ((\mathrm{e\text{-}ensures}\,(p,\ q,\ c,\ prg)$
$\quad\quad \wedge\quad \mathrm{n}\,(old,\ new,\ \mathrm{eee}\,(c,\ p,\ prg,\ q))$
$\quad\quad \wedge\quad \mathrm{eval}\,(\mathrm{list}\,(\texttt{'and},\ p,\ \mathrm{list}\,(\texttt{'not},\ q)),\ old))$
$\quad\quad \rightarrow\quad \mathrm{eval}\,(q,\ new))$

EVENT: Disable e-ensures-implies.

EVENT: Disable help-prove-e-ensures.

```
;;; Union Theorems
```

THEOREM: total-union-1
$\mathrm{total}\,(\mathrm{append}\,(prg\text{-}1,\ prg\text{-}2)) \rightarrow (\mathrm{total}\,(prg\text{-}1) \wedge \mathrm{total}\,(prg\text{-}2))$

THEOREM: total-union-2
$(\text{total}\,(\textit{prg-1}) \wedge \text{total}\,(\textit{prg-2})) \rightarrow \text{total}\,(\text{append}\,(\textit{prg-1}, \textit{prg-2}))$

THEOREM: total-union
$\text{total}\,(\text{append}\,(\textit{prg-1}, \textit{prg-2})) = (\text{total}\,(\textit{prg-1}) \wedge \text{total}\,(\textit{prg-2}))$

EVENT: Disable total-union-1.

EVENT: Disable total-union-2.

THEOREM: unless-union-1
$\text{unless}\,(p, q, \text{append}\,(\textit{prg-1}, \textit{prg-2}))$
$\rightarrow \quad (\text{unless}\,(p, q, \textit{prg-1}) \wedge \text{unless}\,(p, q, \textit{prg-2}))$

THEOREM: unless-union-2
$(\text{unless}\,(p, q, \textit{prg-1}) \wedge \text{unless}\,(p, q, \textit{prg-2}))$
$\rightarrow \quad \text{unless}\,(p, q, \text{append}\,(\textit{prg-1}, \textit{prg-2}))$

THEOREM: unless-union
$\text{unless}\,(p, q, \text{append}\,(\textit{prg-1}, \textit{prg-2}))$
$= \quad (\text{unless}\,(p, q, \textit{prg-1}) \wedge \text{unless}\,(p, q, \textit{prg-2}))$

EVENT: Disable unless-union-1.

EVENT: Disable unless-union-2.

THEOREM: ensures-union-1
$\text{ensures}\,(p, q, \text{append}\,(\textit{prg-1}, \textit{prg-2}))$
$\rightarrow \quad (\text{ensures}\,(p, q, \textit{prg-1}) \vee \text{ensures}\,(p, q, \textit{prg-2}))$

THEOREM: ensures-union-2
$(\text{ensures}\,(p, q, \textit{prg-1}) \vee \text{ensures}\,(p, q, \textit{prg-2}))$
$\rightarrow \quad \text{ensures}\,(p, q, \text{append}\,(\textit{prg-1}, \textit{prg-2}))$

EVENT: Disable ensures-union-1.

EVENT: Disable ensures-union-2.

THEOREM: ensures-union
$\text{ensures}\,(p, q, \text{append}\,(\textit{prg-1}, \textit{prg-2}))$
$= \quad (\text{ensures}\,(p, q, \textit{prg-1}) \vee \text{ensures}\,(p, q, \textit{prg-2}))$

```
;;; Help Prove Total, Unless, and Ensures.
```

DEFINITION:
total-sufficient (*statement*, *program*, *old*, *new*)
= ((*statement* ∈ *program*) → n (*old*, *new*, *statement*))

THEOREM: help-prove-total
total-sufficient (et (*prg*), *prg*, oldt (*prg*), *new*) → total (*prg*)

DEFINITION:
unless-sufficient (*statement*, *program*, *old*, *new*, *p*, *q*)
= (((*statement* ∈ *program*)
∧ n (*old*, *new*, *statement*)
∧ eval (*p*, *old*)
∧ (¬ eval (*q*, *old*)))
→ eval (list ('or, *p*, *q*), *new*))

THEOREM: help-prove-unless
unless-sufficient (eu (*p*, *prg*, *q*), *prg*, oldu (*p*, *prg*, *q*), newu (*p*, *prg*, *q*), *p*, *q*)
→ unless (*p*, *q*, *prg*)

DEFINITION:
ensures-key (*statement*, *program*, *old*, *new*, *p*, *q*)
= ((*statement* ∈ *program*)
∧ ((n (*old*, *new*, *statement*) ∧ eval (*p*, *old*) ∧ (¬ eval (*q*, *old*)))
→ eval (*q*, *new*)))

THEOREM: help-prove-ensures
ensures-key (*statement*, *prg*, olde (*statement*, *p*, *q*), newe (*statement*, *p*, *q*), *p*, *q*)
→ ensures (*p*, *q*, *prg*)

DEFINITION:
ensures-rest (*statement*, *key*, *program*, *old*, *new*, *p*, *q*)
= (((*statement* ∈ *program*)
∧ (*statement* ≠ *key*)
∧ n (*old*, *new*, *statement*)
∧ eval (*p*, *old*)
∧ (¬ eval (*q*, *old*)))
→ eval (*p*, *new*))

THEOREM: help-prove-unless-ensures
(ensures-key (*statement*, *prg*, oldu (*p*, *prg*, *q*), newu (*p*, *prg*, *q*), *p*, *q*)
∧ ensures-rest (eu (*p*, *prg*, *q*),

$statement$,
prg,
oldu $(p,\ prg,\ q)$,
newu $(p,\ prg,\ q)$,
p,
q))
$\rightarrow$ unless $(p,\ q,\ prg)$

```
;;; Strengthening and Weakening Unless and Ensures
```

DEFINITION:
stronger-p $(p,\ q) \leftrightarrow \forall\ state$ (eval $(p,\ state) \rightarrow$ eval $(q,\ state)$)

EVENT: Disable stronger-p.

THEOREM: stronger-p-implies
(stronger-p $(p,\ q) \wedge$ eval $(p,\ state)) \rightarrow$ eval $(q,\ state)$

THEOREM: stronger-p-rewrite
stronger-p $(p,\ q) =$ (eval $(p,$ states $(p,\ q)) \rightarrow$ eval $(q,$ states $(p,\ q)))$

EVENT: Disable stronger-p-implies.

DEFINITION:
equal-p $(p,\ q) \leftrightarrow \forall\ state$ (eval $(p,\ state) =$ eval $(q,\ state)$)

EVENT: Disable equal-p.

THEOREM: equal-p-implies
equal-p $(p,\ q) \rightarrow$ (eval $(p,\ state) =$ eval $(q,\ state)$)

EVENT: Disable equal-p-implies.

THEOREM: equal-p-rewrite
equal-p $(p,\ q) =$ (eval $(p,$ statee $(p,\ q)) =$ eval $(q,$ statee $(p,\ q)))$

THEOREM: equal-p-commutative
equal-p $(p,\ q) =$ equal-p $(q,\ p)$

THEOREM: ensures-strengthen-left
(ensures $(q,\ r,\ prg) \wedge$ stronger-p $(p,\ q)) \rightarrow$ ensures $(p,\ r,\ prg)$

EVENT: Disable ensures-strengthen-left.

THEOREM: ensures-weaken-right
(ensures (*p*, *q*, *prg*) ∧ stronger-p (*q*, *r*)) → ensures (*p*, *r*, *prg*)

EVENT: Disable ensures-weaken-right.

THEOREM: unless-weaken-right
(unless (*p*, *q*, *prg*) ∧ stronger-p (*q*, *r*)) → unless (*p*, *r*, *prg*)

EVENT: Disable unless-weaken-right.

THEOREM: unless-equal-p
equal-p (*p*, *q*) → (unless (*p*, *r*, *prg*) = unless (*q*, *r*, *prg*))

THEOREM: unless-conjunction
(unless (*p-1*, *q*, *prg*) ∧ unless (*p-2*, *q*, *prg*) ∧ equal-p (*p*, list (`'and`, *p-1*, *p-2*)))
→ unless (*p*, *q*, *prg*)

THEOREM: unless-disjunction
(unless (*p-1*, *q*, *prg*) ∧ unless (*p-2*, *q*, *prg*) ∧ equal-p (*p*, list (`'or`, *p-1*, *p-2*)))
→ unless (*p*, *q*, *prg*)

```
;;; Fairness Theorems
```

THEOREM: unconditional-fairness
(unless (*p*, *q*, *prg*) ∧ ensures (*p*, *q*, *prg*) ∧ total (*prg*)) → leads-to (*p*, *q*, *prg*)

EVENT: Disable unconditional-fairness.

THEOREM: unconditional-fairness-general
(unless (*p-1*, *q-1*, *prg*)
∧ ensures (*p-2*, *q-2*, *prg*)
∧ (eval (*p*, s (*prg*, ileads (*p*, *prg*, *q*))) → eval (*p-1*, s (*prg*, ileads (*p*, *prg*, *q*))))
∧ stronger-p (*p-1*, *p-2*)
∧ (eval (*q-1*, s (*prg*, jleads (ileads (*p-1*, *prg*, *q*), *prg*, *q-1*)))
→ eval (*q*, s (*prg*, jleads (ileads (*p-1*, *prg*, *q*), *prg*, *q-1*))))
∧ stronger-p (*q-2*, *q-1*)
∧ total (*prg*))
→ leads-to (*p*, *q*, *prg*)

EVENT: Disable unconditional-fairness-general.

CONSERVATIVE AXIOM: strong-fairness
(unless (*p*, *q*, *prg*)
∧ e-ensures (*p*, *q*, *c*, *prg*)
∧ leads-to (*p*, list ('or, *q*, *c*), *prg*)
∧ strongly-fair (*prg*))
→ leads-to (*p*, *q*, *prg*)

Simultaneously, we introduce the new function symbol *strongly-fair*.
EVENT: Disable strong-fairness.

THEOREM: strong-fairness-general
(unless (*p-1*, *q-1*, *prg*)
∧ e-ensures (*p-2*, *q-2*, *c*, *prg*)
∧ leads-to (*p*, list ('or, *q*, *c*), *prg*)
∧ (eval (*p*, s (*prg*, ileads (*p*, *prg*, *q*))) → eval (*p-1*, s (*prg*, ileads (*p*, *prg*, *q*))))
∧ (eval (*p-1*, s (*prg*, ileads (*p-1*, *prg*, list ('or, *q-1*, *c*))))
→ eval (*p*, s (*prg*, ileads (*p-1*, *prg*, list ('or, *q-1*, *c*)))))
∧ stronger-p (*p-1*, *p-2*)
∧ (eval (*q-1*, s (*prg*, jleads (ileads (*p-1*, *prg*, *q*), *prg*, *q-1*)))
→ eval (*q*, s (*prg*, jleads (ileads (*p-1*, *prg*, *q*), *prg*, *q-1*))))
∧ (eval (list ('or, *q*, *c*),
s (*prg*,
jleads (ileads (*p*, *prg*, list ('or, *q-1*, *c*)), *prg*, list ('or, *q*, *c*))))
→ eval (list ('or, *q-1*, *c*),
s (*prg*,
jleads (ileads (*p*, *prg*, list ('or, *q-1*, *c*)),
prg,
list ('or, *q*, *c*)))))
∧ stronger-p (*q-2*, *q-1*)
∧ strongly-fair (*prg*))
→ leads-to (*p*, *q*, *prg*)

EVENT: Disable strong-fairness-general.

DEFINITION:
wfw (*i*, *j*, *p*, *q*, *c*, *prg*)
= **if** $i < j$
then if eval (*q*, s (*prg*, *i*)) **then** *i*
elseif eval (*p*, s (*prg*, *i*))
then if eval (*c*, s (*prg*, *i*)) **then** wfw (1 + *i*, *j*, *p*, *q*, *c*, *prg*)
else *i* **endif**
else *i* **endif**
else fix (*i*) **endif**

THEOREM: wfw-bigger
$\text{wfw}(i, j, p, q, c, prg) \not< i$

THEOREM: about-wfw
$(\text{unless}(p, q, prg) \wedge \text{listp}(prg) \wedge \text{eval}(p, \text{s}(prg, i)) \wedge (j \not< i))$
$\rightarrow \quad (\text{eval}(q, \text{s}(prg, \text{wfw}(i, j, p, q, c, prg)))$
$\vee \quad (\text{eval}(p, \text{s}(prg, \text{wfw}(i, j, p, q, c, prg)))$
$\wedge \quad (\neg\, \text{eval}(q, \text{s}(prg, \text{wfw}(i, j, p, q, c, prg))))$
$\wedge \quad (\neg\, \text{eval}(c, \text{s}(prg, \text{wfw}(i, j, p, q, c, prg)))))$
$\vee \quad ((\text{wfw}(i, j, p, q, c, prg) = \text{fix}(j))$
$\wedge \quad (\text{eval}(p, \text{s}(prg, j)) \vee \text{eval}(q, \text{s}(prg, j)))))$

DEFINITION:
witness (p, q, c, prg)
$=$ wfw (ileads (p, prg, q),
next $(prg, \text{eee}(c, p, prg, q), \text{ileads}(p, prg, q))$,
p,
q,
c,
prg)

THEOREM: weak-fairness
$(\text{unless}(p, q, prg)$
$\wedge \quad \text{e-ensures}(p, q, c, prg)$
$\wedge \quad (\text{eval}(\text{list}(\texttt{'and}, p, \text{list}(\texttt{'not}, q)), \text{s}(prg, \text{witness}(p, q, c, prg)))$
$\rightarrow \quad \text{eval}(c, \text{s}(prg, \text{witness}(p, q, c, prg)))))$
$\rightarrow \quad \text{leads-to}(p, q, prg)$

EVENT: Disable weak-fairness.

THEOREM: weak-fairness-general
$(\text{unless}(p\text{-}1, q\text{-}1, prg)$
$\wedge \quad \text{e-ensures}(p\text{-}2, q\text{-}2, c, prg)$
$\wedge \quad (\text{eval}(\text{list}(\texttt{'and}, p\text{-}1, \text{list}(\texttt{'not}, q\text{-}1)), \text{s}(prg, \text{witness}(p\text{-}1, q\text{-}1, c, prg)))$
$\rightarrow \quad \text{eval}(c, \text{s}(prg, \text{witness}(p\text{-}1, q\text{-}1, c, prg))))$
$\wedge \quad (\text{eval}(p, \text{s}(prg, \text{ileads}(p, prg, q))) \rightarrow \text{eval}(p\text{-}1, \text{s}(prg, \text{ileads}(p, prg, q))))$
$\wedge \quad \text{stronger-p}(p\text{-}1, p\text{-}2)$
$\wedge \quad (\text{eval}(q\text{-}1, \text{s}(prg, \text{jleads}(\text{ileads}(p\text{-}1, prg, q), prg, q\text{-}1)))$
$\rightarrow \quad \text{eval}(q, \text{s}(prg, \text{jleads}(\text{ileads}(p\text{-}1, prg, q), prg, q\text{-}1))))$
$\wedge \quad \text{stronger-p}(q\text{-}2, q\text{-}1))$
$\rightarrow \quad \text{leads-to}(p, q, prg)$

EVENT: Disable weak-fairness-general.

EVENT: Disable witness.

THEOREM: deadlock-freedom-witness
(unless (*inv*, `'(false)`, *prg*)
 ∧ enabling-condition (*c*, *e*, *prg*)
 ∧ (eval (*inv*, s (*prg*, next (*prg*, *e*, ileads (*inv*, *prg*, `'(false)`))))
  → (¬ eval (*c*, s (*prg*, next (*prg*, *e*, ileads (*inv*, *prg*, `'(false)`)))))))
 ∧ schedulable (*prg*))
→ leads-to (*inv*, `'(false)`, *prg*)

EVENT: Disable deadlock-freedom-witness.

CONSERVATIVE AXIOM: deadlock-freedom
(unless (*inv*, `'(false)`, *prg*)
 ∧ enabling-condition (*c*, *e*, *prg*)
 ∧ (eval (*inv*, s (*prg*, next (*prg*, *e*, ileads (*inv*, *prg*, `'(false)`))))
  → (¬ eval (*c*, s (*prg*, next (*prg*, *e*, ileads (*inv*, *prg*, `'(false)`)))))))
 ∧ deadlock-free (*prg*))
→ leads-to (*inv*, `'(false)`, *prg*)

Simultaneously, we introduce the new function symbol *deadlock-free*.

THEOREM: deadlock-freedom-general
(unless (*inv*, `'(false)`, *prg*)
 ∧ enabling-condition (*c*, *e*, *prg*)
 ∧ (eval (*inv*, s (*prg*, next (*prg*, *e*, ileads (*inv*, *prg*, `'(false)`))))
  → (¬ eval (*c*, s (*prg*, next (*prg*, *e*, ileads (*inv*, *prg*, `'(false)`)))))))
 ∧ ((¬ eval (*inv*, s (*prg*, ii (*p*, *prg*)))) → eval (*p*, s (*prg*, ii (*p*, *prg*))))
 ∧ deadlock-free (*prg*))
→ invariant (*p*, *prg*)

EVENT: Enable eval.

EVENT: Enable n.

```
;;; Some Helpful Definitions and Theorems
```

DEFINITION:
update-assoc (*key*, *value*, *alist*)
= **if** listp (*alist*)
 then if caar (*alist*) = *key* **then** cons (cons (*key*, *value*), cdr (*alist*))
  else cons (car (*alist*), update-assoc (*key*, *value*, cdr (*alist*))) **endif**
 else list (cons (*key*, *value*)) **endif**

THEOREM: simplify-assoc
assoc (*key-1*, update-assoc (*key-2*, *value*, *alist*))
= **if** *key-1* = *key-2* **then** cons (*key-1*, *value*)
else assoc (*key-1*, *alist*) **endif**

DEFINITION:
add1-mod (*n*, *x*)
= **if** (1 + *x*) < *n* **then** 1 + *x*
else 0 **endif**

DEFINITION:
sub1-mod (*n*, *x*)
= **if** *x* < *n*
then if *x* ≃ 0 **then** *n* − 1
else *x* − 1 **endif**
else 0 **endif**

DEFINITION:
uc (*old*, *new*, *keys*, *excpt*)
= **if** listp (*keys*)
then if car (*keys*) ∈ *excpt* **then** uc (*old*, *new*, cdr (*keys*), *excpt*)
elseif assoc (car (*keys*), *old*) = assoc (car (*keys*), *new*)
then uc (*old*, *new*, cdr (*keys*), *excpt*)
else f endif
else t endif

THEOREM: uc-basic-property
(uc (*old*, *new*, *keys*, *excpt*) ∧ (*key* ∈ *keys*) ∧ (*key* ∉ *excpt*))
→ ((assoc (*key*, *old*) = assoc (*key*, *new*)) = **t**)

THEOREM: uc-commutative
uc (*old*, *new*, *keys*, *excpt*) = uc (*new*, *old*, *keys*, *excpt*)

THEOREM: uc-reflexive
uc (*list*, *list*, *keys*, *excpt*)

THEOREM: uc-of-update-assoc
uc (*list-1*, update-assoc (*key*, *value*, *list-2*), *keys*, *excpt*)
= **if** *key* ∈ *excpt* **then** uc (*list-1*, *list-2*, *keys*, *excpt*)
elseif *key* ∈ *keys*
then (assoc (*key*, *list-1*) = cons (*key*, *value*))
∧ uc (*list-1*, *list-2*, *keys*, cons (*key*, *excpt*))
else uc (*list-1*, *list-2*, *keys*, *excpt*) **endif**

THEOREM: strip-cars-append
strip-cars (append (*a*, *b*)) = append (strip-cars (*a*), strip-cars (*b*))

THEOREM: uc-append
$\text{uc}(\mathit{old}, \mathit{new}, \text{append}(a, b), \mathit{excpt})$
$= \quad (\text{uc}(\mathit{old}, \mathit{new}, a, \mathit{excpt}) \wedge \text{uc}(\mathit{old}, \mathit{new}, b, \mathit{excpt}))$

THEOREM: uc-commutative-2
$\text{uc}(\mathit{old}, \mathit{new}, \text{append}(a, b), \mathit{excpt}) = \text{uc}(\mathit{old}, \mathit{new}, \text{append}(b, a), \mathit{excpt})$

EVENT: Disable uc-append.

THEOREM: key-not-member-strip-cars
$(\mathit{key} \notin \text{strip-cars}(\mathit{alist})) \rightarrow (\text{assoc}(\mathit{key}, \mathit{alist}) = \mathbf{f})$

THEOREM: uc-property
$(\text{uc}(\mathit{old}, \mathit{new}, \text{append}(\text{strip-cars}(\mathit{old}), \text{strip-cars}(\mathit{new})), \mathit{excpt})$
$\wedge \quad (\mathit{key} \notin \mathit{excpt}))$
$\rightarrow \quad ((\text{assoc}(\mathit{key}, \mathit{old}) = \text{assoc}(\mathit{key}, \mathit{new})) = \mathbf{t})$

THEOREM: about-uc
$(\text{uc}(a, b, \text{append}(\text{strip-cars}(a), \text{strip-cars}(b)), \mathit{excpt}) \wedge (\mathit{key} \notin \mathit{excpt}))$
$\rightarrow \quad (\text{assoc}(\mathit{key}, a) = \text{assoc}(\mathit{key}, b))$

DEFINITION:
$\text{changed}(\mathit{old}, \mathit{new}, \mathit{excpt}) = \text{uc}(\mathit{old}, \mathit{new}, \text{strip-cars}(\text{append}(\mathit{old}, \mathit{new})), \mathit{excpt})$

EVENT: Make the library `"interpreter"`.

Index

about-uc, 27
about-wfw, 24
add1-mod, 26
append-is-associative, 2

cancellation-leads-to, 10
cancellation-leads-to-general, 10
changed, 27
choose, 4, 7, 8
computation, 4
computation-n, 8

deadlock-free, 25
deadlock-freedom, 25
deadlock-freedom-general, 25
deadlock-freedom-witness, 25
disjoin-left, 9
disjoin-left-general, 9

e-ensures, 17, 18, 23, 24
e-ensures-enabling, 18
e-ensures-implies, 18
ee, 15, 16
eee, 18, 24
effective-idle, 8
enabling-condition, 17, 18, 25
enabling-condition-implies, 17
ensures, 15, 16, 19–22
ensures-implies, 15
ensures-interval, 10, 11, 15, 16
ensures-interval-bigger, 10
ensures-interval-fixes, 10
ensures-key, 20
ensures-proves-leads-to, 16
ensures-proves-something, 15
ensures-rest, 20, 21
ensures-strengthen-left, 21
ensures-union, 19
ensures-union-1, 19
ensures-union-2, 19
ensures-weaken-right, 22

equal-iff, 1
equal-p, 21, 22
equal-p-commutative, 21
equal-p-implies, 21
equal-p-rewrite, 21
et, 16, 20
eu, 6, 20
eval, 5–18, 20–25
eval-and, 5
eval-equal, 6
eval-false, 6
eval-iff, 6
eval-implies, 6
eval-not, 5
eval-or, 6
eval-true, 6
eventually-stable, 12–14
eventually-stable-conjunction, 14
eventually-stable-false, 14
eventually-stable-implies, 13
eventually-stable-proves-not-true-leads-to, 13
eventually-stable-weaken, 14
exists, 3, 7, 12, 15–18
exists-successor, 3
exists-successor-implies, 3

false-leads-to-anything, 8
forall, 6, 7, 12, 15–18, 21

help-prove-e-ensures, 18
help-prove-ensures, 20
help-prove-total, 20
help-prove-unless, 20
help-prove-unless-ensures, 20

ies, 13, 14
ii, 12, 25
ileads, 7–11, 13, 22–25
initial-condition, 11, 12
invariant, 12, 25

invariant-consequence, 12
invariant-implies, 12
invariants-persist-general, 12
is, 7

jes, 13, 14
jleads, 7–11, 13, 14, 22–24

key-not-member-strip-cars, 27

leads-to, 7–14, 16, 22–25
leads-to-false-invariant, 12
leads-to-implies, 7
leads-to-modify-both, 9
leads-to-strengthen-left, 9
leads-to-transitive, 8
leads-to-transitive-general, 8
leads-to-true, 11
leads-to-weaken-right, 9
length, 1–3
listp-not-zero-length, 1

mchoose, 2, 4
mchoose-chooses, 2
mchoose-fixes, 2
member-append, 2
mnext, 2
mnext-choice-1, 2
mnext-choice-2, 2
mnext-choice-2-simplified, 2
mnext-fixes, 2
ms, 4
ms-transition-idle, 4
ms-transition-successful, 4

n, 3, 4, 6–8, 15–18, 20
newc, 17
newc-1, 17
newe, 15, 20
newee, 17
neweee, 18
news, 7
newt, 16
newu, 6, 20, 21
newx, 3, 4
next, 4, 8, 15, 16, 24, 25
not-eventually-stable-proves-leads-to, 13
not-leads-to-proves-eventually-stable, 13
nth, 1–3
nth-member, 1
nth-position, 1
numberp-mnext, 2

oldc, 17
oldc-1, 17
olde, 15, 20
oldee, 17
oldeee, 18
oldeee-1, 18
oldt, 16, 20
oldu, 6, 20, 21

p-implies-q-leads-to, 9
position, 1–3
position-lessp, 1
position-nth, 2
position-zero, 1
prove-e-ensures, 17
prove-enabling-condition, 17
prove-ensures, 15
prove-eventually-stable, 13
prove-exists-successor, 3
prove-invariant, 12
prove-leads-to, 7
prove-schedulable, 7
prove-total, 16
prove-unless, 6
psp, 11
psp-general, 11
psp-proves-something, 10

q-leads-to-q, 8

remainder-of-add1, 3
remainder-of-add1-1, 3
remainder-of-add1-2, 3
remainder-of-add1-3, 3

remainder-of-add1-3-1, 3

s, 4, 7–16, 22–25
schedulable, 7, 8, 15, 16, 25
schedulable-implies, 7
schedulable-implies-effective-c
omputation, 8
simplify-assoc, 26
stable-occurs-proves-eventually
-stable, 14
statee, 21
states, 21
strip-cars-append, 26
strong-fairness, 23
strong-fairness-general, 23
stronger-p, 21–24
stronger-p-implies, 21
stronger-p-rewrite, 21
strongly-fair, 23
sub1-mod, 26

the-interval-of-ensures, 16
total, 16, 18–20, 22
total-implies, 16
total-implies-schedulable, 16
total-sufficient, 20
total-union, 19
total-union-1, 18
total-union-2, 19
true-leads-to-proves-not-eventu
ally-stable, 13

uc, 26, 27
uc-append, 27
uc-basic-property, 26
uc-commutative, 26
uc-commutative-2, 27
uc-of-update-assoc, 26
uc-property, 27
uc-reflexive, 26
unconditional-fairness, 22
unconditional-fairness-general, 22
unless, 6, 10–12, 14–16, 19–25
unless-conjunction, 22
unless-disjunction, 22
unless-equal-p, 22
unless-implies, 6
unless-proves-invariant, 12
unless-sufficient, 20
unless-union, 19
unless-union-1, 19
unless-union-2, 19
unless-weaken-right, 22
update-assoc, 25, 26

weak-fairness, 24
weak-fairness-general, 24
wfw, 23, 24
wfw-bigger, 24
witness, 24