#|

A Formal Model of Asynchronous Communication and Its Use in Mechanically Verifying a Biphase Mark Protocol

J Strother Moore

Technical Report 68
August, 1991

Abstract (of CLI Tech Report 68):

In this paper we present a formal model of asynchronous communication as a function in the Boyer-Moore logic. The function transforms the signal stream generated by one processor into the signal stream consumed by an independently clocked processor. This transformation ‘‘blurs’’ edges and ‘‘dilates’’ time due to differences in the phases and rates of

the two clocks and the communications delay. The model can be used quantitatively to derive concrete performance bounds on asynchronous communications at ISO protocol level 1 (physical level). We develop part of the reusable formal theory that permits the convenient application of the model. We use the theory to show that a biphase mark protocol can be used to send messages of arbitrary length between two asynchronous processors. We study two versions of the protocol, a conventional one which uses cells of size 32 cycles and an unconventional one which uses cells of size 18. Our proof of the former protocol requires the ratio of the clock rates of the two processors to be within 3% of unity. The unconventional biphase mark protocol permits the ratio to vary by 5%. At nominal clock rates of 20MHz, the unconventional protocol allows transmissions at a burst rate of slightly over 1MHz. These claims are formally stated in terms of our model of asynchrony; the proofs of the claims have been mechanically checked with the Boyer-Moore theorem prover, NQTHM. We conjecture that the protocol can be proved to work under our model for smaller cell sizes and more divergent clock rates but the proofs would be harder. Known inadequacies of our model include that (a) distortion due to the presence of an edge is limited to the time span of the cycle during which the edge was written, (b) both clocks are assumed to be linear functions of time (i.e., the rate of a given clock is unwavering) and (c) reading ``on an edge'' produces a nondeterministically defined value rather than an indeterminate value. We discuss these problems.

This event file contains all of the definitions and theorems mentioned in CLI Tech Report 68. In addition, it contains our proof that ``deterministic fuzzy edge detection is impossible,'' which is mentioned in a footnote of the report. However, the theorem BMP18 of the paper is here called TOP and BMP18-LEMMA is here LOOP. In addition, the functions lst' and ts' of the paper are here named LST+ and TS+.

This proof deals with the case of an 18-bit cell divided into subcells of size 5 and 13. The following numeric constants, which occur in this file, are related to this particular choice of cell configuration. It is possible to obtain proofs of different configurations by consistently replacing these constants and replaying the script. The proof could be lifted to a much more general one, but I just didn't feel like doing it.

```
 5 = mark-size
13 = code-size
18 = cell-size = (mark-size + code-size)
17 = cell-size-1
19 = cell-size+1
10 = sampling distance = mark-size+(code-size/2)-1
 4 = mark-size-1
12 = code-size-1

|#

; ------------------------------------------------------------------------
; Arithmetic
```

EVENT: Start with the initial **nqthm** theory.

DEFINITION:
rate-proximity $(w,\ r)$
$=$ $(((18 * w) \not< (17 * r)) \wedge ((19 * r) \not< (18 * w)))$

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

THEOREM: plus-commutes1
$(x + y) = (y + x)$

THEOREM: plus-commutes2
$(x + y + z) = (y + x + z)$

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

THEOREM: times-0
$(x * 0) = 0$

THEOREM: times-non-numberp
$(z \notin \mathbf{N}) \rightarrow ((x * z) = 0)$

THEOREM: times-add1
$(x * (1 + y)) = (x + (x * y))$

THEOREM: times-distributes1
$(x * (y + z)) = ((x * y) + (x * z))$

THEOREM: times-commutes1
$(x * y) = (y * x)$

THEOREM: times-commutes2
$(x * y * z) = (y * x * z)$

THEOREM: times-associates
$((x * y) * z) = (x * y * z)$

THEOREM: times-distributes2
$((x + y) * z) = ((x * z) + (y * z))$

THEOREM: difference-is-0
$(y \not< x) \rightarrow ((x - y) = 0)$

THEOREM: difference-plus-cancellation1
$((i + x) - i) = \text{fix}(x)$

THEOREM: difference-plus-cancellation2
$((i + x) - (i + y)) = (x - y)$

THEOREM: difference-plus-cancellation3
$((i + j + x) - j) = (i + x)$

THEOREM: lessp-remainder
$((x \textbf{ mod } y) < y) = (y \not\simeq 0)$

THEOREM: remainder-quotient-elim
$(x \in \mathbf{N}) \rightarrow (((x \textbf{ mod } y) + (y * (x \div y))) = x)$

THEOREM: quotient-plus-times
$(w \not\simeq 0) \rightarrow (((v + (w * i)) \div w) = (i + (v \div w)))$

DEFINITION:
$\text{len}(x)$
$=$ **if** $x \simeq$ **nil then** 0
else $1 + \text{len}(\text{cdr}(x))$ **endif**

THEOREM: equal-len-0
$(\text{len}(x) = 0) = (x \simeq \textbf{nil})$

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

THEOREM: app-cancellation
$(\text{app}(a,\ b) = \text{app}(a,\ c)) = (b = c)$

THEOREM: app-assoc
$\text{app}(\text{app}(a,\ b),\ c) = \text{app}(a,\ \text{app}(b,\ c))$

THEOREM: len-app
$\text{len}(\text{app}(a,\ b)) = (\text{len}(a) + \text{len}(b))$

THEOREM: quotient-x-x
$(x \not\simeq \mathtt{0}) \rightarrow ((x \div x) = \mathtt{1})$

THEOREM: not-lessp-times-quotient
$n \not< (w * (n \div w))$

THEOREM: times-monotonic
$((w \not\simeq \mathtt{0}) \wedge (a < b)) \rightarrow ((w * a) < (w * b))$

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

THEOREM: nsig*-alg-lemma-hack1
$((n < ((r + x) \div w)) \wedge (w \in \mathbf{N}) \wedge (w \neq \mathtt{0}))$
$\rightarrow \quad (((ts + (n * w)) < (r + ts + x)) = \mathbf{t})$

THEOREM: difference-plus-cancellation-4
$((r + ts + x) - (ts + y)) = ((r + x) - y)$

THEOREM: nts*-alg-lemma2-hack1
$((n < ((r + x) \div w))$
$\wedge \quad (x \in \mathbf{N})$
$\wedge \quad (r \neq \mathtt{0})$
$\wedge \quad (r \in \mathbf{N})$
$\wedge \quad ((n * w) \not< (r + x))$
$\wedge \quad (n \in \mathbf{N})$
$\wedge \quad (w \neq \mathtt{0})$
$\wedge \quad (w \in \mathbf{N})$
$\wedge \quad (x < w))$
$\rightarrow \quad ((((r + x) \div w)$
$= \quad ((x + (r * (((n * w) - x) \div r))) \div w))$
$= \quad \mathbf{t})$

THEOREM: times-cancellation1
$(i \not\simeq \mathtt{0}) \rightarrow (((i * j) = (i * k)) = (\text{fix}(j) = \text{fix}(k)))$

THEOREM: times-cancellation2
$(w \not\simeq \mathtt{0})$
$\rightarrow \quad ((((w * x) + (w * y)) = (w * z)) = ((x + y) = \mathrm{fix}\,(z)))$

THEOREM: difference-difference
$(b \not< c) \rightarrow ((a - (b - c)) = ((a + c) - b))$

THEOREM: difference-difference-other
$((a - b) - c) = (a - (b + c))$

THEOREM: quotient-plus-times2
$(w \not\simeq \mathtt{0})$
$\rightarrow \quad (((v + (i * w) + (w * j)) \div w) = (i + j + (v \div w)))$

THEOREM: difference-elim
$((i \in \mathbf{N}) \wedge (i \not< j)) \rightarrow ((j + (i - j)) = i)$

THEOREM: quotient-difference
$((w \not\simeq \mathtt{0}) \wedge (a \not< (w * b)))$
$\rightarrow \quad (((a - (w * b)) \div w) = ((a \div w) - b))$

THEOREM: quotient-monotonic-lemma
$((z1 < y) \wedge (z2 < y) \wedge (y \neq \mathtt{0}) \wedge (y \in \mathbf{N}) \wedge (x < v))$
$\rightarrow \quad (((z1 + (y * x)) < (z2 + (v * y))) = \mathbf{t})$

THEOREM: ntr*-alg-lemma2-hack1
$((n < ((r + x) \div w))$
$\wedge \quad (x \in \mathbf{N})$
$\wedge \quad (r \neq \mathtt{0})$
$\wedge \quad (r \in \mathbf{N})$
$\wedge \quad ((ts + (n * w)) \not< (r + ts + x))$
$\wedge \quad (n \in \mathbf{N})$
$\wedge \quad (ts \in \mathbf{N})$
$\wedge \quad (w \neq \mathtt{0})$
$\wedge \quad (w \in \mathbf{N})$
$\wedge \quad ((ts + x) \not< ts)$
$\wedge \quad ((ts + x) < (ts + w)))$
$\rightarrow \quad (((r + ts + x)$
$\quad = \quad (ts + x + (r * (((n * w) - x) \div r))))$
$\quad = \quad \mathbf{t})$

THEOREM: not-lessp-times-quotient-other
$(x \not< r) \rightarrow ((r * (x \div r)) \not< r)$

THEOREM: quotient-monotonic
$((w \not\simeq \mathtt{0}) \wedge (a \not< b)) \rightarrow (((a \div w) < (b \div w)) = \mathbf{f})$

THEOREM: nlst*-alg-lemma2-hack1
$$\begin{array}{ll}
((x \in \mathbf{N}) & \\
\wedge & (r \neq \mathtt{0}) \\
\wedge & (r \in \mathbf{N}) \\
\wedge & ((ts + (n * w)) \not< (r + ts + x)) \\
\wedge & (n \in \mathbf{N}) \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (w \neq \mathtt{0}) \\
\wedge & (w \in \mathbf{N}) \\
\wedge & ((ts + x) \not< ts) \\
\wedge & ((ts + x) < (ts + w))) \\
\rightarrow & ((((r + x) \div w) \\
& \quad + \quad (((x + (r * (((n * w) - x) \div r))) \\
& \qquad - \quad (w * ((r + x) \div w))) \\
& \qquad \div \quad w)) \\
& \quad = \quad ((x + (r * (((n * w) - x) \div r))) \div w))
\end{array}$$

THEOREM: nsig*-upper-bound-lemma1
$$(r \not\simeq \mathtt{0}) \rightarrow (((n * w) \div r) \not< (((n * w) - (tr - ts)) \div r))$$

THEOREM: quotient-times
$$(r \not\simeq \mathtt{0}) \rightarrow (((n * r) \div r) = \mathrm{fix}\,(n))$$

THEOREM: multiply-both-sides-of-lessp
$$(r \not\simeq \mathtt{0}) \rightarrow (((a * r) < (b * r)) = (a < b))$$

THEOREM: lessp-quotient-to-lessp-times-lemma1
$$((r \not\simeq \mathtt{0}) \wedge ((x \div r) \not< n)) \rightarrow (x \not< (n * r))$$

THEOREM: lessp-quotient-to-lessp-times-lemma2
$$((r \not\simeq \mathtt{0}) \wedge (x \not< (n * r))) \rightarrow ((x \div r) \not< n)$$

THEOREM: lessp-quotient-to-lessp-times
$$(r \not\simeq \mathtt{0}) \rightarrow (((x \div r) < n) = (x < (n * r)))$$

THEOREM: quotient-plus-times1
$$(r \not\simeq \mathtt{0}) \rightarrow ((((n * r) + z) \div r) = (n + (z \div r)))$$

THEOREM: equal-times-0
$$((i * j) = \mathtt{0}) = ((i \simeq \mathtt{0}) \vee (j \simeq \mathtt{0}))$$

THEOREM: nsig*-upper-bound-hack1
$$\begin{array}{ll}
((r \not\simeq \mathtt{0}) \wedge (r \not< (\mathtt{18} * \mathit{delta})) \wedge (n < \mathtt{18})) & \\
\rightarrow & (((n * \mathit{delta}) < r) = \mathbf{t})
\end{array}$$

THEOREM: nsig*-upper-bound-hack2

$$
\begin{array}{ll}
(((r + r + (\mathtt{17} * r)) \not< (w + (\mathtt{17} * w))) \\
\ \wedge & ((w + (\mathtt{17} * w)) \not< (\mathtt{17} * r)) \\
\ \wedge & (w \not\simeq \mathtt{0}) \\
\ \wedge & (r \not\simeq \mathtt{0}) \\
\ \wedge & (n \not\simeq \mathtt{0}) \\
\ \wedge & (n < \mathtt{18}) \\
\ \wedge & (w < r)) \\
\rightarrow & ((w + (w * (n - 1))) \\
& \quad = \quad (((n - 1) * (r - w)) \\
& \qquad + \quad (w - ((n - 1) * (r - w))) \\
& \qquad + \quad (w * (n - 1))))
\end{array}
$$

THEOREM: quotient-plus-times3

$$
\begin{array}{ll}
((x + z + (x * n)) \not\simeq \mathtt{0}) \\
\rightarrow & (((z + (x * n) + (z * n) + (x * n * n)) \\
& \quad \div \quad (x + z + (x * n))) \\
& \quad = \quad (n + (z \div (x + z + (n * x)))))
\end{array}
$$

THEOREM: nsig*-upper-bound-hack3

$$
\begin{array}{ll}
((z + (x * (n - 1))) < (x + z + (x * (n - 1)))) \\
\rightarrow & ((z < (x + z + (x * (n - 1)))) = \mathbf{t})
\end{array}
$$

THEOREM: intro-delta

$$((w \in \mathbf{N}) \wedge (w \not< r)) \rightarrow (w = (r + (w - r)))$$

THEOREM: nsig*-upper-bound-lemma2-1

$$
\begin{array}{ll}
((n \in \mathbf{N}) \\
\ \wedge & (w \not\simeq \mathtt{0}) \\
\ \wedge & (r \not\simeq \mathtt{0}) \\
\ \wedge & \text{rate-proximity}\,(w,\, r) \\
\ \wedge & (n < \mathtt{18}) \\
\ \wedge & (w \not< r)) \\
\rightarrow & (((n * w) \div r) = n)
\end{array}
$$

THEOREM: nsig*-upper-bound-lemma2-2

$$
\begin{array}{ll}
((n \in \mathbf{N}) \\
\ \wedge & (w \not\simeq \mathtt{0}) \\
\ \wedge & (r \not\simeq \mathtt{0}) \\
\ \wedge & \text{rate-proximity}\,(w,\, r) \\
\ \wedge & (n < \mathtt{18}) \\
\ \wedge & (w < r)) \\
\rightarrow & (((n * w) \div r) = (n - 1))
\end{array}
$$

THEOREM: nsig*-upper-bound-lemma2-equality

$((n \in \mathbf{N}) \wedge (w \not\simeq \mathtt{0}) \wedge (r \not\simeq \mathtt{0}) \wedge \text{rate-proximity}\,(w,\, r) \wedge (n < \mathtt{18}))$
$\rightarrow \quad (((n * w) \div r)$
$\quad = \quad \textbf{if } w < r \textbf{ then } n - 1$
$\quad\quad \textbf{else } n \textbf{ endif})$

THEOREM: nsig*-upper-bound-lemma2
$((n \in \mathbf{N}) \wedge (w \not\simeq \mathtt{0}) \wedge (r \not\simeq \mathtt{0}) \wedge \text{rate-proximity}\,(w,\, r) \wedge (n < \mathtt{18}))$
$\rightarrow \quad (n \not< ((n * w) \div r))$

THEOREM: nsig*-lower-bound-lemma1
$((r \not\simeq \mathtt{0}) \wedge (w \not< \mathit{tr\text{-}ts}))$
$\rightarrow \quad ((((n * w) - \mathit{tr\text{-}ts}) \div r) \not< (((n - 1) * w) \div r))$

THEOREM: lessp-times
$(n \not\simeq \mathtt{0}) \rightarrow ((n * w) \not< w)$

THEOREM: plus-cancellation
$((i + j) = (i + k)) = (\text{fix}\,(j) = \text{fix}\,(k))$

DEFINITION: $\text{boolp}\,(x) = ((x = \mathbf{t}) \vee (x = \mathbf{f}))$

EVENT: Disable boolp.

THEOREM: boolp-t
$(\text{boolp}\,(x) \wedge x) \rightarrow ((\mathbf{t} = x) = \mathbf{t})$

DEFINITION: $\text{b-not}\,(x) = (\neg\, x)$

EVENT: Disable b-not.

DEFINITION:
b-xor $(x,\, y)$
$=$ **if** x **then** $\neg\, y$
elseif y **then t**
else f endif

EVENT: Disable b-xor.

THEOREM: b-xor-b-not
b-xor $(x,$ b-not $(x))$ $\wedge$ b-xor (b-not $(x),\, x)$

DEFINITION:
smooth $(\mathit{prev\text{-}val},\, \mathit{lst})$
$=$ **if** $\mathit{lst} \simeq$ **nil then nil**
elseif b-xor $(\mathit{prev\text{-}val},$ car $(\mathit{lst}))$
then cons $(\mathtt{'q},$ smooth $(\text{car}\,(\mathit{lst}),$ cdr $(\mathit{lst})))$
else cons $(\text{car}\,(\mathit{lst}),$ smooth $(\text{car}\,(\mathit{lst}),$ cdr $(\mathit{lst})))$ **endif**

DEFINITION:
reconcile-signals $(a,\ b)$
$=$ **if** $a = b$ **then** a
else 'q **endif**

DEFINITION:
sig $(lst,\ ts,\ tr,\ w)$
$=$ **if** $lst \simeq$ **nil then** 'q
elseif $(ts + w) < tr$
then reconcile-signals (car (lst), sig (cdr (lst), $ts + w$, tr, w))
else car (lst) **endif**

DEFINITION:
endp $(lst,\ ts,\ tr,\ w)$
$=$ **if** $lst \simeq$ **nil then t**
elseif $(ts + w) < tr$ **then** endp (cdr (lst), $ts + w$, tr, w)
else f endif

DEFINITION:
lst+ $(lst,\ ts,\ nxtr,\ w)$
$=$ **if** $lst \simeq$ **nil then** lst
elseif $nxtr < (ts + w)$ **then** lst
else lst+ (cdr (lst), $ts + w$, $nxtr$, w) **endif**

DEFINITION:
ts+ $(lst,\ ts,\ nxtr,\ w)$
$=$ **if** $lst \simeq$ **nil then** ts
elseif $nxtr < (ts + w)$ **then** ts
else ts+ (cdr (lst), $ts + w$, $nxtr$, w) **endif**

THEOREM: lst+-weakly-shortens-lst
count $(lst) \not< $ count (lst+ $(lst,\ ts,\ tr,\ w)$)

THEOREM: ts+-increases-tr
((¬ endp $(lst,\ ts,\ r + tr,\ w)$)
$\wedge$ $(r \not\simeq 0)$
$\wedge$ (count (lst) = count (lst+ $(lst,\ ts,\ r + tr,\ w)$)))
$\rightarrow$ ((((w + ts+ $(lst,\ ts,\ r + tr,\ w)$) $-$ $(r + tr)$)
$<$ $((ts + w) - tr)$)
$=$ t)

DEFINITION:
warp $(lst,\ ts,\ tr,\ w,\ r)$
$=$ **if** $(r \simeq 0) \vee$ endp $(lst,\ ts,\ tr + r,\ w)$ **then nil**
else cons (sig $(lst,\ ts,\ tr + r,\ w)$,

warp (lst+ (*lst*, *ts*, *tr* + *r*, *w*),
ts+ (*lst*, *ts*, *tr* + *r*, *w*),
tr + *r*,
w,
r)) **endif**

DEFINITION:
det (*lst*, *oracle*)
= **if** *lst* ≃ **nil** **then** *lst*
elseif car (*lst*) = 'q
then cons (**if** car (*oracle*) **then t**
else f endif,
det (cdr (*lst*), cdr (*oracle*)))
else cons (car (*lst*), det (cdr (*lst*), *oracle*)) **endif**

DEFINITION:
async (*lst*, *ts*, *tr*, *w*, *r*, *oracle*) = det (warp (smooth (**t**, *lst*), *ts*, *tr*, *w*, *r*), *oracle*)

DEFINITION:
listn (*n*, *value*)
= **if** *n* ≃ 0 **then nil**
else cons (*value*, listn (*n* − 1, *value*)) **endif**

DEFINITION:
csig (*prev-signal*, *bit*)
= **if** *bit* **then** *prev-signal*
else b-not (*prev-signal*) **endif**

DEFINITION:
cell (*prev-signal*, *n1*, *n2*, *bit*)
= app (listn (*n1*, b-not (*prev-signal*)), listn (*n2*, csig (*prev-signal*, *bit*)))

DEFINITION:
cells (*prev-signal*, *n1*, *n2*, *msg*)
= **if** *msg* ≃ **nil** **then nil**
else app (cell (*prev-signal*, *n1*, *n2*, car (*msg*)),
cells (csig (*prev-signal*, car (*msg*)), *n1*, *n2*, cdr (*msg*))) **endif**

DEFINITION:
send (*msg*, *pad1*, *n1*, *n2*, *pad2*)
= app (listn (*pad1*, **t**), app (cells (**t**, *n1*, *n2*, *msg*), listn (*pad2*, **t**)))

DEFINITION:
scan (*prev-signal*, *lst*)
= **if** *lst* ≃ **nil** **then nil**
elseif b-xor (*prev-signal*, car (*lst*)) **then** *lst*
else scan (*prev-signal*, cdr (*lst*)) **endif**

DEFINITION:
cdrn $(n,\ lst)$
= **if** $n \simeq$ `0` **then** lst
else cdrn $(n - 1,$ cdr $(lst))$ **endif**

DEFINITION: nth $(n,\ lst)$ = car (cdrn $(n,\ lst)$)

DEFINITION:
recv-bit $(k,\ lst)$
= **if** b-xor (car (lst), nth $(k,\ lst)$) **then t**
else f endif

DEFINITION:
recv $(i,\ flg,\ k,\ lst)$
= **if** $i \simeq$ `0` **then nil**
else cons (recv-bit $(k,$ scan $(flg,\ lst))$,
recv $(i - 1,$
nth $(k,$ scan $(flg,\ lst))$,
k,
cdrn $(k,$ scan $(flg,\ lst))))$ **endif**

DEFINITION:
bvp (x)
= **if** $x \simeq$ **nil then** $x =$ **nil**
else boolp (car (x)) $\wedge$ bvp (cdr (x)) **endif**

```
#|
We can now state the top level theorem without proof.

(prove-lemma top nil
  (implies (and (bvp msg)
                (numberp ts)
                (numberp tr)
                (not (zerop w))
                (not (zerop r))
                (not (lessp tr ts))
                (lessp tr (plus ts w))
                (rate-proximity w r)
                (numberp p1))
           (equal (recv (len msg)
                        t
                        10
                        (async (send msg p1 5 13 p2)
                               ts tr w r oracle))
                  msg)))
```

```
We will prove this by first massaging it into a different form,
suitable for induction.  The inductive form shall be called loop
instead of top.

|#
```

EVENT: Disable listn.

EVENT: Disable *1*listn.

EVENT: Disable cell.

EVENT: Disable *1*cell.

EVENT: Disable csig.

EVENT: Disable *1*csig.

EVENT: Disable boolp.

EVENT: Disable smooth.

EVENT: Disable warp.

EVENT: Disable det.

EVENT: Disable recv-bit.

EVENT: Disable scan.

EVENT: Disable send.

THEOREM: b-xor-x-x
$\neg$ b-xor (x, x)

```
; Our first goal will be to transform the recv-async-send theorem
```

```
; into the form we will prove inductively.  This is done by moving
; the initial header out through the smooth, warp, det, and recv, milking
; a 'q out of the smooth and easing it through the warp to nestle a
; cdrn around the smooth and leaving the det hanging on a q string.

; Because the warping is harder than the smoothing, we'll do it first.
```

DEFINITION:
$\text{lst}^*(lst,\ ts,\ tr,\ w,\ r)$
$=$ **if** $(r \simeq 0) \vee \text{endp}(lst,\ ts,\ tr + r,\ w)$ **then** lst
else $\text{lst}^*(\text{lst}+(lst,\ ts,\ tr + r,\ w),$
$\text{ts}+(lst,\ ts,\ tr + r,\ w),$
$tr + r,$
$w,$
$r)$ **endif**

DEFINITION:
$\text{ts}^*(lst,\ ts,\ tr,\ w,\ r)$
$=$ **if** $(r \simeq 0) \vee \text{endp}(lst,\ ts,\ tr + r,\ w)$ **then** ts
else $\text{ts}^*(\text{lst}+(lst,\ ts,\ tr + r,\ w),$
$\text{ts}+(lst,\ ts,\ tr + r,\ w),$
$tr + r,$
$w,$
$r)$ **endif**

DEFINITION:
$\text{tr}^*(lst,\ ts,\ tr,\ w,\ r)$
$=$ **if** $(r \simeq 0) \vee \text{endp}(lst,\ ts,\ tr + r,\ w)$ **then** tr
else $\text{tr}^*(\text{lst}+(lst,\ ts,\ tr + r,\ w),$
$\text{ts}+(lst,\ ts,\ tr + r,\ w),$
$tr + r,$
$w,$
$r)$ **endif**

THEOREM: not-lessp-ts+
$((tr \not< ts) \wedge (w \not\simeq 0)) \rightarrow (tr \not< \text{ts}+(lst,\ ts,\ tr,\ w))$

THEOREM: lessp-ts+
$((\neg\ \text{endp}(lst,\ ts,\ tr,\ w)) \wedge (tr \not< ts) \wedge (w \not\simeq 0))$
$\rightarrow\quad (tr < (w + \text{ts}+(lst,\ ts,\ tr,\ w)))$

THEOREM: lst+-app
$((\neg\ \text{endp}(lst1,\ ts,\ tr+,\ w)) \wedge (w \not\simeq 0))$
$\rightarrow\quad (\text{lst}+(\text{app}(lst1,\ lst2),\ ts,\ tr+,\ w) = \text{app}(\text{lst}+(lst1,\ ts,\ tr+,\ w),\ lst2))$

THEOREM: ts+-app
$((\neg \text{endp}(\mathit{lst1}, \mathit{ts}, \mathit{tr}+, w)) \wedge (w \not\simeq \mathtt{0}))$
$\rightarrow \quad (\text{ts}+(\text{app}(\mathit{lst1}, \mathit{lst2}), \mathit{ts}, \mathit{tr}+, w) = \text{ts}+(\mathit{lst1}, \mathit{ts}, \mathit{tr}+, w))$

THEOREM: sig-app
$((\neg \text{endp}(\mathit{lst1}, \mathit{ts}, \mathit{tr}+, w)) \wedge (w \not\simeq \mathtt{0}))$
$\rightarrow \quad (\text{sig}(\text{app}(\mathit{lst1}, \mathit{lst2}), \mathit{ts}, \mathit{tr}+, w) = \text{sig}(\mathit{lst1}, \mathit{ts}, \mathit{tr}+, w))$

THEOREM: endp-app
$((\neg \text{endp}(\mathit{lst1}, \mathit{ts}, \mathit{tr}+, w)) \wedge (w \not\simeq \mathtt{0}))$
$\rightarrow \quad (\neg \text{endp}(\text{app}(\mathit{lst1}, \mathit{lst2}), \mathit{ts}, \mathit{tr}+, w))$

DEFINITION: $\text{target}(x) = x$

EVENT: Disable target.

THEOREM: warp-app
$((\mathit{ts} \in \mathbf{N})$
$\wedge \quad (\mathit{tr} \in \mathbf{N})$
$\wedge \quad (\mathit{tr} \not< \mathit{ts})$
$\wedge \quad (\mathit{tr} < (\mathit{ts} + w))$
$\wedge \quad (w \not\simeq \mathtt{0})$
$\wedge \quad (r \not\simeq \mathtt{0}))$
$\rightarrow \quad (\text{target}(\text{warp}(\text{app}(\mathit{lst1}, \mathit{lst2}), \mathit{ts}, \mathit{tr}, w, r))$
$= \quad \text{app}(\text{warp}(\mathit{lst1}, \mathit{ts}, \mathit{tr}, w, r),$
$\text{warp}(\text{app}(\text{lst}^*(\mathit{lst1}, \mathit{ts}, \mathit{tr}, w, r), \mathit{lst2}),$
$\text{ts}^*(\mathit{lst1}, \mathit{ts}, \mathit{tr}, w, r),$
$\text{tr}^*(\mathit{lst1}, \mathit{ts}, \mathit{tr}, w, r),$
$w,$
$r)))$

```
; The above lemma, when applied, will generate (warp (listn p1 t) ...)
; and (lst* (listn p1 t) ...) which we now simplify.
```

DEFINITION:
$\text{nlst}+(n, \mathit{ts}, \mathit{tr}, w)$
$=$ **if** $n \simeq \mathtt{0}$ **then** n
elseif $\mathit{tr} < (\mathit{ts} + w)$ **then** n
else $\text{nlst}+(n - 1, \mathit{ts} + w, \mathit{tr}, w)$ **endif**

THEOREM: len-lst+
$\text{len}(\text{lst}+(\mathit{lst}, \mathit{ts}, \mathit{tr}, w)) = \text{nlst}+(\text{len}(\mathit{lst}), \mathit{ts}, \mathit{tr}, w)$

DEFINITION:
nts+ $(n,\ ts,\ tr,\ w)$
= **if** $n \simeq \mathtt{0}$ **then** ts
elseif $tr < (ts + w)$ **then** ts
else nts+ $(n - 1,\ ts + w,\ tr,\ w)$ **endif**

THEOREM: len-ts+
ts+ $(lst,\ ts,\ tr,\ w)$ = nts+ $(\text{len}\,(lst),\ ts,\ tr,\ w)$

DEFINITION:
nendp $(n,\ ts,\ tr,\ w)$
= **if** $n \simeq \mathtt{0}$ **then t**
elseif $(ts + w) < tr$ **then** nendp $(n - 1,\ ts + w,\ tr,\ w)$
else f endif

THEOREM: len-endp
endp $(lst,\ ts,\ tr,\ w)$ = nendp $(\text{len}\,(lst),\ ts,\ tr,\ w)$

THEOREM: lessp-nlst+
$n \not< \text{nlst+}\,(n,\ ts,\ tr,\ w)$

THEOREM: nlst+-equal-n
$(\text{nlst+}\,(n,\ ts,\ r + tr,\ w) = n)$
= $((n \simeq \mathtt{0}) \vee ((r + tr) < (ts + w)))$

DEFINITION:
nlst* $(n,\ ts,\ tr,\ w,\ r)$
= **if** $(r \simeq \mathtt{0}) \vee$ nendp $(n,\ ts,\ tr + r,\ w)$ **then** n
else nlst* (nlst+ $(n,\ ts,\ tr + r,\ w)$,
nts+ $(n,\ ts,\ tr + r,\ w)$,
$tr + r$,
w,
r) **endif**

THEOREM: len-lst*
len (lst* $(lst,\ ts,\ tr,\ w,\ r)$) = nlst* $(\text{len}\,(lst),\ ts,\ tr,\ w,\ r)$

DEFINITION:
nts* $(n,\ ts,\ tr,\ w,\ r)$
= **if** $(r \simeq \mathtt{0}) \vee$ nendp $(n,\ ts,\ tr + r,\ w)$ **then** ts
else nts* (nlst+ $(n,\ ts,\ tr + r,\ w)$,
nts+ $(n,\ ts,\ tr + r,\ w)$,
$tr + r$,
w,
r) **endif**

THEOREM: len-ts*
$\text{ts*}(lst,\ ts,\ tr,\ w,\ r) = \text{nts*}(\text{len}(lst),\ ts,\ tr,\ w,\ r)$

DEFINITION:
$\text{ntr*}(n,\ ts,\ tr,\ w,\ r)$
$=$ **if** $(r \simeq \texttt{0}) \vee \text{nendp}(n,\ ts,\ tr + r,\ w)$ **then** tr
else $\text{ntr*}(\text{nlst+}(n,\ ts,\ tr + r,\ w),$
$\text{nts+}(n,\ ts,\ tr + r,\ w),$
$tr + r,$
$w,$
$r)$ **endif**

THEOREM: len-tr*
$\text{tr*}(lst,\ ts,\ tr,\ w,\ r) = \text{ntr*}(\text{len}(lst),\ ts,\ tr,\ w,\ r)$

THEOREM: len-listn
$\text{len}(\text{listn}(n,\ flg)) = \text{fix}(n)$

DEFINITION:
$\text{lastn}(n,\ lst)$
$=$ **if** $n = \text{len}(lst)$ **then** lst
elseif $lst \simeq$ **nil then** lst
else $\text{lastn}(n,\ \text{cdr}(lst))$ **endif**

DEFINITION:
$\text{tailp}(x,\ lst)$
$=$ **if** $x = lst$ **then t**
elseif $lst \simeq$ **nil then f**
else $\text{tailp}(x,\ \text{cdr}(lst))$ **endif**

THEOREM: tailp-transitive
$(\text{tailp}(x,\ y) \wedge \text{tailp}(y,\ z)) \rightarrow \text{tailp}(x,\ z)$

THEOREM: tailp-lst+
$\text{tailp}(\text{lst+}(lst,\ ts,\ tr,\ w),\ lst)$

THEOREM: tailp-lst*
$\text{tailp}(\text{lst*}(lst,\ ts,\ tr,\ w,\ r),\ lst)$

THEOREM: len-lastn
$\text{len}(\text{lastn}(n,\ lst))$
$=$ **if** $\text{len}(lst) < n$ **then** $\texttt{0}$
else $\text{fix}(n)$ **endif**

THEOREM: tailp-implies-lastn-len
$\text{tailp}(x,\ y) \rightarrow (\text{lastn}(\text{len}(x),\ y) = x)$

THEOREM: lst*-is-lastn
$\text{lastn}(\text{len}(\text{lst}^*(lst,\ ts,\ tr,\ w,\ r)),\ lst) = \text{lst}^*(lst,\ ts,\ tr,\ w,\ r)$

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

THEOREM: lastn-nil
$(\text{properp}(lst) \wedge (\text{len}(lst) < n)) \rightarrow (\text{lastn}(n,\ lst) = \textbf{nil})$

THEOREM: properp-listn
$\text{properp}(\text{listn}(n,\ flg))$

THEOREM: lastn-listn
$((n \in \mathbf{N}) \wedge (k \in \mathbf{N}) \wedge (k \not< n))$
$\rightarrow$ $(\text{lastn}(n,\ \text{listn}(k,\ flg)) = \text{listn}(n,\ flg))$

THEOREM: lessp-nlst*
$n \not< \text{nlst}^*(n,\ ts,\ tr,\ w,\ r)$

THEOREM: lst*-listn
$(n \in \mathbf{N})$
$\rightarrow$ $(\text{lst}^*(\text{listn}(n,\ flg),\ ts,\ tr,\ w,\ r) = \text{listn}(\text{nlst}^*(n,\ ts,\ tr,\ w,\ r),\ flg))$

DEFINITION:
$\text{n}^*(n,\ ts,\ tr,\ w,\ r)$
$=$ **if** $(r \simeq \mathtt{0}) \vee \text{nendp}(n,\ ts,\ tr + r,\ w)$ **then** $\mathtt{0}$
else $1 + \text{n}^*(\text{nlst+}(n,\ ts,\ tr + r,\ w),$
$\text{nts+}(n,\ ts,\ tr + r,\ w),$
$tr + r,$
$w,$
$r)$ **endif**

THEOREM: not-lessp-nts+
$((tr \not< ts) \wedge (w \not\simeq \mathtt{0})) \rightarrow (tr \not< \text{nts+}(n,\ ts,\ tr,\ w))$

THEOREM: lessp-nts+
$((\neg\ \text{nendp}(n,\ ts,\ tr,\ w)) \wedge (tr \not< ts) \wedge (w \not\simeq \mathtt{0}))$
$\rightarrow$ $(tr < (w + \text{nts+}(n,\ ts,\ tr,\ w)))$

THEOREM: lst+-listn
$((n \in \mathbf{N}) \wedge (\neg\ \text{nendp}(n,\ ts,\ tr+,\ w)) \wedge (w \not\simeq \mathtt{0}))$
$\rightarrow$ $(\text{lst+}(\text{listn}(n,\ flg),\ ts,\ tr+,\ w) = \text{listn}(\text{nlst+}(n,\ ts,\ tr+,\ w),\ flg))$

THEOREM: sig-listn
$$
\begin{array}{ll}
((r \neq \mathtt{0}) & \\
\wedge & (r \in \mathbf{N}) \\
\wedge & (\neg\, \mathrm{nendp}\,(n,\, ts,\, r + tr,\, w)) \\
\wedge & (n \in \mathbf{N}) \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & (w \neq \mathtt{0}) \\
\wedge & (w \in \mathbf{N})) \\
\rightarrow & (\mathrm{sig}\,(\mathrm{listn}\,(n,\, \mathit{flg}),\, ts,\, r + tr,\, w) = \mathit{flg})
\end{array}
$$

THEOREM: warp-listn
$$
\begin{array}{ll}
((n \in \mathbf{N}) & \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & (w \not\simeq \mathtt{0}) \\
\wedge & (r \not\simeq \mathtt{0})) \\
\rightarrow & (\mathrm{warp}\,(\mathrm{listn}\,(n,\, \mathit{flg}),\, ts,\, tr,\, w,\, r) = \mathrm{listn}\,(\mathrm{n}^*\,(n,\, ts,\, tr,\, w,\, r),\, \mathit{flg}))
\end{array}
$$

THEOREM: lst+-app-gap
$$
\begin{array}{ll}
((ts \in \mathbf{N}) \wedge (tr{+} \in \mathbf{N}) \wedge \mathrm{nendp}\,(\mathrm{len}\,(\mathit{pad}),\, ts,\, tr{+},\, w)) & \\
\rightarrow & (\mathrm{lst}{+}\,(\mathrm{app}\,(\mathit{pad},\, \mathit{lst}),\, ts,\, tr{+},\, w) \\
& \quad = \quad \mathrm{lst}{+}\,(\mathit{lst},\, ts + (\mathrm{len}\,(\mathit{pad}) * w),\, tr{+},\, w))
\end{array}
$$

THEOREM: nts+-app-gap
$$
\begin{array}{ll}
((ts \in \mathbf{N}) \wedge (tr{+} \in \mathbf{N}) \wedge (k \in \mathbf{N}) \wedge \mathrm{nendp}\,(k,\, ts,\, tr{+},\, w)) & \\
\rightarrow & (\mathrm{nts}{+}\,(n + k,\, ts,\, tr{+},\, w) = \mathrm{nts}{+}\,(n,\, ts + (k * w),\, tr{+},\, w))
\end{array}
$$

THEOREM: warp-app-gap
$$
\begin{array}{ll}
((ts \in \mathbf{N}) \wedge (r \not\simeq \mathtt{0}) \wedge \mathrm{nendp}\,(\mathrm{len}\,(\mathit{pad}),\, ts,\, r + tr,\, w)) & \\
\rightarrow & (\mathrm{warp}\,(\mathrm{app}\,(\mathit{pad},\, \mathit{lst}),\, ts,\, tr,\, w,\, r) \\
& \quad = \quad \textbf{if}\ \mathrm{nendp}\,(\mathrm{len}\,(\mathit{pad}) + \mathrm{len}\,(\mathit{lst}),\, ts,\, r + tr,\, w)\ \textbf{then nil} \\
& \qquad \textbf{else}\ \mathrm{cons}\,(\mathrm{sig}\,(\mathrm{app}\,(\mathit{pad},\, \mathit{lst}),\, ts,\, r + tr,\, w), \\
& \qquad\qquad \mathrm{warp}\,(\mathrm{lst}{+}\,(\mathit{lst}, \\
& \qquad\qquad\qquad ts + (\mathrm{len}\,(\mathit{pad}) * w), \\
& \qquad\qquad\qquad r + tr, \\
& \qquad\qquad\qquad w), \\
& \qquad\qquad\quad \mathrm{nts}{+}\,(\mathrm{len}\,(\mathit{lst}), \\
& \qquad\qquad\qquad ts + (\mathrm{len}\,(\mathit{pad}) * w), \\
& \qquad\qquad\qquad r + tr, \\
& \qquad\qquad\qquad w),
\end{array}
$$

$r + tr$,
w,
r)) **endif**)

THEOREM: nendp-nlst*
((*ts* ∈ **N**)
∧ (*tr* ∈ **N**)
∧ (*r* ≠ 0)
∧ (*w* ≠ 0)
∧ (*tr* ≮ *ts*)
∧ (*tr* < (*ts* + *w*)))
→ nendp (nlst* (*k*, *ts*, *tr*, *w*, *r*), nts* (*k*, *ts*, *tr*, *w*, *r*), *r* + ntr* (*k*, *ts*, *tr*, *w*, *r*), *w*)

DEFINITION:
nqg (*k*, *ts*, *tr*, *w*, *r*)
= **if** (*r* + *tr*) < (*w* + *ts* + (*w* ∗ *k*))
then if (*r* + *r* + *tr*) < (*w* + *ts* + (*w* ∗ *k*)) **then** 3
else 2 **endif**
else 1 **endif**

```
; case e

#|
(defn dwg (k ts tr w r)

; I have gotten this function wrong more times than I care to admit.
; I won't even try to explain the "logic" behind this defn.

  (let ((ts1 (plus ts (times k w)))
        (ts2 (plus ts w (times k w)))
        (ts3 (plus ts w w (times k w)))
        (tr1 (plus r tr))
        (tr2 (plus r r tr)))
    (cond ((lessp tr1 ts2)
           (if (lessp tr2 ts3) 0 1))
          ((equal tr1 ts2) 0)
          ((and (lessp ts2 tr1) (lessp tr1 ts3)) 0)
          ((lessp tr1 ts3) 0)
          (t 1))))
|#
```

DEFINITION:
dwg (*k*, *ts*, *tr*, *w*, *r*)
= **if** (*r* + *tr*) < (*ts* + *w* + (*k* ∗ *w*))

then if $(r + r + tr) < (ts + w + w + (k * w))$ **then** 0
else 1 **endif**
elseif $(r + tr) = (ts + w + (k * w))$ **then** 0
elseif $((ts + w + (k * w)) < (r + tr))$
$\wedge \quad ((r + tr) < (ts + w + w + (k * w)))$ **then** 0
elseif $(r + tr) < (ts + w + w + (k * w))$ **then** 0
else 1 **endif**

DEFINITION:
$\mathrm{tsg}(k, ts, tr, w, r) = (ts + (w * k) + w + (w * \mathrm{dwg}(k, ts, tr, w, r)))$

```
; the eaten element of n2
```

DEFINITION:
$\mathrm{trg}(k, ts, tr, w, r) = (tr + (r * \mathrm{nqg}(k, ts, tr, w, r)))$

```
; the qs
```

THEOREM: not-lessp-ntr*-nts*
$((ts \in \mathbf{N}) \wedge (tr \in \mathbf{N}) \wedge (tr \not< ts) \wedge (tr < (ts + w)) \wedge (w \not\simeq 0))$
$\rightarrow \quad (\mathrm{ntr}^*(k, ts, tr, w, r) \not< \mathrm{nts}^*(k, ts, tr, w, r))$

THEOREM: lessp-ntr*-nts*
$((ts \in \mathbf{N}) \wedge (tr \in \mathbf{N}) \wedge (tr \not< ts) \wedge (tr < (ts + w)) \wedge (w \not\simeq 0))$
$\rightarrow \quad (\mathrm{ntr}^*(k, ts, tr, w, r) < (w + \mathrm{nts}^*(k, ts, tr, w, r)))$

THEOREM: nendp-is-usually-f
$((ts \in \mathbf{N})$
$\wedge \quad (tr \in \mathbf{N})$
$\wedge \quad (w \neq 0)$
$\wedge \quad (w \in \mathbf{N})$
$\wedge \quad (r \neq 0)$
$\wedge \quad (r \in \mathbf{N})$
$\wedge \quad (tr < (ts + w))$
$\wedge \quad (w < (2 * r))$
$\wedge \quad (r < (2 * w))$
$\wedge \quad (2 < k))$
$\rightarrow \quad (\neg\, \mathrm{nendp}(k, ts, r + tr, w))$

```
; Note: The following is the nts+ analogue of ts+-app.  However the
; rule is a little strange because one naturally would have written
; (nts+ (plus k rest) ...) since k is here the len of the initial
; portion of the list being scanned.  But in my actual application,
; the plus has been commuted so I commute it here.
```

THEOREM: lessp-plus-nts*-times-w-nlst*-plus-r-ntr*-lemma
$(\text{nendp}\,(k,\, ts,\, r + tr,\, w)$
$\wedge \quad (ts \in \mathbf{N})$
$\wedge \quad (tr \in \mathbf{N})$
$\wedge \quad (w \neq 0)$
$\wedge \quad (w \in \mathbf{N})$
$\wedge \quad (r \neq 0)$
$\wedge \quad (r \in \mathbf{N})$
$\wedge \quad ((r + tr) \not< ts)$
$\wedge \quad (tr < (ts + w)))$
$\rightarrow \quad ((r + tr) \not< (ts + (k * w)))$

THEOREM: not-lessp-plus-r-ntr*-plus-nts*-times-w-nlst*
$((ts \in \mathbf{N})$
$\wedge \quad (tr \in \mathbf{N})$
$\wedge \quad (w \not\simeq 0)$
$\wedge \quad (r \not\simeq 0)$
$\wedge \quad (tr \not< ts)$
$\wedge \quad (tr < (ts + w)))$
$\rightarrow \quad ((r + \text{ntr}^*\,(k,\, ts,\, tr,\, w,\, r))$
$\not< \quad (\text{nts}^*\,(k,\, ts,\, tr,\, w,\, r) + (w * \text{nlst}^*\,(k,\, ts,\, tr,\, w,\, r))))$

```
; That takes care of ts'<=tr'.  Now I'll get tr' < ts'+w.  In our case,
; ts' is (nts+ m ts'' tr' w r).  Now nts+ is going to push ts'' to within
; w or tr' unless m runs out.  So we have to make an argument that
; m is big enough.  It turns out that we know m is at least 18
; (because it is the encoding of a listp msg) and so if we assume w and r
; are within a factor of 2 of eachother, we can manage.

; These helper lemmas are generated mechanically by just grabbing
; the unproved goals generated in the next real theorem and forcing
; the expansion of the relevant terms.  Chances are there are some
; irrelevant hypotheses.  Some helper's subsumed others or changed the
; course of the proof so as to make others irrelevant.  Thus, their
; numbering is not consecutive.
```

THEOREM: helper1
$((ts \in \mathbf{N})$
$\wedge \quad (tr \in \mathbf{N})$
$\wedge \quad (w \neq 0)$
$\wedge \quad (w \in \mathbf{N})$
$\wedge \quad (r \neq 0)$
$\wedge \quad (r \in \mathbf{N})$
$\wedge \quad (tr \not< ts)$

$$
\begin{array}{ll}
\wedge & (tr < (ts + w)) \\
\wedge & (w < (2 * r)) \\
\wedge & (r < (2 * w)) \\
\wedge & \text{listp}\,(rest) \\
\wedge & ((ts + w + w) < (r + tr)) \\
\wedge & ((ts + w) < (r + tr)) \\
\wedge & ((r + tr) \not< (ts + w + w))) \\
\rightarrow & (\text{warp}\,(\text{lst+}\,(rest,\ ts + w + w,\ r + tr,\ w), \\
 & \qquad \text{nts+}\,(\text{len}\,(rest),\ ts + w + w,\ r + tr,\ w), \\
 & \qquad r + tr, \\
 & \qquad w, \\
 & \qquad r) \\
 & = \quad \text{warp}\,(rest,\ ts + w + w,\ r + tr,\ w,\ r))
\end{array}
$$

THEOREM: helper5

$$
\begin{array}{ll}
((ts \in \mathbf{N}) & \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (k \in \mathbf{N}) \\
\wedge & (w \neq 0) \\
\wedge & (w \in \mathbf{N}) \\
\wedge & (r \neq 0) \\
\wedge & (r \in \mathbf{N}) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & (w < (2 * r)) \\
\wedge & (r < (2 * w)) \\
\wedge & \text{listp}\,(rest) \\
\wedge & \text{listp}\,(\text{cdr}\,(rest)) \\
\wedge & (((k - 1) - 1) = 0) \\
\wedge & ((ts + w + w) < (r + tr)) \\
\wedge & ((ts + w) < (r + tr)) \\
\wedge & ((r + tr) < (ts + w + (k * w))) \\
\wedge & ((r + r + tr) \not< (ts + w + (k * w))) \\
\wedge & ((r + r + tr) < (ts + w + w + (k * w)))) \\
\rightarrow & (\text{warp}\,(\text{cons}\,(\texttt{'q},\ rest),\ ts + (k * w),\ r + tr,\ w,\ r) \\
 & = \quad \text{cons}\,(\texttt{'q}, \\
 & \qquad \text{warp}\,(rest,\ ts + w + (k * w),\ tr + (2 * r),\ w,\ r)))
\end{array}
$$

THEOREM: helper7

$$
\begin{array}{ll}
((ts \in \mathbf{N}) & \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (w \neq 0) \\
\wedge & (w \in \mathbf{N}) \\
\wedge & (r \neq 0)
\end{array}
$$

$$
\begin{array}{ll}
\wedge & (r \in \mathbf{N}) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & (w < (2 * r)) \\
\wedge & (r < (2 * w)) \\
\wedge & \mathrm{listp}\,(rest) \\
\wedge & \mathrm{listp}\,(\mathrm{cdr}\,(rest)) \\
\wedge & ((ts + w) < (r + tr)) \\
\wedge & ((r + tr) \not< (ts + w + w)) \\
\wedge & ((r + tr) < (ts + w + w + w))) \\
\rightarrow & (\mathrm{warp}\,(\mathrm{cons}\,(flg1, \mathrm{cons}\,(\texttt{'q}, rest)), ts, tr, w, r) \\
& \quad = \quad \mathrm{cons}\,(\texttt{'q}, \mathrm{warp}\,(rest, ts + w + w, r + tr, w, r)))
\end{array}
$$

THEOREM: helper8

$$
\begin{array}{ll}
((ts \in \mathbf{N}) & \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (w \neq 0) \\
\wedge & (w \in \mathbf{N}) \\
\wedge & (r \neq 0) \\
\wedge & (r \in \mathbf{N}) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & (w < (2 * r)) \\
\wedge & (r < (2 * w)) \\
\wedge & \mathrm{listp}\,(rest) \\
\wedge & \mathrm{listp}\,(\mathrm{cdr}\,(rest)) \\
\wedge & ((ts + w) < (r + tr)) \\
\wedge & ((r + tr) \not< (ts + w)) \\
\wedge & ((r + tr) \neq (ts + w)) \\
\wedge & ((r + tr) \not< (ts + w + w))) \\
\rightarrow & (\mathrm{warp}\,(\mathrm{cons}\,(\texttt{'q}, rest), ts, tr, w, r) \\
& \quad = \quad \mathrm{cons}\,(\texttt{'q}, \mathrm{warp}\,(\mathrm{cdr}\,(rest), ts + w + w, r + tr, w, r)))
\end{array}
$$

THEOREM: helper9

$$
\begin{array}{ll}
((ts \in \mathbf{N}) & \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (w \neq 0) \\
\wedge & (w \in \mathbf{N}) \\
\wedge & (r \neq 0) \\
\wedge & (r \in \mathbf{N}) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & (w < (r + r)) \\
\wedge & (r < (2 * w))
\end{array}
$$

$\wedge$ listp (*rest*)
$\wedge$ listp (cdr (*rest*))
$\wedge$ $((ts + w) < (r + tr))$
$\wedge$ $((r + tr) < (ts + w + w))$
$\wedge$ $((r + r + tr) \not< (ts + w + w))$
$\wedge$ $((r + r + tr) \not< (ts + w + w + w)))$
$\rightarrow$ (warp (cons (*flg1*, cons (`'q`, *rest*)), ts, tr, w, r)
= cons (`'q`,
cons (`'q`,
warp (cdr (*rest*),
$ts + w + w + w$,
$r + r + tr$,
w,
r))))

THEOREM: helper10
$((ts \in \mathbf{N})$
$\wedge$ $(tr \in \mathbf{N})$
$\wedge$ $(w \neq \mathtt{0})$
$\wedge$ $(w \in \mathbf{N})$
$\wedge$ $(r \neq \mathtt{0})$
$\wedge$ $(r \in \mathbf{N})$
$\wedge$ $(tr \not< ts)$
$\wedge$ $(tr < (ts + w))$
$\wedge$ $(w < (r + r))$
$\wedge$ $(r < (\mathtt{2} * w))$
$\wedge$ listp (*rest*)
$\wedge$ listp (cdr (*rest*))
$\wedge$ $((ts + w) < (r + tr))$
$\wedge$ $((r + tr) < (ts + w + w))$
$\wedge$ $((r + r + tr) \not< (ts + w + w))$
$\wedge$ $((r + r + tr) < (ts + w + w + w)))$
$\rightarrow$ (warp (cons (*flg1*, cons (`'q`, *rest*)), ts, tr, w, r)
= cons (`'q`,
cons (`'q`, warp (*rest*, $ts + w + w$, $r + r + tr$, w, r))))

THEOREM: helper11
$((ts \in \mathbf{N})$
$\wedge$ $(tr \in \mathbf{N})$
$\wedge$ $(w \neq \mathtt{0})$
$\wedge$ $(w \in \mathbf{N})$
$\wedge$ $(r \neq \mathtt{0})$
$\wedge$ $(r \in \mathbf{N})$
$\wedge$ $(tr \not< ts)$

$$
\begin{array}{ll}
\wedge & (tr < (ts + w)) \\
\wedge & (w < (r + r)) \\
\wedge & (r < (\mathtt{2} * w)) \\
\wedge & \mathrm{listp}\,(rest) \\
\wedge & \mathrm{listp}\,(\mathrm{cdr}\,(rest)) \\
\wedge & ((ts + w) < (r + tr)) \\
\wedge & ((r + tr) < (ts + w + w)) \\
\wedge & ((r + r + tr) < (ts + w + w)) \\
\wedge & ((r + r + tr) < (ts + w + w + w))) \\
\rightarrow & (\mathrm{warp}\,(\mathrm{cons}\,(\mathit{flg1}, \mathrm{cons}\,(\mathtt{'q}, rest)), ts, tr, w, r) \\
& \quad = \quad \mathrm{cons}\,(\mathtt{'q}, \\
& \qquad\quad \mathrm{cons}\,(\mathtt{'q}, \\
& \qquad\qquad \mathrm{cons}\,(\mathtt{'q}, \\
& \qquad\qquad\quad \mathrm{warp}\,(rest, ts + w + w, tr + (\mathtt{3} * r), w, r)))))
\end{array}
$$

THEOREM: lessp-times-18
$\mathrm{listp}\,(msg) \rightarrow ((\mathtt{18} * \mathrm{len}\,(msg)) \not< \mathtt{18})$

THEOREM: listn-add1
$\mathrm{listn}\,(1 + n, \mathit{flg}) = \mathrm{cons}\,(\mathit{flg}, \mathrm{listn}\,(n, \mathit{flg}))$

THEOREM: helper14
$$
\begin{array}{ll}
((ts \in \mathbf{N}) & \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (w \neq \mathtt{0}) \\
\wedge & (w \in \mathbf{N}) \\
\wedge & (r \neq \mathtt{0}) \\
\wedge & (r \in \mathbf{N}) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & (w < (r + r)) \\
\wedge & (r < (\mathtt{2} * w)) \\
\wedge & \mathrm{listp}\,(rest) \\
\wedge & \mathrm{listp}\,(\mathrm{cdr}\,(rest)) \\
\wedge & ((r + tr) \not< (ts + w)) \\
\wedge & ((r + tr) < (ts + w + w))) \\
\rightarrow & (\mathrm{warp}\,(\mathrm{cons}\,(\mathtt{'q}, rest), ts, tr, w, r) \\
& \quad = \quad \mathrm{cons}\,(\mathtt{'q}, \mathrm{warp}\,(rest, ts + w, r + tr, w, r)))
\end{array}
$$

THEOREM: helper15
$$
\begin{array}{ll}
((ts \in \mathbf{N}) & \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (w \neq \mathtt{0}) \\
\wedge & (w \in \mathbf{N}) \\
\wedge & (r \neq \mathtt{0})
\end{array}
$$

$\wedge \quad (r \in \mathbf{N})$
$\wedge \quad (tr \not< ts)$
$\wedge \quad (tr < (ts + w))$
$\wedge \quad (w < (r + r))$
$\wedge \quad (r < (\mathtt{2} * w))$
$\wedge \quad \text{listp}\,(rest)$
$\wedge \quad \text{listp}\,(\text{cdr}\,(rest))$
$\wedge \quad ((r + tr) < (ts + w))$
$\wedge \quad ((r + r + tr) \not< (ts + w))$
$\wedge \quad ((r + r + tr) < (ts + w + w)))$
$\rightarrow \quad (\text{warp}\,(\text{cons}\,(\mathtt{'q}, rest), ts, tr, w, r)$
$\quad = \quad \text{cons}\,(\mathtt{'q}, \text{cons}\,(\mathtt{'q}, \text{warp}\,(rest, ts + w, r + r + tr, w, r))))$

THEOREM: warp-app-across-gap
$((ts \in \mathbf{N})$
$\wedge \quad (tr \in \mathbf{N})$
$\wedge \quad (k \in \mathbf{N})$
$\wedge \quad (w \not\simeq \mathtt{0})$
$\wedge \quad (r \not\simeq \mathtt{0})$
$\wedge \quad (tr \not< ts)$
$\wedge \quad (tr < (ts + w))$
$\wedge \quad (w < (\mathtt{2} * r))$
$\wedge \quad (r < (\mathtt{2} * w))$
$\wedge \quad \text{listp}\,(rest)$
$\wedge \quad \text{listp}\,(\text{cdr}\,(rest))$
$\wedge \quad \text{nendp}\,(k, ts, r + tr, w))$
$\rightarrow \quad (\text{warp}\,(\text{app}\,(\text{listn}\,(k, flg1), \text{cons}\,(\mathtt{'q}, rest)), ts, tr, w, r)$
$\quad = \quad \text{app}\,(\text{listn}\,(\text{nqg}\,(k, ts, tr, w, r), \mathtt{'q}),$
$\qquad \text{warp}\,(\text{cdrn}\,(\text{dwg}\,(k, ts, tr, w, r), rest),$
$\qquad\quad \text{tsg}\,(k, ts, tr, w, r),$
$\qquad\quad \text{trg}\,(k, ts, tr, w, r),$
$\qquad\quad w,$
$\qquad\quad r)))$

EVENT: Disable helper1.

EVENT: Disable helper5.

EVENT: Disable helper7.

EVENT: Disable helper8.

EVENT: Disable helper9.

EVENT: Disable helper10.

EVENT: Disable helper11.

EVENT: Disable helper14.

EVENT: Disable helper15.

EVENT: Disable nqg.

EVENT: Disable dwg.

EVENT: Disable tsg.

EVENT: Disable trg.

DEFINITION:
$\mathrm{nq}(n,\ ts,\ tr,\ w,\ r)$
$=\quad \mathrm{nqg}(\mathrm{nlst}^*(n,\ ts,\ tr,\ w,\ r),\ \mathrm{nts}^*(n,\ ts,\ tr,\ w,\ r),\ \mathrm{ntr}^*(n,\ ts,\ tr,\ w,\ r),\ w,\ r)$

DEFINITION:
$\mathrm{dw}(n,\ ts,\ tr,\ w,\ r)$
$=\quad \mathrm{dwg}(\mathrm{nlst}^*(n,\ ts,\ tr,\ w,\ r),\ \mathrm{nts}^*(n,\ ts,\ tr,\ w,\ r),\ \mathrm{ntr}^*(n,\ ts,\ tr,\ w,\ r),\ w,\ r)$

DEFINITION:
$\mathrm{ts}(n,\ ts,\ tr,\ w,\ r)$
$=\quad \mathrm{tsg}(\mathrm{nlst}^*(n,\ ts,\ tr,\ w,\ r),\ \mathrm{nts}^*(n,\ ts,\ tr,\ w,\ r),\ \mathrm{ntr}^*(n,\ ts,\ tr,\ w,\ r),\ w,\ r)$

DEFINITION:
$\mathrm{tr}(n,\ ts,\ tr,\ w,\ r)$
$=\quad \mathrm{trg}(\mathrm{nlst}^*(n,\ ts,\ tr,\ w,\ r),\ \mathrm{nts}^*(n,\ ts,\ tr,\ w,\ r),\ \mathrm{ntr}^*(n,\ ts,\ tr,\ w,\ r),\ w,\ r)$

THEOREM: lessp-2-len-implies-listps
$(2 < \mathrm{len}(x)) \rightarrow (\mathrm{listp}(x) \wedge \mathrm{listp}(\mathrm{cdr}(x)))$

THEOREM: warp-app-listn-q1
$((ts \in \mathbf{N})$
$\wedge\quad (tr \in \mathbf{N})$
$\wedge\quad (w \not\simeq 0)$
$\wedge\quad (r \not\simeq 0)$
$\wedge\quad (tr \not< ts)$

$$
\begin{array}{ll}
\wedge & (tr < (ts + w)) \\
\wedge & (w < (2 * r)) \\
\wedge & (r < (2 * w)) \\
\wedge & (n1 \in \mathbf{N}) \\
\wedge & (2 < \text{len}\,(rest))) \\
\rightarrow & (\text{target}\,(\text{warp}\,(\text{app}\,(\text{listn}\,(n1,\ flg1),\ \text{cons}\,(\texttt{'q},\ rest)),\ ts,\ tr,\ w,\ r)) \\
& \quad = \quad \text{app}\,(\text{listn}\,(\text{n*}\,(n1,\ ts,\ tr,\ w,\ r),\ flg1), \\
& \qquad\quad \text{app}\,(\text{listn}\,(\text{nq}\,(n1,\ ts,\ tr,\ w,\ r),\ \texttt{'q}), \\
& \qquad\qquad \text{warp}\,(\text{cdrn}\,(\text{dw}\,(n1,\ ts,\ tr,\ w,\ r),\ rest), \\
& \qquad\qquad\quad \text{ts}\,(n1,\ ts,\ tr,\ w,\ r), \\
& \qquad\qquad\quad \text{tr}\,(n1,\ ts,\ tr,\ w,\ r), \\
& \qquad\qquad\quad w, \\
& \qquad\qquad\quad r))))
\end{array}
$$

THEOREM: warp-app-listn-q

$$
\begin{array}{ll}
((ts \in \mathbf{N}) & \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (w \not\simeq 0) \\
\wedge & (r \not\simeq 0) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & (w < (2 * r)) \\
\wedge & (r < (2 * w)) \\
\wedge & (n1 \in \mathbf{N}) \\
\wedge & (2 < \text{len}\,(rest))) \\
\rightarrow & (\text{warp}\,(\text{app}\,(\text{listn}\,(n1,\ flg1),\ \text{cons}\,(\texttt{'q},\ rest)),\ ts,\ tr,\ w,\ r) \\
& \quad = \quad \text{app}\,(\text{listn}\,(\text{n*}\,(n1,\ ts,\ tr,\ w,\ r),\ flg1), \\
& \qquad\quad \text{app}\,(\text{listn}\,(\text{nq}\,(n1,\ ts,\ tr,\ w,\ r),\ \texttt{'q}), \\
& \qquad\qquad \text{warp}\,(\text{cdrn}\,(\text{dw}\,(n1,\ ts,\ tr,\ w,\ r),\ rest), \\
& \qquad\qquad\quad \text{ts}\,(n1,\ ts,\ tr,\ w,\ r), \\
& \qquad\qquad\quad \text{tr}\,(n1,\ ts,\ tr,\ w,\ r), \\
& \qquad\qquad\quad w, \\
& \qquad\qquad\quad r))))
\end{array}
$$

EVENT: Disable listn-add1.

EVENT: Disable warp-app-across-gap.

EVENT: Disable warp-app-gap.

EVENT: Disable warp-listn.

EVENT: Disable warp-app.

```
; Now there are two applications we must think about:  the proof of top and
; the proof of loop.  We will deal with top first.  To explore I want to
; do the smoothing and det part of top now and then come back to warp.

; Now we do the smoothing.
```

THEOREM: smooth-congruence
(¬ b-xor (*flg1*, *flg2*))
→ ((smooth (*flg2*, *rest*) = smooth (*flg1*, *rest*)) = **t**)

THEOREM: smooth-flg-app-listn-flg
(¬ b-xor (*flg1*, *flg2*))
→ (smooth (*flg1*, app (listn (*p1*, *flg2*), *rest*))
= app (listn (*p1*, *flg2*), smooth (*flg1*, *rest*)))

THEOREM: listp-app-listn
listp (app (listn (*n*, *flg*), *rest*)) = ((*n* ≯ 0) ∨ listp (*rest*))

THEOREM: listp-cells
listp (cells (*flg*, 5, 13, *msg*)) = listp (*msg*)

EVENT: Disable listp-app-listn.

THEOREM: car-app
car (app (*a*, *b*))
= **if** listp (*a*) **then** car (*a*)
else car (*b*) **endif**

THEOREM: listp-listn
listp (listn (*n*, *flg*)) = (*n* ≯ 0)

THEOREM: car-cells
listp (*msg*) → (car (cells (*flg*, 5, 13, *msg*)) = b-not (*flg*))

EVENT: Disable car-app.

THEOREM: top-smooth-step
listp (*msg*)
→ (smooth (**t**, app (listn (*p1*, **t**), app (cells (**t**, 5, 13, *msg*), listn (*p2*, **t**))))
= app (listn (*p1*, **t**),
cons ('q,
smooth (**f**, app (cdr (cells (**t**, 5, 13, *msg*)), listn (*p2*, **t**))))))

EVENT: Disable smooth-flg-app-listn-flg.

EVENT: Disable car-cells.

EVENT: Disable listp-cells.

```
; Now past the det
```

DEFINITION:
oracle* (*lst*, *oracle*)
= **if** *lst* ≃ **nil** **then** *oracle*
elseif car (*lst*) = 'q **then** oracle* (cdr (*lst*), cdr (*oracle*))
else oracle* (cdr (*lst*), *oracle*) **endif**

EVENT: Disable oracle*.

THEOREM: det-app
det (app (*lst1*, *lst2*), *oracle*)
= app (det (*lst1*, *oracle*), det (*lst2*, oracle* (*lst1*, *oracle*)))

THEOREM: det-listn
(*flg* ≠ 'q) → (det (listn (*n*, *flg*), *oracle*) = listn (*n*, *flg*))

THEOREM: oracle*-listn
(*flg* ≠ 'q) → (oracle* (listn (*n*, *flg*), *oracle*) = *oracle*)

```
; So now we combine all these.
```

THEOREM: len-smooth
len (smooth (*flg*, *lst*)) = len (*lst*)

THEOREM: cdr-app
cdr (app (*a*, *b*))
= **if** listp (*a*) **then** app (cdr (*a*), *b*)
else cdr (*b*) **endif**

THEOREM: top-async-send-lemma1
listp (*msg*) → (2 < len (cdr (cells (**t**, 5, 13, *msg*))))

EVENT: Disable cdr-app.

THEOREM: top-async-send

$$
\begin{array}{ll}
 & (\mathrm{bvp}\,(msg) \\
\wedge & \mathrm{listp}\,(msg) \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (w \not\simeq \mathtt{0}) \\
\wedge & (r \not\simeq \mathtt{0}) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & \mathrm{rate\text{-}proximity}\,(w,\ r) \\
\wedge & (p1 \in \mathbf{N})) \\
\rightarrow & (\mathrm{async}\,(\mathrm{send}\,(msg,\ p1,\ \mathtt{5},\ \mathtt{13},\ p2),\ ts,\ tr,\ w,\ r,\ oracle) \\
 & \quad = \quad \mathrm{app}\,(\mathrm{listn}\,(\mathrm{n}^*\,(p1,\ ts,\ tr,\ w,\ r),\ \mathbf{t}), \\
 & \qquad \mathrm{app}\,(\mathrm{det}\,(\mathrm{listn}\,(\mathrm{nqg}\,(\mathrm{nlst}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad \mathrm{nts}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad \mathrm{ntr}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad w, \\
 & \qquad\qquad r), \\
 & \qquad\quad \mathtt{'q}), \\
 & \qquad oracle), \\
 & \quad \mathrm{det}\,(\mathrm{warp}\,(\mathrm{cdrn}\,(\mathrm{dwg}\,(\mathrm{nlst}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad \mathrm{nts}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad \mathrm{ntr}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad w, \\
 & \qquad\qquad\qquad r), \\
 & \qquad\qquad \mathrm{smooth}\,(\mathbf{f}, \\
 & \qquad\qquad\qquad \mathrm{app}\,(\mathrm{cdr}\,(\mathrm{cells}\,(\mathbf{t},\ \mathtt{5},\ \mathtt{13},\ msg)), \\
 & \qquad\qquad\qquad\quad \mathrm{listn}\,(p2,\ \mathbf{t})))), \\
 & \qquad\quad \mathrm{tsg}\,(\mathrm{nlst}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad \mathrm{nts}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad \mathrm{ntr}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad w, \\
 & \qquad\qquad r), \\
 & \qquad\quad \mathrm{trg}\,(\mathrm{nlst}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad \mathrm{nts}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad \mathrm{ntr}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad w, \\
 & \qquad\qquad r), \\
 & \qquad\quad w, \\
 & \qquad\quad r), \\
 & \qquad \mathrm{oracle}^*\,(\mathrm{listn}\,(\mathrm{nqg}\,(\mathrm{nlst}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad \mathrm{nts}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad \mathrm{ntr}^*\,(p1,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad w,
\end{array}
$$

r),
’q),
oracle)))))

EVENT: Disable top-async-send-lemma1.

EVENT: Disable top-smooth-step.

```
; Now we get recv to eat the header.
```

THEOREM: scan-app-listn
scan (*flg*, app (listn (*n*, *flg*), *rest*)) = scan (*flg*, *rest*)

THEOREM: top-recv-step
recv (*n*, **t**, *k*, app (listn (*p1*, **t**), *rest*)) = recv (*n*, **t**, *k*, *rest*)

```
#| At this point our top level goal looks like this
(equal (recv (len msg)
             t 10
             (app (det (listn (nqg (nlst* p1 ts tr w r)
                                   (nts* p1 ts tr w r)
                                   (ntr* p1 ts tr w r)
                                   w r)
                               'q)
                       oracle)
                  (det (warp (cdrn (dwg (nlst* p1 ts tr w r)
                                        (nts* p1 ts tr w r)
                                        (ntr* p1 ts tr w r)
                                        w r)
                                   (smooth f
                                           (app (cdr (cells t 5 13 msg))
                                                (listn p2 t))))
                             (tsg (nlst* p1 ts tr w r)
                                  (nts* p1 ts tr w r)
                                  (ntr* p1 ts tr w r)
                                  w r)
                             (trg (nlst* p1 ts tr w r)
                                  (nts* p1 ts tr w r)
                                  (ntr* p1 ts tr w r)
                                  w r)
                             w r)
                       (oracle* (listn (nqg (nlst* p1 ts tr w r)
                                            (nts* p1 ts tr w r)
```

```
                                              (ntr* p1 ts tr w r)
                                              w r)
                                         'q)
                                   oracle))))
       msg)
and we will generalize it to
(equal (recv (len msg)
             t 10
             (app (det (listn nq 'q) oracle1)
                  (det (warp (cdrn dw
                                   (smooth f
                                           (app (cdr (cells t 5 13 msg))
                                                (listn p2 t))))
                                ts tr w r)
                       oracle2)))
       msg)
where we have the usual bounds on nq, dw, ts and tr.

To justify this generalization we must prove the theorems that the
expressions generalized satisfy the usual bounds.  We do that now.

|#
```

THEOREM: nqg-bounds
$(0 < \mathrm{nqg}(n, ts, tr, w, r)) \wedge (3 \not< \mathrm{nqg}(n, ts, tr, w, r))$

THEOREM: dwg-bounds
$1 \not< \mathrm{dwg}(n, ts, tr, w, r)$

```
; Typically trg and tsg will be instantiated as below.  We package up
; the required backchaining here.  It is my impression that the
; generalized form in which the * terms do not appear is not a
; theorem.  The proof of this relies upon
; NOT-LESSP-PLUS-R-NTR*-PLUS-NTS*-TIMES-W-NLST*.
```

THEOREM: not-lessp-trg*-tsg*
$((ts \in \mathbf{N})$
$\wedge \quad (tr \in \mathbf{N})$
$\wedge \quad (n \in \mathbf{N})$
$\wedge \quad (w \not\simeq 0)$
$\wedge \quad (r \not\simeq 0)$
$\wedge \quad (tr \not< ts)$
$\wedge \quad (tr < (ts + w))$

$$
\begin{array}{ll}
\wedge & (w < (2 * r)) \\
\wedge & (r < (2 * w))) \\
\rightarrow & (\mathrm{trg}\,(\mathrm{nlst}^*\,(n,\, ts,\, tr,\, w,\, r),\, \mathrm{nts}^*\,(n,\, ts,\, tr,\, w,\, r),\, \mathrm{ntr}^*\,(n,\, ts,\, tr,\, w,\, r),\, w,\, r) \\
 & \quad \not< \quad \mathrm{tsg}\,(\mathrm{nlst}^*\,(n,\, ts,\, tr,\, w,\, r), \\
 & \qquad\quad \mathrm{nts}^*\,(n,\, ts,\, tr,\, w,\, r), \\
 & \qquad\quad \mathrm{ntr}^*\,(n,\, ts,\, tr,\, w,\, r), \\
 & \qquad\quad w, \\
 & \qquad\quad r))
\end{array}
$$

THEOREM: lessp-trg*-plus-w-tsg*

$$
\begin{array}{ll}
((ts \in \mathbf{N}) & \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (n \in \mathbf{N}) \\
\wedge & (w \not\simeq 0) \\
\wedge & (r \not\simeq 0) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & (w < (2 * r)) \\
\wedge & (r < (2 * w))) \\
\rightarrow & (\mathrm{trg}\,(\mathrm{nlst}^*\,(n,\, ts,\, tr,\, w,\, r),\, \mathrm{nts}^*\,(n,\, ts,\, tr,\, w,\, r),\, \mathrm{ntr}^*\,(n,\, ts,\, tr,\, w,\, r),\, w,\, r) \\
 & \quad < \quad (w + \mathrm{tsg}\,(\mathrm{nlst}^*\,(n,\, ts,\, tr,\, w,\, r), \\
 & \qquad\quad \mathrm{nts}^*\,(n,\, ts,\, tr,\, w,\, r), \\
 & \qquad\quad \mathrm{ntr}^*\,(n,\, ts,\, tr,\, w,\, r), \\
 & \qquad\quad w, \\
 & \qquad\quad r)))
\end{array}
$$

```
#|
Consider the lhs of the concl of loop:

(recv (len msg)
      flg1
      10
      (app (det (listn nq 'q) oracle1)
           (det (warp (cdrn dw
                              (smooth flg2
                                      (app (cdr (cells flg1 5 13 msg))
                                           (listn p2 t))))
                      ts tr w r)
                oracle2)))

We will derive a killer rewrite rule that expands this, under the
conditions governing our induction, into (cons (car msg) ...) where
... is an instance of the lhs above.  I.e., the rule will step the
lhs of the ind concl into the induction hypothesis.  The derivation of
```

```
this rule will be documented by showing the successive steps.  The
actual discovery of the form of the lhs above and the steps was
made by proving the "killer rule" over and over again, each time
making some transformation.

The first step is to open (cells flg1 5 13 msg), drive the cdr around
the first cell and absorb it into the cell size, and associate the apps,
|#
```

THEOREM: cdr-app-cell-cells
$(n1 \not\simeq 0)$
$\rightarrow$ (cdr (app (cell (*flg1*, *n1*, *n2*, *bit*), cells (*flg2*, *n1*, *n2*, *msg*)))
$=$ app (cell (*flg1*, *n1* $-$ 1, *n2*, *bit*), cells (*flg2*, *n1*, *n2*, *msg*)))

```
#|
So lhs becomes:
(recv (len msg)
      flg1
      10
      (app (det (listn nq 'q) oracle1)
           (det (warp (cdrn dw
                              (smooth flg2
                                      (app (cell flg1 4 13 (car msg))
                                           (app (cells (csig flg1 (car msg)) 5 13 (cdr msg))
                                                (listn p2 t)))))
                      ts tr w r)
                oracle2)))

Our next goal is to drive the smooth through the app.
|#
```

THEOREM: smooth-flg-listn-flg
$(\neg$ b-xor (*flg1*, *flg2*)) $\rightarrow$ (smooth (*flg1*, listn (*n*, *flg2*)) $=$ listn (*n*, *flg2*))

THEOREM: not-b-xor-b-not
b-xor (*flg1*, *flg2*)
$\rightarrow$ (($\neg$ b-xor (*flg1*, b-not (*flg2*))) $\wedge$ ($\neg$ b-xor (b-not (*flg2*), *flg1*)))

THEOREM: smooth-flg-app-listn-not-flg
$((n \not\simeq 0) \wedge$ b-xor (*flg1*, *flg2*))
$\rightarrow$ (smooth (*flg1*, app (listn (*n*, *flg2*), *rest*))
$=$ app (smooth (*flg1*, listn (*n*, *flg2*)), smooth (*flg2*, *rest*)))

THEOREM: smooth-app-cell-app-cells

$((n1 \not\simeq 0) \wedge \text{b-xor}(flg1, flg2))$
$\rightarrow$ (smooth (*flg1*,
app (cell (*flg2*, *m1*, *n1*, *bit*),
app (cells (csig (*flg2*, *bit*), *m*, *n*, *msg*), listn (*p2*, **t**))))
= app (smooth (*flg1*, cell (*flg2*, *m1*, *n1*, *bit*)),
smooth (csig (*flg2*, *bit*),
app (cells (csig (*flg2*, *bit*), *m*, *n*, *msg*), listn (*p2*, **t**)))))

EVENT: Disable smooth-flg-listn-flg.

EVENT: Disable smooth-flg-app-listn-not-flg.

```
; If we had loop-stoppers on iff rules we could use an iff and be more efficient.
```

THEOREM: b-xor-commutes
b-xor $(x, y) \rightarrow$ b-xor (y, x)

```
#|
So lhs is now:
(recv (len msg)
      flg1
      10
      (app (det (listn nq 'q) oracle1)
           (det (warp (cdrn dw
                              (app (smooth flg2 (cell flg1 4 13 (car msg)))
                                   (smooth (csig flg1 (car msg))
                                           (app (cells (csig flg1 (car msg)) 5 13 (cdr msg))
                                                (listn p2 t)))))
                      ts tr w r)
                oracle2)))
Now drive the cdrn into the app and the smooth and absorb it in the
cell size.

|#
```

THEOREM: listp-smooth-cell
$(m \not\simeq 0) \rightarrow$ listp (smooth (*flg1*, cell (*flg2*, *m*, *n*, *bit*)))

THEOREM: cdrn-dw-app-smooth-cell
$((m \not\simeq 0) \wedge (dw \in \mathbf{N}) \wedge (1 \not< dw))$
$\rightarrow$ (cdrn (*dw*, app (smooth (*flg1*, cell (*flg2*, *m*, *n*, *bit*)), *rest*))
= app (cdrn (*dw*, smooth (*flg1*, cell (*flg2*, *m*, *n*, *bit*))), *rest*))

THEOREM: car-listn
car (listn (*n*, *flg*))
= **if** *n* ≃ 0 **then** 0
else *flg* **endif**

THEOREM: cdrn-dw-smooth-cell
((*m* ≉ 0) ∧ (*dw* ∈ **N**) ∧ (1 ≮ *dw*) ∧ b-xor (*flg1*, *flg2*))
→ (cdrn (*dw*, smooth (*flg2*, cell (*flg1*, *m*, *n*, car (*msg*))))
= smooth (*flg2*, cell (*flg1*, *m* − *dw*, *n*, car (*msg*))))

EVENT: Disable car-listn.

```
#|
So lhs is
(recv (len msg)
      flg1
      10
      (app (det (listn nq 'q) oracle1)
           (det (warp (app (smooth flg2 (cell flg1 (difference 4 dw) 13 (car msg)))
                           (smooth (csig flg1 (car msg))
                                   (app (cells (csig flg1 (car msg)) 5 13 (cdr msg))
                                        (listn p2 t))))
                      ts tr w r)
                oracle2)))
Next we drive the warp through the app.  We do that by putting a TARGET around
the warp and enabling warp-app.  To simplify the resulting len expressions we
use:
|#
```

THEOREM: len-cell
len (cell (*flg*, *m*, *n*, *bit*)) = (*m* + *n*)

THEOREM: add1-plus-12-difference-4-dw
(1 ≮ *dw*) → ((1 + (12 + (4 − *dw*))) = (17 − *dw*))

```
#|
Det-app also applies to drive the det through the app.
So now lhs is
(recv
 (len msg)
 flg1 10
 (app (det (listn nq 'q) oracle1)
      (app (det (warp (smooth flg2
```

```
                        (cell flg1
                              (difference 4 dw)
                              13
                              (car msg)))
                  ts tr w r)
            oracle2)
      (det (warp (app (lst* (smooth flg2
                                    (cell flg1
                                          (difference 4 dw)
                                          13
                                          (car msg)))
                            ts tr w r)
                       (smooth (csig flg1 (car msg))
                               (app (cells (csig flg1 (car msg)) 5
                                           13
                                           (cdr msg))
                                    (listn p2 t))))
                  (nts* (difference 17 dw) ts tr w r)
                  (ntr* (difference 17 dw) ts tr w r)
                  w r)
            (oracle* (app (lst* (smooth flg2
                                        (cell flg1
                                              (difference 4 dw)
                                              13
                                              (car msg)))
                                  ts tr w r)
                             (smooth (csig flg1 (car msg))
                                     (app (cells (csig flg1 (car msg)) 5
                                                 13
                                                 (cdr msg))
                                          (listn p2 t))))
                         oracle2)))))
```

Consider the second warp expression, the one applied to (app (lst* ...)...).
We want to apply warp-app-listn-q to this, which expects
(app (listn n1 flg1) (cons 'q rest))

So now we focus on causing that to happen. The lst* produces a listn.
Our induction will make a base case out of the one-bit msg. So we know
the current cell is followed by another and hence by an edge.
So our 'q comes from there.

We have this induction in mind but we don't yet know the ... parts. The
variables are listed in order of appearance.

```
(defn loop-ind-hint (nq oracle1 dw flg2 flg1 msg ts tr w r oracle2)
  (cond ((nlistp msg) t)
        ((nlistp (cdr msg)) t)
        (t (loop-ind-hint ... (cdr msg) ...))))

|#

; "The lst* produces a listn."
```

THEOREM: lst*-is-lastn-nlst*
$\text{lst*}\,(lst,\ ts,\ tr,\ w,\ r) = \text{lastn}\,(\text{nlst*}\,(\text{len}\,(lst),\ ts,\ tr,\ w,\ r),\ lst)$

THEOREM: lastn-app
$(\text{len}\,(b) \not< n) \rightarrow (\text{lastn}\,(n,\ \text{app}\,(a,\ b)) = \text{lastn}\,(n,\ b))$

THEOREM: not-lessp-2-nlst*
$((ts \in \mathbf{N})$
$\wedge\ \ (tr \in \mathbf{N})$
$\wedge\ \ (w \not\simeq 0)$
$\wedge\ \ (r \not\simeq 0)$
$\wedge\ \ (tr \not< ts)$
$\wedge\ \ (tr < (ts + w))$
$\wedge\ \ (w < (2 * r))$
$\wedge\ \ (r < (2 * w)))$
$\rightarrow\ \ (2 \not< \text{nlst*}\,(n,\ ts,\ tr,\ w,\ r))$

THEOREM: smooth-flg-listn-not-flg
$((n \not\simeq 0) \wedge \text{b-xor}\,(flg1,\ flg2))$
$\rightarrow\ \ (\text{smooth}\,(flg1,\ \text{listn}\,(n,\ flg2)) = \text{cons}\,(\texttt{'q},\ \text{listn}\,(n - 1,\ flg2)))$

EVENT: Disable smooth-flg-listn-not-flg.

THEOREM: lst*-smooth-cell
$((ts \in \mathbf{N})$
$\wedge\ \ (tr \in \mathbf{N})$
$\wedge\ \ (w \not\simeq 0)$
$\wedge\ \ (r \not\simeq 0)$
$\wedge\ \ (tr \not< ts)$
$\wedge\ \ (tr < (ts + w))$
$\wedge\ \ (w < (2 * r))$
$\wedge\ \ (r < (2 * w))$
$\wedge\ \ (m \not\simeq 0)$

$$
\begin{array}{ll}
\wedge & (n \in \mathbf{N}) \\
\wedge & (2 < n) \\
\wedge & \text{b-xor}\,(\mathit{flg1}, \mathit{flg2})) \\
\rightarrow & (\text{lst*}\,(\text{smooth}\,(\mathit{flg2}, \text{cell}\,(\mathit{flg1}, m, n, \mathit{bit})), ts, tr, w, r) \\
= & \text{listn}\,(\text{nlst*}\,(m + n, ts, tr, w, r), \text{csig}\,(\mathit{flg1}, \mathit{bit})))
\end{array}
$$

EVENT: Disable lst*-is-lastn-nlst*.

```
#|
So lhs is
(recv
 (len msg)
 flg1 10
 (app (det (listn nq 'q) oracle1)
      (app (det (warp (smooth flg2
                              (cell flg1
                                    (difference 4 dw)
                                    13
                                    (car msg)))
                      ts tr w r)
                oracle2)
           (det (warp (app (listn (nlst* (difference 17 dw) ts tr w r)
                                  (csig flg1 (car msg)))
                           (smooth (csig flg1 (car msg))
                                   (app (cells (csig flg1 (car msg)) 5
                                               13
                                               (cdr msg))
                                        (listn p2 t))))
                      (nts* (difference 17 dw) ts tr w r)
                      (ntr* (difference 17 dw) ts tr w r)
                      w r)
                (oracle* (warp (smooth flg2
                                       (cell flg1
                                             (difference 4 dw)
                                             13
                                             (car msg)))
                               ts tr w r)
                         oracle2)))))

And we're going to milk a 'q out of the smoothing of the cells term.
|#

; "So our 'q comes from there."
```

THEOREM: cdr-listn
cdr (listn (*n*, *flg*))
= **if** *n* ≃ 0 **then** 0
else listn (*n* − 1, *flg*) **endif**

EVENT: Disable cdr-listn.

THEOREM: smooth-app-cells
(listp (*msg*) ∧ (*m* ≄ 0))
→ (smooth (*flg*, app (cells (*flg*, *m*, *n*, *msg*), listn (*p2*, **t**)))
= cons ('q,
smooth (b-not (*flg*),
app (cdr (cells (*flg*, *m*, *n*, *msg*)), listn (*p2*, **t**)))))

```
#|
So lhs is
(recv
 (len msg)
 flg1 10
 (app (det (listn nq 'q) oracle1)
      (app (det (warp (smooth flg2
                              (cell flg1
                                    (difference 4 dw)
                                    13
                                    (car msg)))
                      ts tr w r)
                oracle2)
           (det (warp (app (listn (nlst* (difference 17 dw) ts tr w r)
                                  (csig flg1 (car msg)))
                           (cons 'q
                                 (smooth (b-not (csig flg1 (car msg)))
                                         (app (cdr (cells (csig flg1 (car msg)) 5
                                                          13
                                                          (cdr msg)))
                                              (listn p2 t)))))
                      (nts* (difference 17 dw) ts tr w r)
                      (ntr* (difference 17 dw) ts tr w r)
                      w r)
                (oracle* (warp (smooth flg2
                                       (cell flg1
                                             (difference 4 dw)
                                             13
                                             (car msg)))
                               ts tr w r)
```

```
                          oracle2)))))
Observe that warp-app-listn-q is now applicable if we can relieve the hypotheses.
The only nonobvious one is relieved by:
|#
```

THEOREM: lessp-2-len-cdr-cells
$\text{listp}\,(msg) \rightarrow (2 < \text{len}\,(\text{cdr}\,(\text{cells}\,(flg, 5, 13, msg))))$

```
; So warp-app-listn-q will push the warp through the two apps and let
; det-app push the det through.  Note that while warp-app-listn-q normally
; kicks out some t's or f's before the 'qs the initial string here is
; empty because we're starting with only nlst* of them.  Of course, we
; have to know
```

THEOREM: app-listn-0
$\text{app}\,(\text{listn}\,(0, flg), rest) = rest$

```
; We include the app because in the proof of our killer rule, app will be
; disabled.

; We are now ready to solidfy our gains as the first step of our killer
; rule. We use the name loop-killer-1a because it has a TARGET on our
; warp.  Loop-killer-1, next, doesn't.
```


```
; We include the app because in the proof of our killer rule, app will be
; disabled.

; We are now ready to solidfy our gains as the first step of our killer
; rule. We use the name loop-killer-1a because it has a TARGET on our
; warp.  Loop-killer-1, next, doesn't.
```


```
(prove-lemma loop-killer-1a nil
  (implies (and (bvp msg)
                (listp msg)
                (listp (cdr msg))
                (numberp ts)
                (numberp tr)
                (not (zerop w))
                (not (zerop r))
                (not (lessp tr ts))
```

```
     (lessp tr (plus ts w))
     (rate-proximity w r)

     (numberp nq)
     (not (lessp 3 nq))
     (numberp dw)
     (not (lessp 1 dw))
     (b-xor flg1 flg2))
(equal (recv (len msg)
             flg1
             10
             (app (det (listn nq 'q) oracle1)
                  (det (target (warp (cdrn dw
                                           (smooth flg2
                                                   (app (cdr (cells flg1 5 13 ms
                                                        (listn p2 t))))
                                      ts tr w r))
                               oracle2)))
       (recv
        (len msg)
        flg1 10
        (app (det (listn nq 'q) oracle1)
             (app (det (warp (smooth flg2
                                     (cell flg1
                                           (difference 4 dw)
                                           13
                                           (car msg)))
                               ts tr w r)
                          oracle2)
                  (app (det (listn (nqg (nlst* (difference 17 dw) ts tr w r)
                                        (nts* (difference 17 dw) ts tr w r)
                                        (ntr* (difference 17 dw) ts tr w r)
                                        w r)
                                   'q)
                            (oracle* (warp (smooth flg2
                                                   (cell flg1
                                                         (difference 4 dw)
                                                         13
                                                         (car msg)))
                                           ts tr w r)
                                     oracle2))
                       (det (warp (cdrn (dwg (nlst* (difference 17 dw) ts tr w r
                                             (nts* (difference 17 dw) ts tr w r)
                                             (ntr* (difference 17 dw) ts tr w r)
```

```
                    w r)
                (smooth (b-not (csig flg1 (car msg)))
                        (app (cdr (cells (csig flg1 (car
                                         13
                                         (cdr msg)))
                                 (listn p2 t))))
          (tsg (nlst* (difference 17 dw) ts tr w r)
               (nts* (difference 17 dw) ts tr w r)
               (ntr* (difference 17 dw) ts tr w r)
               w r)
          (trg (nlst* (difference 17 dw) ts tr w r)
               (nts* (difference 17 dw) ts tr w r)
               (ntr* (difference 17 dw) ts tr w r)
               w r)
          w r)
    (oracle* (listn (nqg (nlst* (difference 17 dw) ts tr
                         (nts* (difference 17 dw) ts tr
                         (ntr* (difference 17 dw) ts tr
                         w r)
                   'q)
       (oracle* (warp (smooth flg2
                      (cell flg1
                            (difference 4 dw)
                            13
                            (car msg)))
            ts tr w r)
        oracle2)))))))))
```

```
  ((disable len app cells)
   (enable warp-app)
   (expand (cells flg1 5 13 msg))))
```

EVENT: Disable cdr-app-cell-cells.

EVENT: Disable smooth-app-cells.

EVENT: Disable smooth-app-cell-app-cells.

```
; used to be loop-smooth-step
```

EVENT: Disable cdrn-dw-smooth-cell.

EVENT: Disable cdrn-dw-app-smooth-cell.

EVENT: Disable lst*-smooth-cell.

```
(prove-lemma loop-killer-1 (rewrite)
  (implies (and (bvp msg)
                (listp msg)
                (listp (cdr msg))
                (numberp ts)
                (numberp tr)
                (not (zerop w))
                (not (zerop r))
                (not (lessp tr ts))
                (lessp tr (plus ts w))
                (rate-proximity w r)

                (numberp nq)
                (not (lessp 3 nq))
                (numberp dw)
                (not (lessp 1 dw))
                (b-xor flg1 flg2))
           (equal (recv (len msg)
                        flg1
                        10
                        (app (det (listn nq 'q) oracle1)
                             (det (warp (cdrn dw
                                              (smooth flg2
                                                      (app (cdr (cells flg1 5 13 msg))
                                                           (listn p2 t))))
                                        ts tr w r)
                                  oracle2)))
                  (recv
                   (len msg)
                   flg1 10
                   (app (det (listn nq 'q) oracle1)
                        (app (det (warp (smooth flg2
                                                (cell flg1
                                                      (difference 4 dw)
                                                      13
                                                      (car msg)))
                                        ts tr w r)
                                  oracle2)
                             (app (det (listn (nqg (nlst* (difference 17 dw) ts tr w r)
```

```
                    (nts* (difference 17 dw) ts tr w r)
                    (ntr* (difference 17 dw) ts tr w r)
                    w r)
             'q)
       (oracle* (warp (smooth flg2
                              (cell flg1
                                    (difference 4 dw)
                                    13
                                    (car msg)))
                      ts tr w r)
                oracle2))
 (det (warp (cdrn (dwg (nlst* (difference 17 dw) ts tr w r)
                       (nts* (difference 17 dw) ts tr w r)
                       (ntr* (difference 17 dw) ts tr w r)
                       w r)
                  (smooth (b-not (csig flg1 (car msg)))
                          (app (cdr (cells (csig flg1 (car
                                                      13
                                                      (cdr msg)))
                                           (listn p2 t))))
                  (tsg (nlst* (difference 17 dw) ts tr w r)
                       (nts* (difference 17 dw) ts tr w r)
                       (ntr* (difference 17 dw) ts tr w r)
                       w r)
                  (trg (nlst* (difference 17 dw) ts tr w r)
                       (nts* (difference 17 dw) ts tr w r)
                       (ntr* (difference 17 dw) ts tr w r)
                       w r)
                  w r)
       (oracle* (listn (nqg (nlst* (difference 17 dw) ts tr
                                   (nts* (difference 17 dw) ts tr
                                   (ntr* (difference 17 dw) ts tr
                                   w r)
                        'q)
                   (oracle* (warp (smooth flg2
                                    (cell flg1
                                          (difference 4 dw)
                                          13
                                          (car msg)))
                            ts tr w r)
                       oracle2)))))))))
((enable target)
 (use (loop-killer-1a))))
```

```
#|
Observe in the rhs of loop-killer-1 the emergence of an instance of the
critical part of the lhs:

(app (det (listn nq 'q)
          oracle1)
     (det (warp (cdrn dw
                      (smooth flg2
                              (app (cdr (cells flg1 5 13 msg))
                                   (listn p2 t))))
                ts tr w r)
          oracle2))
From this instance we can read off our induction hypothesis:
|#
```

DEFINITION:
loop-ind-hint (nq, *oracle1*, dw, *flg2*, *flg1*, msg, ts, tr, w, r, *oracle2*)
= **if** $msg \simeq$ **nil then t**
elseif cdr (msg) $\simeq$ **nil then t**
else loop-ind-hint (nqg (nlst* (17 − dw, ts, tr, w, r),
nts* (17 − dw, ts, tr, w, r),
ntr* (17 − dw, ts, tr, w, r),
w,
r),
oracle* (warp (smooth (*flg2*,
cell (*flg1*,
4 − dw,
13,
car (msg))),
ts,
tr,
w,
r),
oracle2),
dwg (nlst* (17 − dw, ts, tr, w, r),
nts* (17 − dw, ts, tr, w, r),
ntr* (17 − dw, ts, tr, w, r),
w,
r),
b-not (csig (*flg1*, car (msg))),
csig (*flg1*, car (msg)),
cdr (msg),
tsg (nlst* (17 − dw, ts, tr, w, r),

nts* (17 − *dw*, *ts*, *tr*, *w*, *r*),
ntr* (17 − *dw*, *ts*, *tr*, *w*, *r*),
w,
r),
trg (nlst* (17 − *dw*, *ts*, *tr*, *w*, *r*),
nts* (17 − *dw*, *ts*, *tr*, *w*, *r*),
ntr* (17 − *dw*, *ts*, *tr*, *w*, *r*),
w,
r),
w,
r,
oracle* (listn (nqg (nlst* (17 − *dw*, *ts*, *tr*, *w*, *r*),
nts* (17 − *dw*, *ts*, *tr*, *w*, *r*),
ntr* (17 − *dw*, *ts*, *tr*, *w*, *r*),
w,
r),
'q),
oracle* (warp (smooth (*flg2*,
cell (*flg1*,
4 − *dw*,
13,
car (*msg*))),
ts,
tr,
w,
r),
oracle2))) **endif**

```
#|
In order for this induction to work out, we have to be able to establish that
the instantiations satisfy the hypotheses of the theorem being proved.  We
have made sure of this through the previously proved lemmas NQG-BOUNDS,
DWG-BOUNDS, NOT-LESSP-TRG*-TSG* and LESSP-TRG*-PLUS-W-TSG*.

Loop-killer-1 does not get us back to the induction hypothesis because
we have to scan past the first cell.  So we now continue with our loop
killer development.  We aim for the following rewrite (under appropriate
hyps):

(equal (recv (len msg) flg1 10
             (app (det (listn nq 'q) oracle1)
                  (app (det (warp (smooth flg2 (cell flg1 (difference 4 dw) 13 bit))
                                  ts tr w r)
                            oracle)
```

```
                    rest)))
       (cons bit
             (recv (len (cdr msg))
                   (csig flg1 bit)
                   10
                   rest)))
```

This will be the conclusion of loop-killer-2. Observe that killer-2 takes up where killer-1 left off and, in conjunction with the induction hyp, will reduce the lhs of the induction concl to (cons (car msg) (cdr msg)).

We resume our step-by-step development by considering the lhs recv above and successively transforming it.

By opening up the lhs recv above we end up with two hard requirements. The first is that recv reads the correct bit. We'll call this loop-killer-2a:

```
(recv-bit 10
          (scan flg1
                (app (det (listn nq 'q) oracle1)
                     (app (det (warp (smooth flg2
                                             (cell flg1
                                                   (difference 4 dw)
                                                   13
                                                   (car msg)))
                                       ts tr w r)
                                  oracle)
                             rest))))
= (car msg)
```

The second is that it resumes it scan on a string of bits of the correct parity.

Loop-killer-2b:

```
(cdrn 10
      (scan flg1
            (app (det (listn nq 'q) oracle1)
                 (app (det (warp (smooth flg2
                                         (cell flg1 (difference 4 dw) 13 (car msg))
                                   ts tr w r)
                              oracle)
                         rest)))))
= (app (listn ??? (csig flg1 x)) rest)
```

where ??? is some non-zero number we'll determine during the proof below.

```
It is loop-killer-2b that gets us back to our induction hyp and
so we will work on it first.  As usual the game is to move the app from
its hiding place in cell through smooth, warp, and det and then do some
arithmetic.

Because it is crucial to all that we do henceforth, we are going to
first show what (warp (smooth flg2 (cell flg1 m n bit)) ts tr w r)
looks like.


|#
```

THEOREM: app-listn-flg-listn-flg
$\text{app}(\text{listn}(m, \mathit{flg}), \text{listn}(n, \mathit{flg})) = \text{listn}(m + n, \mathit{flg})$

THEOREM: cdrn-listn
$(n \not< dw) \rightarrow (\text{cdrn}(dw, \text{listn}(n, \mathit{flg})) = \text{listn}(n - dw, \mathit{flg}))$

```
; The requirement on n below stems from (a) smooth-flg-listn-flg, as it
; smooths the second subcell, needs to know that n is at least 1 so that it
; can take a 'q out; (b) warp-app-listn-q as it is warping that subcell
; needs to know that n-1 is greater than 2 so it can eat at most two
; signals after the q.
```

THEOREM: warp-smooth-cell
$$
\begin{array}{ll}
(\text{boolp}(\mathit{bit}) & \\
\quad \wedge & (\mathit{ts} \in \mathbf{N}) \\
\quad \wedge & (\mathit{tr} \in \mathbf{N}) \\
\quad \wedge & (w \not\simeq 0) \\
\quad \wedge & (r \not\simeq 0) \\
\quad \wedge & (\mathit{tr} \not< \mathit{ts}) \\
\quad \wedge & (\mathit{tr} < (\mathit{ts} + w)) \\
\quad \wedge & \text{rate-proximity}(w, r) \\
\quad \wedge & (m \in \mathbf{N}) \\
\quad \wedge & (n \in \mathbf{N}) \\
\quad \wedge & (3 < n) \\
\quad \wedge & \text{b-xor}(\mathit{flg1}, \mathit{flg2})) \\
\rightarrow & (\text{warp}(\text{smooth}(\mathit{flg2}, \text{cell}(\mathit{flg1}, m, n, \mathit{bit})), \mathit{ts}, \mathit{tr}, w, r) \\
\quad = & \textbf{if } \mathit{bit} \\
& \textbf{then } \text{app}(\text{listn}(\text{n*}(m, \mathit{ts}, \mathit{tr}, w, r), \text{b-not}(\mathit{flg1})), \\
& \qquad \text{app}(\text{listn}(\text{nqg}(\text{nlst*}(m, \mathit{ts}, \mathit{tr}, w, r), \\
& \qquad\qquad \text{nts*}(m, \mathit{ts}, \mathit{tr}, w, r), \\
& \qquad\qquad \text{ntr*}(m, \mathit{ts}, \mathit{tr}, w, r),
\end{array}
$$

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ w,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \texttt{'q}),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \text{listn}\,(\text{n*}\,((n-1)\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad -\quad \text{dwg}\,(\text{nlst*}\,(m,\ ts,\ tr,\ w,\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \text{nts*}\,(m,\ ts,\ tr,\ w,\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \text{ntr*}\,(m,\ ts,\ tr,\ w,\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ w,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{tsg}\,(\text{nlst*}\,(m,\ ts,\ tr,\ w,\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \text{nts*}\,(m,\ ts,\ tr,\ w,\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \text{ntr*}\,(m,\ ts,\ tr,\ w,\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ w,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{trg}\,(\text{nlst*}\,(m,\ ts,\ tr,\ w,\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \text{nts*}\,(m,\ ts,\ tr,\ w,\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \text{ntr*}\,(m,\ ts,\ tr,\ w,\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ w,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad w,\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad r),\\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \textit{flg1})))\\
\textbf{else}\ \text{listn}\,(\text{n*}\,(m+n,\ ts,\ tr,\ w,\ r),\ \text{b-not}\,(\textit{flg1}))\ \textbf{endif})
\end{array}
$$

```
; To cdrn through the rhs above we need bounds on n*.  To prove the
; bounds we will have to characterize n* algebraically.  The bounds
; theorems are too weak to prove inductively.
```

THEOREM: nendp-alg
$((ts \in \mathbf{N}) \wedge (tr+ \in \mathbf{N}) \wedge (w \not\simeq 0) \wedge (ts < tr+))$
$\rightarrow \quad (\text{nendp}\,(n,\ ts,\ tr+,\ w) = ((ts + (n * w)) < tr+))$

THEOREM: nlst+-ts-plus-ts-w
$((n \in \mathbf{N}) \wedge (w \not\simeq 0)) \rightarrow (\text{nlst+}\,(n,\ ts,\ ts + w,\ w) = (n - 1))$

THEOREM: nlst+-alg
$((n \in \mathbf{N}) \wedge (ts \in \mathbf{N}) \wedge (tr+ \in \mathbf{N}) \wedge (w \not\simeq 0) \wedge (ts < tr+))$
$\rightarrow \quad (\text{nlst+}\,(n,\ ts,\ tr+,\ w) = (n - ((tr+ - ts) \div w)))$

EVENT: Disable nlst+-ts-plus-ts-w.

THEOREM: not-lessp-nts+-ts
$\text{nts+}\,(n,\ ts,\ tr+,\ w) \not< ts$

THEOREM: nts+-alg
$((n \in \mathbf{N}) \wedge (ts \in \mathbf{N}) \wedge (tr+ \in \mathbf{N}) \wedge (w \not\simeq 0) \wedge (tr+ \not< ts))$
$\rightarrow \quad (\text{nts+}\,(n,\ ts,\ tr+,\ w)$
$\quad = \quad \textbf{if}\ (ts + (n * w)) < tr+\ \textbf{then}\ ts + (n * w)$
$\quad\quad \textbf{else}\ ts + (w * ((tr+ - ts) \div w))\ \textbf{endif})$

THEOREM: n*-alg-lemma
$((x \in \mathbf{N})$
$\wedge \quad (r \neq 0)$
$\wedge \quad (r \in \mathbf{N})$
$\wedge \quad (\neg\ \text{nendp}\,(n,\ ts,\ r + ts + x,\ w))$
$\wedge \quad (n \in \mathbf{N})$
$\wedge \quad (ts \in \mathbf{N})$
$\wedge \quad (w \in \mathbf{N})$
$\wedge \quad (w \neq 0)$
$\wedge \quad ((ts + x) \not< ts)$
$\wedge \quad ((ts + x) < (ts + w)))$
$\rightarrow \quad ((1 + (((w * \text{nlst+}\,(n,\ ts,\ r + ts + x,\ w))$
$\quad\quad - \quad ((r + ts + x) - \text{nts+}\,(n,\ ts,\ r + ts + x,\ w)))$
$\quad\quad \div \quad r))$
$\quad = \quad (((n * w) - x) \div r))$

EVENT: Disable nsig*-alg-lemma-hack1.

THEOREM: n*-alg-hack1
$(\text{nendp}\,(n,\ ts,\ r + tr,\ w)$
$\wedge \quad (n \in \mathbf{N})$
$\wedge \quad (ts \in \mathbf{N})$
$\wedge \quad (tr \in \mathbf{N})$
$\wedge \quad (w \in \mathbf{N})$
$\wedge \quad (w \neq 0)$
$\wedge \quad (r \in \mathbf{N})$
$\wedge \quad (r \neq 0)$
$\wedge \quad (tr \not< ts)$
$\wedge \quad (tr < (ts + w))$
$\wedge \quad ((n * w) \not< (tr - ts)))$
$\rightarrow \quad ((((n * w) - (tr - ts)) < r) = \mathbf{t})$

THEOREM: n*-alg
$((n \in \mathbf{N})$
$\wedge \quad (ts \in \mathbf{N})$
$\wedge \quad (tr \in \mathbf{N})$
$\wedge \quad (w \not\simeq 0)$
$\wedge \quad (r \not\simeq 0)$

$\wedge \quad (tr \not< ts)$
$\wedge \quad (tr < (ts + w)))$
$\rightarrow \quad (\mathrm{n}^*(n,\ ts,\ tr,\ w,\ r) = (((n * w) - (tr - ts)) \div r))$

EVENT: Disable n*-alg-hack1.

EVENT: Disable n*-alg.

EVENT: Disable n*-alg-lemma.

EVENT: Disable nts+-alg.

EVENT: Disable nlst+-alg.

EVENT: Disable nendp-alg.

EVENT: Disable nsig*-upper-bound-hack1.

EVENT: Disable nsig*-upper-bound-hack2.

EVENT: Disable nsig*-upper-bound-hack3.

EVENT: Disable nsig*-upper-bound-lemma2-equality.

THEOREM: n*-upper-bound
$((n \in \mathbf{N})$
$\wedge \quad (ts \in \mathbf{N})$
$\wedge \quad (tr \in \mathbf{N})$
$\wedge \quad (w \not\simeq \mathtt{0})$
$\wedge \quad (r \not\simeq \mathtt{0})$
$\wedge \quad (tr \not< ts)$
$\wedge \quad (tr < (ts + w))$
$\wedge \quad \text{rate-proximity}\,(w,\ r)$
$\wedge \quad (n < \mathtt{18}))$
$\rightarrow \quad (n \not< \mathrm{n}^*(n,\ ts,\ tr,\ w,\ r))$

EVENT: Disable nsig*-upper-bound-lemma1.

EVENT: Disable nsig*-upper-bound-lemma2.

THEOREM: n*-lower-bound
$((n \in \mathbf{N})$
$\wedge \quad (ts \in \mathbf{N})$
$\wedge \quad (tr \in \mathbf{N})$
$\wedge \quad (w \not\simeq \mathtt{0})$
$\wedge \quad (r \not\simeq \mathtt{0})$
$\wedge \quad (tr \not< ts)$
$\wedge \quad (tr < (ts + w))$
$\wedge \quad \text{rate-proximity}\,(w,\, r)$
$\wedge \quad (n < \mathtt{18}))$
$\rightarrow \quad (\text{n*}\,(n,\, ts,\, tr,\, w,\, r) \not< ((n - 1) - 1))$

EVENT: Disable nsig*-lower-bound-lemma1.

```
; So now we know n-2 <= (n* n ts tr w r) <= n.

; Now we consider scanning and cdrning.
```

THEOREM: len-det
$\text{len}\,(\text{det}\,(lst,\, oracle)) = \text{len}\,(lst)$

DEFINITION:
det-listn-hint $(nq,\, oracle)$
$=$ **if** $nq \simeq \mathtt{0}$ **then t**
else det-listn-hint $(nq - 1,\, \text{cdr}\,(oracle))$ **endif**

DEFINITION:
no $(flg,\, nq,\, oracle)$
$=$ **if** $nq \simeq \mathtt{0}$ **then** 0
elseif b-xor $(flg,\, \text{car}\,(oracle))$ **then** nq
else no $(flg,\, nq - 1,\, \text{cdr}\,(oracle))$ **endif**

```
; The following lemma is provided by way of explanation.
```

THEOREM: no-is-len-scan-det-listn
$(nq \in \mathbf{N})$
$\rightarrow \quad (\text{no}\,(flg,\, nq,\, oracle) = \text{len}\,(\text{scan}\,(flg,\, \text{det}\,(\text{listn}\,(nq,\, \mathtt{'q}),\, oracle))))$

THEOREM: scan-flg-app-listn-not-flg
$((\mathtt{0} < n) \wedge \text{b-xor}\,(flg1,\, flg2))$
$\rightarrow \quad (\text{scan}\,(flg1,\, \text{app}\,(\text{listn}\,(n,\, flg2),\, rest)) = \text{app}\,(\text{listn}\,(n,\, flg2),\, rest))$

DEFINITION:
scan-oracle $(flg,\, nq,\, oracle)$

= **if** $nq \simeq$ `0` **then** *oracle*
elseif b-xor (*flg*, car (*oracle*)) **then** *oracle*
else scan-oracle (*flg*, $nq - 1$, cdr (*oracle*)) **endif**

```
; The following lemma looks horrible -- the det expression it introduces
; seems a bad trade.  But its oracle is irrelevant elsewhere and its len,
; which is in fact is all we care about, is simple.
```

THEOREM: scan-app-det-listn
$((\texttt{0} < n) \wedge$ b-xor (*flg1*, *flg2*))
$\rightarrow$ (scan (*flg1*, app (det (listn (*nq*, `'q`), *oracle*), app (listn (*n*, *flg2*), *rest*)))
= app (det (listn (no (*flg1*, *nq*, *oracle*), `'q`),
scan-oracle (*flg1*, *nq*, *oracle*)),
app (listn (*n*, *flg2*), *rest*)))

EVENT: Disable scan-flg-app-listn-not-flg.

THEOREM: cdrn-app
cdrn (n, app (a, b))
= **if** $n <$ len (a) **then** app (cdrn (n, a), b)
else cdrn ($n -$ len (a), b) **endif**

THEOREM: not-lessp-no
$n \not< $ no (*flg*, n, *oracle*)

THEOREM: boolp-implies-det-listn
boolp (*flg*) $\rightarrow$ (det (listn (n, *flg*), *oracle*) = listn (n, *flg*))

```
; We are about to prove the key lemma that determines baud rate.
```

THEOREM: len-warp
len (warp (*lst*, *ts*, *tr*, w, r)) = n* (len (*lst*), *ts*, *tr*, w, r)

```
; To prove n*-plus we have to USE len-warp with an instantiation
; that causes warp-app-listn-q to fire.  Len-warp is of lemma class
; nil so that the equality thus derived stays around.  But it turns out
; that elsewhere in the proof we have to use len-warp as a rewrite rule!
; So we prove an instance of it that suits our purposes.
```

THEOREM: n*-plus-lemma
len (warp (listn ($j -$ *dwg*, **f**), *ts*, *tr*, w, r)) = n* ($j -$ *dwg*, *ts*, *tr*, w, r)

```
; The name n*-plus, below, suggests that the lhs is (n* (plus ...)
; ...) but in fact I named it after the rhs; couldn't bear to rewrite
; the other way.
```

THEOREM: n*-plus

$$
\begin{array}{ll}
 & ((ts \in \mathbf{N}) \\
 & \wedge \quad (tr \in \mathbf{N}) \\
 & \wedge \quad (w \not\simeq 0) \\
 & \wedge \quad (r \not\simeq 0) \\
 & \wedge \quad (tr \not< ts) \\
 & \wedge \quad (tr < (ts + w)) \\
 & \wedge \quad (w < (2 * r)) \\
 & \wedge \quad (r < (2 * w)) \\
 & \wedge \quad (2 < j) \\
 & \wedge \quad (i \in \mathbf{N}) \\
 & \wedge \quad (j \in \mathbf{N})) \\
\rightarrow & \quad ((\mathrm{n}^*(i,\ ts,\ tr,\ w,\ r) \\
 & \qquad + \quad \mathrm{nqg}(\mathrm{nlst}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad \mathrm{nts}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad \mathrm{ntr}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad w, \\
 & \qquad\qquad r) \\
 & \qquad + \quad \mathrm{n}^*(j - \mathrm{dwg}(\mathrm{nlst}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad \mathrm{nts}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad \mathrm{ntr}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad w, \\
 & \qquad\qquad\qquad r), \\
 & \qquad\qquad \mathrm{tsg}(\mathrm{nlst}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad \mathrm{nts}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad \mathrm{ntr}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad w, \\
 & \qquad\qquad\qquad r), \\
 & \qquad\qquad \mathrm{trg}(\mathrm{nlst}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad \mathrm{nts}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad \mathrm{ntr}^*(i,\ ts,\ tr,\ w,\ r), \\
 & \qquad\qquad\qquad w, \\
 & \qquad\qquad\qquad r), \\
 & \qquad\qquad w, \\
 & \qquad\qquad r)) \\
 & \qquad = \quad \mathrm{n}^*(1 + (i + j),\ ts,\ tr,\ w,\ r))
\end{array}
$$

EVENT: Disable n*-plus-lemma.

```
; If this lemma can be proved for different constants then the
; whole proof can be shifted to those constants.  In particular
; make the replacements indicated below.
; To send cells of size: 32   24  16
;                        31   23  15
;                        16   12   8
;                        15   11   7
```

THEOREM: loop-killer-2b

$$
\begin{array}{ll}
 & (\mathrm{listp}\,(\mathit{msg}) \\
\wedge & \mathrm{boolp}\,(\mathrm{car}\,(\mathit{msg})) \\
\wedge & \mathrm{bvp}\,(\mathrm{cdr}\,(\mathit{msg})) \\
\wedge & (\mathit{ts} \in \mathbf{N}) \\
\wedge & (\mathit{tr} \in \mathbf{N}) \\
\wedge & (w \not\simeq \mathtt{0}) \\
\wedge & (r \not\simeq \mathtt{0}) \\
\wedge & (\mathit{tr} \not< \mathit{ts}) \\
\wedge & (\mathit{tr} < (\mathit{ts} + w)) \\
\wedge & \text{rate-proximity}\,(w,\, r) \\
\wedge & (\mathit{nq} \in \mathbf{N}) \\
\wedge & (\mathtt{3} \not< \mathit{nq}) \\
\wedge & (\mathit{dw} \in \mathbf{N}) \\
\wedge & (\mathtt{1} \not< \mathit{dw}) \\
\wedge & \mathrm{boolp}\,(\mathit{flg1}) \\
\wedge & \text{b-xor}\,(\mathit{flg1},\, \mathit{flg2})) \\
\rightarrow & (\mathrm{cdrn}\,(\mathtt{10}, \\
 & \quad \mathrm{scan}\,(\mathit{flg1}, \\
 & \quad\quad \mathrm{app}\,(\mathrm{det}\,(\mathrm{listn}\,(\mathit{nq},\, \mathtt{'q}),\, \mathit{oracle1}), \\
 & \quad\quad\quad \mathrm{app}\,(\mathrm{det}\,(\mathrm{warp}\,(\mathrm{smooth}\,(\mathit{flg2}, \\
 & \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \mathrm{cell}\,(\mathit{flg1},\, \mathtt{4} - \mathit{dw},\, \mathtt{13},\, \mathrm{car}\,(\mathit{msg}))), \\
 & \quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{ts}, \\
 & \quad\quad\quad\quad\quad\quad\quad\quad\quad \mathit{tr}, \\
 & \quad\quad\quad\quad\quad\quad\quad\quad\quad w, \\
 & \quad\quad\quad\quad\quad\quad\quad\quad\quad r), \\
 & \quad\quad\quad\quad\quad\quad\quad\quad \mathit{oracle2}), \\
 & \quad\quad\quad\quad\quad\quad\quad \mathit{rest})))) \\
 & \quad = \quad \mathrm{app}\,(\mathrm{listn}\,((\mathrm{no}\,(\mathit{flg1},\, \mathit{nq},\, \mathit{oracle1}) + \mathrm{n{*}}\,(\mathtt{17} - \mathit{dw},\, \mathit{ts},\, \mathit{tr},\, w,\, r)) \\
 & \quad\quad\quad\quad\quad\quad - \quad \mathtt{10}, \\
 & \quad\quad\quad\quad\quad\quad \mathrm{csig}\,(\mathit{flg1},\, \mathrm{car}\,(\mathit{msg}))), \\
 & \quad\quad\quad\quad \mathit{rest}))
\end{array}
$$

EVENT: Disable n*-plus.

EVENT: Disable cdrn-app.

EVENT: Disable scan-app-det-listn.

EVENT: Disable boolp-implies-det-listn.

EVENT: Disable warp-smooth-cell.

THEOREM: car-det-listn
car (det (listn (*n*, 'q), *oracle*))
= **if** *n* ≃ 0 **then** 0
elseif car (*oracle*) **then t**
else f endif

THEOREM: listp-det
listp (det (*lst*, *oracle*)) = listp (*lst*)

THEOREM: car-scan-oracle
(no (*flg*, *nq*, *oracle*) ≉ 0)
→ (car (scan-oracle (*flg*, *nq*, *oracle*)) ↔ b-not (*flg*))

THEOREM: loop-killer-2a-lemma
(listp (*msg*)
∧ boolp (car (*msg*))
∧ bvp (cdr (*msg*))
∧ (*ts* ∈ **N**)
∧ (*tr* ∈ **N**)
∧ (*w* ≉ 0)
∧ (*r* ≉ 0)
∧ (*tr* ≮ *ts*)
∧ (*tr* < (*ts* + *w*))
∧ rate-proximity (*w*, *r*)
∧ (*nq* ∈ **N**)
∧ (3 ≮ *nq*)
∧ (*dw* ∈ **N**)
∧ (1 ≮ *dw*)
∧ boolp (*flg1*)
∧ boolp (*flg2*)
∧ b-xor (*flg1*, *flg2*))
→ (car (scan (*flg1*,
app (det (listn (*nq*, 'q), *oracle1*),
app (det (warp (smooth (*flg2*,

cell (*flg1*, 4 − *dw*, 13, car (*msg*))),
ts,
tr,
w,
r),
oracle),
rest))))
= b-not (*flg1*))

EVENT: Disable car-det-listn.

THEOREM: equal-difference-0
$((x - y) = 0) = (y \not< x)$

THEOREM: loop-killer-2a
(listp (*msg*)
∧ boolp (car (*msg*))
∧ bvp (cdr (*msg*))
∧ $(ts \in \mathbf{N})$
∧ $(tr \in \mathbf{N})$
∧ $(w \not\simeq 0)$
∧ $(r \not\simeq 0)$
∧ $(tr \not< ts)$
∧ $(tr < (ts + w))$
∧ rate-proximity (w, r)
∧ $(nq \in \mathbf{N})$
∧ $(3 \not< nq)$
∧ $(dw \in \mathbf{N})$
∧ $(1 \not< dw)$
∧ boolp (*flg1*)
∧ boolp (*flg2*)
∧ b-xor (*flg1*, *flg2*))
→ (recv-bit (10,
scan (*flg1*,
app (det (listn (*nq*, 'q), *oracle1*),
app (det (warp (smooth (*flg2*,
cell (*flg1*,
4 − *dw*,
13,
car (*msg*))),
ts,
tr,
w,
r),

$$\begin{array}{l} \qquad\qquad\qquad\qquad\qquad \mathit{oracle}), \\ \qquad\qquad\qquad\qquad \mathit{rest})))) \\ \quad = \ \mathrm{car}\,(\mathit{msg})) \end{array}$$

EVENT: Disable loop-killer-2a-lemma.

```
; We also have to take care of the new flg computed by recv:
```

THEOREM: loop-killer-2c

$$\begin{array}{l} (\mathrm{listp}\,(\mathit{msg}) \\ \ \wedge \ \ \mathrm{boolp}\,(\mathrm{car}\,(\mathit{msg})) \\ \ \wedge \ \ \mathrm{bvp}\,(\mathrm{cdr}\,(\mathit{msg})) \\ \ \wedge \ \ (\mathit{ts} \in \mathbf{N}) \\ \ \wedge \ \ (\mathit{tr} \in \mathbf{N}) \\ \ \wedge \ \ (w \not\simeq 0) \\ \ \wedge \ \ (r \not\simeq 0) \\ \ \wedge \ \ (\mathit{tr} \not< \mathit{ts}) \\ \ \wedge \ \ (\mathit{tr} < (\mathit{ts} + w)) \\ \ \wedge \ \ \text{rate-proximity}\,(w,\, r) \\ \ \wedge \ \ (\mathit{nq} \in \mathbf{N}) \\ \ \wedge \ \ (3 \not< \mathit{nq}) \\ \ \wedge \ \ (\mathit{dw} \in \mathbf{N}) \\ \ \wedge \ \ (1 \not< \mathit{dw})) \\ \rightarrow \ \ (\mathrm{car}\,(\mathrm{app}\,(\mathrm{listn}\,((\mathrm{no}\,(\mathit{flg1},\, \mathit{nq},\, \mathit{oracle1}) + \mathrm{n}^*\,(17 - \mathit{dw},\, \mathit{ts},\, \mathit{tr},\, w,\, r)) \\ \qquad\qquad\qquad\qquad - \ \ 10, \\ \qquad\qquad\qquad\qquad \mathrm{csig}\,(\mathit{flg1},\, \mathrm{car}\,(\mathit{msg}))), \\ \qquad\qquad\qquad \mathit{rest})) \\ \quad = \ \mathrm{csig}\,(\mathit{flg1},\, \mathrm{car}\,(\mathit{msg}))) \end{array}$$

```
; Finally, we must show that recv just eats up the leading string of flgs:
```

THEOREM: recv-app-listn

$$\mathrm{recv}\,(n,\, \mathit{flg},\, k,\, \mathrm{app}\,(\mathrm{listn}\,(m,\, \mathit{flg}),\, \mathit{rest})) = \mathrm{recv}\,(n,\, \mathit{flg},\, k,\, \mathit{rest})$$

THEOREM: loop-killer-2

$$\begin{array}{l} (\mathrm{listp}\,(\mathit{msg}) \\ \ \wedge \ \ \mathrm{boolp}\,(\mathrm{car}\,(\mathit{msg})) \\ \ \wedge \ \ \mathrm{bvp}\,(\mathrm{cdr}\,(\mathit{msg})) \\ \ \wedge \ \ (\mathit{ts} \in \mathbf{N}) \\ \ \wedge \ \ (\mathit{tr} \in \mathbf{N}) \\ \ \wedge \ \ (w \not\simeq 0) \\ \ \wedge \ \ (r \not\simeq 0) \end{array}$$

$$
\begin{array}{ll}
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & \text{rate-proximity}\,(w,\, r) \\
\wedge & (nq \in \mathbf{N}) \\
\wedge & (\mathtt{3} \not< nq) \\
\wedge & (dw \in \mathbf{N}) \\
\wedge & (\mathtt{1} \not< dw) \\
\wedge & \text{boolp}\,(\mathit{flg1}) \\
\wedge & \text{boolp}\,(\mathit{flg2}) \\
\wedge & \text{b-xor}\,(\mathit{flg1},\, \mathit{flg2})) \\
\rightarrow & (\text{recv}\,(\text{len}\,(\mathit{msg}), \\
& \qquad \mathit{flg1}, \\
& \qquad \mathtt{10}, \\
& \qquad \text{app}\,(\text{det}\,(\text{listn}\,(nq,\, \mathtt{'q}),\, \mathit{oracle1}), \\
& \qquad\quad \text{app}\,(\text{det}\,(\text{warp}\,(\text{smooth}\,(\mathit{flg2},\, \text{cell}\,(\mathit{flg1},\, \mathtt{4} - dw,\, \mathtt{13},\, \text{car}\,(\mathit{msg}))), \\
& \qquad\qquad\qquad\qquad ts, \\
& \qquad\qquad\qquad\qquad tr, \\
& \qquad\qquad\qquad\qquad w, \\
& \qquad\qquad\qquad\qquad r), \\
& \qquad\qquad\qquad\quad \mathit{oracle}), \\
& \qquad\qquad\quad \mathit{rest}))) \\
= & \quad \text{cons}\,(\text{car}\,(\mathit{msg}), \\
& \qquad\quad \text{recv}\,(\text{len}\,(\text{cdr}\,(\mathit{msg})),\, \text{csig}\,(\mathit{flg1},\, \text{car}\,(\mathit{msg})),\, \mathtt{10},\, \mathit{rest}))
\end{array}
$$

EVENT: Disable recv-app-listn.

```
; We now turn to the base case in which msg of of len 1.  Our killer lemma
; will be named loop-killer-0.  This case is special because the cell is not
; always followed by an edge.  We'll do a blow-by-blow derivation of loop-killer-0,
; starting with the lhs of the loop concl; the killer lemma will reduce it to the
; rhs, which is just msg.

#|
The lhs of loop-killer-0 is initially:
(RECV (LEN MSG)
      FLG1 10
      (APP (DET (LISTN NQ 'Q) ORACLE1)
           (DET (WARP (CDRN DW
                            (SMOOTH FLG2
                                    (APP (CDR (CELLS FLG1 5 13 MSG))
                                         (LISTN P2 T))))
                      TS TR W R)
                ORACLE2)))
where (listp msg) and (nlistp (cdr msg)) are known.
```

```
|#
```

THEOREM: cdr-app-cell-rest
$(m \not\simeq 0)$
$\rightarrow$ (cdr (app (cell (*flg1*, m, n, *bit*), *rest*))
= app (cell (*flg1*, $m - 1$, n, *bit*), *rest*))

THEOREM: smooth-app-cell-rest
$((n \not\simeq 0) \wedge$ b-xor (*flg2*, *flg1*))
$\rightarrow$ (smooth (*flg2*, app (cell (*flg1*, m, n, *bit*), *rest*))
= app (smooth (*flg2*, cell (*flg1*, m, n, *bit*)),
smooth (csig (*flg1*, *bit*), *rest*)))

```
#|
The lhs is now:
(RECV (LEN MSG)
      FLG1 10
      (APP (DET (LISTN NQ 'Q) ORACLE1)
           (DET (WARP (app (smooth flg2
                                   (cell flg1 (difference 4 dw) 13 (car msg)))
                           (smooth (csig flg1 (car msg)) (listn p2 t)))
                      TS TR W R)
                ORACLE2)))
We want to drive the warp inside the app and we will use warp-app to do it by
TARGETing the warp above.  This introduces a warp-smooth-cell instance, which
we need to analyze, and another warp-app instance about what happens after this
cell.  Normally we know what happens next is an edge, so we have a 'q there and
warp-app-listn-q is used instead of warp-app to do this work.  But here there is
no trailing 'q.  Luckily, we don't need to analyze what goes on after this cell
and so we'll just leave the trailing warp unsimplified.

So the lhs becomes
(RECV (LEN MSG)
      FLG1 10
      (APP (DET (LISTN NQ 'Q) ORACLE1)
           (DET (app (warp (smooth flg2
                                   (cell flg1 (difference 4 dw) 13 (car msg)))
                           ts tr w r)
                     (warp (app (lst* (smooth flg2 (cell flg1 (difference 4 dw) 13 (car msg
                                      ts tr w r)
                                (smooth (csig flg1 (car msg)) (listn p2 t)))
                           (ts* (smooth flg2 (cell flg1 (difference 4 dw) 13 (car msg)))
                                ts tr w r)
                           (tr* (smooth flg2 (cell flg1 (difference 4 dw) 13 (car msg)))
```

```
                                ts tr w r)
                           w r))
                ORACLE2)))
and we can apply det-app to produce
(RECV (LEN MSG)
      FLG1 10
      (APP (DET (LISTN NQ 'Q) ORACLE1)
           (app (det (warp (smooth flg2
                                   (cell flg1 (difference 4 dw) 13 (car msg)))
                           ts tr w r)
                     oracle2)
                (det ...))))
where we really don't care what is in the last det because it is beyond the
only cell we will read.

But now we can apply loop-killer-2 to this and get (list (car msg)).

|#

; The -0a below reminds us that this rule has a TARGET in it.
```

THEOREM: loop-killer-0a

$$
\begin{array}{ll}
 & (\mathrm{listp}\,(msg) \\
\wedge & \mathrm{boolp}\,(\mathrm{car}\,(msg)) \\
\wedge & (\mathrm{cdr}\,(msg) = \mathbf{nil}) \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (w \neq 0) \\
\wedge & (w \in \mathbf{N}) \\
\wedge & (r \neq 0) \\
\wedge & (r \in \mathbf{N}) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & \mathrm{rate\text{-}proximity}\,(w,\, r) \\
\wedge & (nq \in \mathbf{N}) \\
\wedge & (3 \not< nq) \\
\wedge & (dw \in \mathbf{N}) \\
\wedge & (1 \not< dw) \\
\wedge & \mathrm{boolp}\,(\mathit{flg1}) \\
\wedge & \mathrm{boolp}\,(\mathit{flg2}) \\
\wedge & \text{b-xor}\,(\mathit{flg1},\, \mathit{flg2})) \\
\rightarrow & (\mathrm{recv}\,(\mathrm{len}\,(msg), \\
 & \quad \mathit{flg1}, \\
 & \quad 10,
\end{array}
$$

app (det (listn (*nq*, 'q), *oracle1*),
det (target (warp (cdrn (*dw*,
smooth (*flg2*,
app (cdr (cells (*flg1*, 5, 13, *msg*)),
listn (*p2*, **t**)))),
ts,
tr,
w,
r)),
oracle2)))
= *msg*)

```
; We now remove the TARGET.
```

THEOREM: loop-killer-0
(listp (*msg*)
∧ boolp (car (*msg*))
∧ (cdr (*msg*) = **nil**)
∧ (*ts* ∈ **N**)
∧ (*tr* ∈ **N**)
∧ (*w* ≠ 0)
∧ (*w* ∈ **N**)
∧ (*r* ≠ 0)
∧ (*r* ∈ **N**)
∧ (*tr* ≮ *ts*)
∧ (*tr* < (*ts* + *w*))
∧ rate-proximity (*w*, *r*)
∧ (*nq* ∈ **N**)
∧ (3 ≮ *nq*)
∧ (*dw* ∈ **N**)
∧ (1 ≮ *dw*)
∧ boolp (*flg1*)
∧ boolp (*flg2*)
∧ b-xor (*flg1*, *flg2*))
→ (recv (len (*msg*),
flg1,
10,
app (det (listn (*nq*, 'q), *oracle1*),
det (warp (cdrn (*dw*,
smooth (*flg2*,
app (cdr (cells (*flg1*, 5, 13, *msg*)),
listn (*p2*, **t**)))),
ts,

```
                                        tr,
                                        w,
                                        r),
                                   oracle2)))
         =    msg)
```

Event: Disable cdr-app-cell-rest.

```
; The two hypotheses about flg1 and flg2 being boolp were added late in the
; development of the loop theorem and I have to prove that the inductive
; instantiation satisfies them.
```

Theorem: boolp-b-not
boolp (b-not (x))

Theorem: boolp-csig
boolp (*flg*) $\rightarrow$ boolp (csig (*flg*, *bit*))

Theorem: loop

```
 (bvp (msg)
  ∧   (ts ∈ N)
  ∧   (tr ∈ N)
  ∧   (w ≄ 0)
  ∧   (r ≄ 0)
  ∧   (tr ≮ ts)
  ∧   (tr < (ts + w))
  ∧   rate-proximity (w, r)
  ∧   (nq ∈ N)
  ∧   (3 ≮ nq)
  ∧   (dw ∈ N)
  ∧   (1 ≮ dw)
  ∧   boolp (flg1)
  ∧   boolp (flg2)
  ∧   b-xor (flg1, flg2))
 →    (recv (len (msg),
             flg1,
             10,
             app (det (listn (nq, 'q), oracle1),
                  det (warp (cdrn (dw,
                                   smooth (flg2,
                                           app (cdr (cells (flg1, 5, 13, msg)),
                                                listn (p2, t)))),
                        ts,
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\quad tr, \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad w, \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad r), \\
\qquad\qquad\qquad\qquad\qquad\quad oracle2))) \\
\qquad = \quad msg)
\end{array}
$$

```
; The proof idea here is to force the expansion of (bvp msg).  The nil
; case goes through.  The (listp msg) case allows the application of
; top-async-send, which converts the conclusion to an instance of our
; loop form.
```

THEOREM: top

$$
\begin{array}{ll}
& (\text{bvp}\,(msg) \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (w \not\simeq 0) \\
\wedge & (r \not\simeq 0) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w)) \\
\wedge & \text{rate-proximity}\,(w,\, r) \\
\wedge & (p1 \in \mathbf{N})) \\
\rightarrow & (\text{recv}\,(\text{len}\,(msg),\, \mathbf{t},\, 10,\, \text{async}\,(\text{send}\,(msg,\, p1,\, 5,\, 13,\, p2),\, ts,\, tr,\, w,\, r,\, oracle)) \\
& \quad = \quad msg)
\end{array}
$$

```
; JSM
; July 31, 1991
```

```
; That concludes the main theorem.  During the course of discovering
; the proof above I also did some related work that I wish to preserve
; and hence include in this events file.  The first is the development
; of the algebraic identities for the recursive functions used to talk
; about warping.
```

EVENT: Enable nendp-alg.

THEOREM: nts*-alg-lemma1

$$
\begin{array}{ll}
& ((x \in \mathbf{N}) \\
\wedge & ((n * w) = (r + x)) \\
\wedge & (n \in \mathbf{N}) \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (w \neq 0) \\
\wedge & (w \in \mathbf{N})
\end{array}
$$

$$
\begin{array}{ll}
\wedge & (r \neq 0) \\
\wedge & (r \in \mathbf{N})) \\
\rightarrow & (\neg\ \mathrm{nendp}\,(n,\ ts,\ r + ts + x,\ w))
\end{array}
$$

THEOREM: nts*-alg-lemma2

$$
\begin{array}{ll}
((x \in \mathbf{N}) & \\
\wedge & (r \neq 0) \\
\wedge & (r \in \mathbf{N}) \\
\wedge & (\neg\ \mathrm{nendp}\,(n,\ ts,\ r + ts + x,\ w)) \\
\wedge & (n \in \mathbf{N}) \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (w \neq 0) \\
\wedge & (w \in \mathbf{N}) \\
\wedge & ((ts + x) \not< ts) \\
\wedge & ((ts + x) < (ts + w))) \\
\rightarrow & ((\mathrm{nts+}\,(n,\ ts,\ r + ts + x,\ w) \\
& \quad + \quad (w * (((r \\
& \qquad\qquad + \quad ts \\
& \qquad\qquad + \quad x \\
& \qquad\qquad + \quad (r * (((w * \mathrm{nlst+}\,(n,\ ts,\ r + ts + x,\ w)) \\
& \qquad\qquad\qquad\qquad - \quad ((r + ts + x) \\
& \qquad\qquad\qquad\qquad\qquad - \quad \mathrm{nts+}\,(n, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad ts, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad r \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad + \quad ts \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad + \quad x, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad w))) \\
& \qquad\qquad\qquad\qquad \div \quad r))) \\
& \qquad\qquad - \quad \mathrm{nts+}\,(n,\ ts,\ r + ts + x,\ w)) \\
& \qquad\quad \div \quad w))) \\
& \quad = \quad (ts + (w * ((x + (r * (((n * w) - x) \div r))) \\
& \qquad\qquad\qquad \div \quad w))))
\end{array}
$$

EVENT: Disable difference-difference.

EVENT: Disable difference-difference-other.

```
; Acknowledgement:  This lemma was finally stated accurately by Matt Wilding.
```

THEOREM: nts*-alg

$$
\begin{array}{ll}
((n \in \mathbf{N}) & \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (tr \in \mathbf{N})
\end{array}
$$

$$
\begin{array}{ll}
\wedge & (w \not\simeq 0) \\
\wedge & (r \not\simeq 0) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w))) \\
\rightarrow & (\text{nts}^*\,(n,\, ts,\, tr,\, w,\, r) \\
 & \quad = \quad (ts + (w * (((tr + (r * (((n * w) - (tr - ts)) \\
 & \qquad\qquad\qquad\qquad\qquad\qquad \div \quad r))) \\
 & \qquad\qquad\qquad\quad - \quad ts) \\
 & \qquad\qquad\qquad \div \quad w))))
\end{array}
$$

```
; The following lemma ought to be called ntr*-alg-lemma1 because it is the
; first in direct support of ntr*-alg.  But it is analogous to the ...lemma2
; lemmas of the other algebraic theorems (because its lemma1 needs were met by
; a prior lemma1).
```

THEOREM: ntr*-alg-lemma2

$$
\begin{array}{ll}
 & ((x \in \mathbf{N}) \\
\wedge & (r \not\simeq 0) \\
\wedge & (r \in \mathbf{N}) \\
\wedge & (\neg\ \text{nendp}\,(n,\, ts,\, r + ts + x,\, w)) \\
\wedge & (n \in \mathbf{N}) \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (w \not\simeq 0) \\
\wedge & (w \in \mathbf{N}) \\
\wedge & ((ts + x) \not< ts) \\
\wedge & ((ts + x) < (ts + w))) \\
\rightarrow & ((r \\
 & \quad + \quad ts \\
 & \quad + \quad x \\
 & \quad + \quad (r * (((w * \text{nlst+}\,(n,\, ts,\, r + ts + x,\, w)) \\
 & \qquad\qquad - \quad ((r + ts + x) \\
 & \qquad\qquad\qquad - \quad \text{nts+}\,(n,\, ts,\, r + ts + x,\, w))) \\
 & \qquad\quad \div \quad r))) \\
 & \quad = \quad (ts + x + (r * (((n * w) - x) \div r))))
\end{array}
$$

THEOREM: ntr*-alg

$$
\begin{array}{ll}
 & ((n \in \mathbf{N}) \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (w \not\simeq 0) \\
\wedge & (r \not\simeq 0) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w))) \\
\rightarrow & (\text{ntr}^*\,(n,\, ts,\, tr,\, w,\, r)
\end{array}
$$

$$= \quad (tr + (r * (((n * w) - (tr - ts)) \div r))))$$

THEOREM: nlst*-alg-lemma2

$$\begin{array}{ll}
((x \in \mathbf{N}) & \\
\wedge & (r \neq 0) \\
\wedge & (r \in \mathbf{N}) \\
\wedge & (\neg\ \mathrm{nendp}\,(n,\ ts,\ r + ts + x,\ w)) \\
\wedge & (n \in \mathbf{N}) \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (w \neq 0) \\
\wedge & (w \in \mathbf{N}) \\
\wedge & ((ts + x) \not< ts) \\
\wedge & ((ts + x) < (ts + w))) \\
\rightarrow & ((\mathrm{nlst}+\,(n,\ ts,\ r + ts + x,\ w) \\
& \quad - \quad (((ts + x + (r * (((n * w) - x) \div r))) \\
& \qquad - \quad \mathrm{nts}+\,(n,\ ts,\ r + ts + x,\ w)) \\
& \qquad \div \quad w)) \\
& \quad = \quad (n - ((x + (r * (((n * w) - x) \div r))) \div w)))
\end{array}$$

THEOREM: nlst*-alg

$$\begin{array}{ll}
((n \in \mathbf{N}) & \\
\wedge & (ts \in \mathbf{N}) \\
\wedge & (tr \in \mathbf{N}) \\
\wedge & (w \not\simeq 0) \\
\wedge & (r \not\simeq 0) \\
\wedge & (tr \not< ts) \\
\wedge & (tr < (ts + w))) \\
\rightarrow & (\mathrm{nlst}^*\,(n,\ ts,\ tr,\ w,\ r) \\
& \quad = \quad (n - (((tr + (r * (((n * w) - (tr - ts)) \div r))) \\
& \qquad\qquad - \quad ts) \\
& \qquad\qquad \div \quad w)))
\end{array}$$

EVENT: Disable nendp-alg.

EVENT: Disable nlst+-alg.

EVENT: Disable nts+-alg.

EVENT: Disable nts*-alg-lemma1.

EVENT: Disable nts*-alg-lemma2-hack1.

EVENT: Disable nts*-alg-lemma2.

EVENT: Disable nts*-alg.

EVENT: Disable ntr*-alg-lemma2-hack1.

EVENT: Disable ntr*-alg-lemma2.

EVENT: Disable ntr*-alg.

EVENT: Disable nlst*-alg-lemma2-hack1.

EVENT: Disable nlst*-alg-lemma2.

EVENT: Disable nlst*-alg.

```
; The second piece of preserved work is the proof that deterministic fuzzy edge
; detection is impossible.

; An Aside:  Does the Treatment of (X) Preclude Edge Detection?
; May 28, 1991.

; (Note: In this section we adopt the convention that when a list is
; treated as a series of signals arriving at a pin then the last
; (right-most) element of the list is the first signal to arrive.
; This convention is at variance with that used throughout the rest of
; our asychronous work, where the signals arrive in the order in which
; they appear in the list.  We adopted this convention to make our
; pictures of edges and our register chains look traditional.  None of
; the work in this section is used outside of this section.  The whole
; point is to explain why we use Q (which is nondeterministically t or
; f on each occurrence) rather than X (which, in the words of Bishop
; Brock, is "your worst nightmare").)

; I have been trying for some weeks, off and on, to write an edge
; detector that could tolerate an (X) at the edge and never be
; undefined.  I have proved that is impossible.  The formalization
; and proof are given below, after an informal sketch of the problem.

; Let a "fuzzy edge" be a sequence of signals that is initially
; all f's and then becomes all t's -- except that between
; the last f and the first t is an x, i.e.,
```

```
;    - - - - - - - -
;                  \
;                    - - - - - - - -
; ...t t t t t t t x f f f f f f f ...

; A "well-defined" circuit is one that always returns either t or f.
; That is, it never answers x.

; Finally, an "edge detector" is a sequential circuit that sits on the
; line listening to the incoming f's and reporting "no edge yet"; if the
; line eventually goes high, the circuit eventually reports "an edge
; came by."  I am willing to give the circuit designer as many leading
; f's and trailing t's as he wants and I don't care how long after the
; edge he says "an edge came by" as long as he is correct.

; So a "well-defined fuzzy edge detector" is one that always returns t
; or f and successfully detects a fuzzy edge eventually.

; Initially I thought I could build a well-defined fuzzy edge
; detector.  For example, chain three successive pulses into registers
; so that you could see "t x f" in registers r0, r1, and r2, and then
; see if r0 is t while r2 is f.  Well, that doesn't work because at
; the moment before you see "t x f" you see "x f f" and the logic
; described would produce an x.  So you say to yourself, ``there is
; only one x in the sequence, so if I widen the window to "t t x f f"
; in registers r0 through r4, and look for, say, r0 and -r3 or r1 and
; -r4, then that bad case, namely "t x f f f" won't get me.''  But now
; "x f f f f" will get you.  Nevertheless, the feeling persisted that
; since I could have as many t's and f's as I wanted and there was
; only one x, I could somehow (perhaps by majority voting?) protect
; myself from that x while still finding the edge.

; I prove below a theorem that tells me that you can't build a
; well-defined fuzzy edge detector.

; My theorem actually does not consider sequential circuits but
; instead addresses itself to combinational circuits that have access
; to an arbitrarily wide "window" in the signal.  One could imagine,
; for example, that the sequential circuit just chains the signal
; through a sequence of k registers, constantly maintaining a window k
; wide on the signal and testing that window with combinational logic.
; If a fuzzy edge comes through that window, the combinational logic
; circuit must detect it.
```

```
; There are k+2 different views of a fuzzy edge in a window k wide.  For
; example, if k is 6 then the 8 views are:

;   f f f f f f
;   x f f f f f
;   t x f f f f
;   t t x f f f
;   t t t x f f
;   t t t t x f
;   t t t t t x
;   t t t t t t

; I limit myself to combinational expressions in F-AND and F-NOT
; because I believe the rest of the primitives can be defined in terms
; of those two (and here I mean not just for T and F but also for X).
; I define an interpreter for F-AND and F-NOT expressions with
; variables v0, v1, ..., vk, where the variables address the
; corresponding positions in the window.  Call such an expression "a
; candidate fuzzy edge detector of width k."

; I prove that if a candidate fuzzy edge detector of width k is
; well-defined (T or F) on all k+2 views of the window, then it is
; constant!

; This work relies upon certain definitions from Warren and Bishop's
; stuff, namely:

; From (note-lib "/usr/home/brock/constants/lsi/reg")
```

EVENT: Add the shell x, with recognizer function symbol xp and no accessors.

```
; From (note-lib "/usr/home/brock/constants/lsi/reg")
```

DEFINITION:
f-not (a)
$=$ **if** boolp (a) **then** $\neg\ a$
else X **endif**

```
; From (note-lib "/usr/home/brock/constants/lsi/reg")
```

DEFINITION:
f-and $(a,\ b)$
$=$ **if** $(a = \mathbf{f}) \vee (b = \mathbf{f})$ **then f**

elseif $(a = \mathbf{t}) \wedge (b = \mathbf{t})$ **then t**
else x **endif**

DEFINITION:
exprp (x)
= **if** $x \simeq$ **nil then** $(x =$ `'t`$) \vee (x =$ `'f`$) \vee (x \in \mathbf{N})$
elseif car $(x) =$ `'f-not` **then** exprp (cadr (x))
$\wedge$ (cddr $(x) =$ **nil**)
else (car $(x) =$ `'f-and`)
$\wedge$ exprp (cadr (x))
$\wedge$ exprp (caddr (x))
$\wedge$ (cdddr $(x) =$ **nil**) **endif**

DEFINITION:
max-var (x)
= **if** $x \simeq$ **nil**
then if $(x =$ `'t`$) \vee (x =$ `'f`$)$ **then** `0`
else x **endif**
elseif car $(x) =$ `'f-not` **then** max-var (cadr (x))
else max (max-var (cadr (x)), max-var (caddr (x))) **endif**

DEFINITION: width $(x) = (1 +$ max-var $(x))$

```
; Any expression is a candidate for an edge detector of size k, where
; k is the width of the expression.

; To evaluate an expression of a given width, k, we must have a vector
; of length k which assigns values (positionally) to each of the k
; variables.  By convention, if a vector is insufficiently long or
; contains a non-Boolean assignment to a variable, we will assume it
; assigns (X) to that variable.  Thus, the empty vector is a
; convenient way to map all variables to (X).  This is unimportant in
; our main result but is used in a subsequent result that shows
; that an expression that is defined on nil is constant.
```

DEFINITION:
var-val $(n,$ *vector*$)$
= **if** boolp (nth $(n,$ *vector*$)$) **then** nth $(n,$ *vector*$)$
else x **endif**

```
; Here is the interpreter for expressions wrt a given vector:
```

DEFINITION:

val (x, *vector*)
= **if** $x \simeq$ **nil**
then if $x =$ `'t` **then t**
elseif $x =$ `'f` **then f**
else var-val (x, *vector*) **endif**
elseif car (x) = `'f-not` **then** f-not (val (cadr (x), *vector*))
else f-and (val (cadr (x), *vector*), val (caddr (x), *vector*)) **endif**

EVENT: Enable boolp.

THEOREM: var-val-is-x-or-boolp
(var-val (x, *vector*) $\neq$ X) $\rightarrow$ boolp (var-val (x, *vector*))

THEOREM: val-is-x-or-boolp
(val (x, *vector*) $\neq$ X) $\rightarrow$ boolp (val (x, *vector*))

EVENT: Disable boolp.

```
; We need to generate the complete list of k+2 views of an edge coming
; thorugh a window of width k.  We enumerate the edges as suggested
; below, e.g., 0 is the one containing all f's, k+1 contains all t's,
; etc.

; 5   t t t t
; 4   t t t x
; 3   t t x f
; 2   t x f f
; 1   x f f f
; 0   f f f f

; The order of enumeration is important to our proof.  In particular,
; if you think of the edges enumerated as above, then an integer, e.g.,
; 3, can either represent the given edge or all the edges at and below that
; one.  And we sometimes induct on these integers to build the set of all
; edges.  We use the property of the enumeration that if a variable is
; Boolean (never x) in all the edges at or below i, then it is in fact
; constant.
```

DEFINITION:
edge (k, i)
= **if** $k \simeq$ `0` **then nil**
elseif $i < k$ **then** app (edge ($k-1$, i), list (**f**))
elseif $i = k$ **then** app (edge ($k-1$, i), list (X))
else app (edge ($k-1$, $i-1$), list (**t**)) **endif**

```
; Note: In the original development of this theorem, we used the
; natural numbers to encode both edges and sets of edges.  This
; restricted our evaluation function to work only on edges and thus
; restricted what we could say about the value of expressions.  We
; therefore decided to represent edges as bit vectors as above.  But
; to preserve exactly the structure of the proof, we will prove the
; old definition of var-val in var-val-edge below.
```

THEOREM: length-edge
$\text{len}\,(\text{edge}\,(k,\ i)) = \text{fix}\,(k)$

EVENT: Enable cdrn-app.

THEOREM: cdrn-too-big
$(n \not< \text{len}\,(lst)) \rightarrow (\neg\ \text{listp}\,(\text{cdrn}\,(n,\ lst)))$

THEOREM: lessp-n-1
$(n < \mathtt{1}) = (n \simeq \mathtt{0})$

THEOREM: nth-edge
$((n \in \mathbf{N}) \wedge (k \in \mathbf{N}) \wedge (k \not\simeq \mathtt{0}))$
$\rightarrow\ (\text{car}\,(\text{cdrn}\,(n,\ \text{edge}\,(k,\ i)))$
$=\ \mathbf{if}\ n < k$
$\mathbf{then\ if}\ (1 + n) < i\ \mathbf{then\ t}$
$\mathbf{elseif}\ (1 + n) = i\ \mathbf{then}\ \text{x}$
$\mathbf{else\ f\ endif}$
$\mathbf{else}\ \mathtt{0}\ \mathbf{endif})$

THEOREM: var-val-edge
$((n \in \mathbf{N}) \wedge (k \in \mathbf{N}) \wedge (k \not\simeq \mathtt{0}) \wedge (n < k))$
$\rightarrow\ (\text{var-val}\,(n,\ \text{edge}\,(k,\ i))$
$=\ \mathbf{if}\ n < k$
$\mathbf{then\ if}\ (1 + n) < i\ \mathbf{then\ t}$
$\mathbf{elseif}\ (1 + n) = i\ \mathbf{then}\ \text{x}$
$\mathbf{else\ f\ endif}$
$\mathbf{else}\ \text{x}\ \mathbf{endif})$

EVENT: Disable var-val.

EVENT: Disable edge.

```
; We now develop the idea that an expression is "well-defined", i.e.,
; its value is never x on any edge of the appropriate size.  To
```

```
; do this we have to check the value on every edge.
```

DEFINITION:
all-edges $(k,\ i)$
= **if** $i \simeq 0$ **then** list (edge $(k,\ 0)$)
else cons (edge $(k,\ i)$, all-edges $(k,\ i-1)$) **endif**

```
; The following function determines that x is well-defined for all
; vectors in a given set test-set:
```

DEFINITION:
well-defined1 $(x,$ *test-set*$)$
= **if** *test-set* $\simeq$ **nil then t**
else (val $(x,$ car (*test-set*$)) \neq$ X)
$\wedge$ well-defined1 $(x,$ cdr (*test-set*$))$ **endif**

```
; So an expression is well-defined if it is well-defined on all the edges
; of the appropriate size.
```

DEFINITION:
well-defined (x) = well-defined1 $(x,$ all-edges (width (x), 1 + width (x)))

```
; The theorem we wish to prove is:

; (implies (and (exprp x)
;               (well-defined x))
;          (equal (val x (edge (width x) i))
;                 (val x (edge (width x) 0))))

; That is, if an expression is well-defined on all edges, then it is
; constant on all edges.
```

THEOREM: val-on-successive-edges-is-constant-when-defined
(exprp (x)
$\wedge$ $(i \not\simeq 0)$
$\wedge$ $(k \not< \text{width}\,(x))$
$\wedge$ (val $(x,$ edge $(k,\ i)) \neq$ X)
$\wedge$ (val $(x,$ edge $(k,\ i-1)) \neq$ X))
$\rightarrow$ (val $(x,$ edge $(k,\ i))$ = val $(x,$ edge $(k,\ i-1)$))

THEOREM: numberp-max-var
exprp $(x) \rightarrow$ (max-var $(x) \in \mathbf{N}$)

THEOREM: val-is-x-or-boolp-2
$((\text{val}\,(x,\ \textit{vector}) \neq \text{X}) \wedge \text{val}\,(x,\ \textit{vector})) \rightarrow (\text{val}\,(x,\ \textit{vector}) = \mathbf{t})$

THEOREM: everything-defined-at-0
$((\textit{zero} \simeq \texttt{0}) \wedge \text{exprp}\,(x) \wedge (k \not< \text{width}\,(x)))$
$\rightarrow \quad (\text{val}\,(x,\ \text{edge}\,(k,\ \textit{zero})) \neq \text{X})$

THEOREM: well-defined1-is-a-universal-quantifier
$(\text{exprp}\,(x)$
$\wedge \quad (k \not< \text{width}\,(x))$
$\wedge \quad \text{well-defined1}\,(x,\ \text{all-edges}\,(k,\ i))$
$\wedge \quad (i \not< j))$
$\rightarrow \quad (\text{val}\,(x,\ \text{edge}\,(k,\ j)) \neq \text{X})$

THEOREM: edge-at-non-numberp
$(i \notin \mathbf{N}) \rightarrow (\text{edge}\,(k,\ i) = \text{edge}\,(k,\ \texttt{0}))$

THEOREM: well-defined1-implies-constant
$(\text{exprp}\,(x)$
$\wedge \quad (k \not< \text{width}\,(x))$
$\wedge \quad \text{well-defined1}\,(x,\ \text{all-edges}\,(k,\ i))$
$\wedge \quad (i \not< j))$
$\rightarrow \quad (\text{val}\,(x,\ \text{edge}\,(k,\ j)) = \text{val}\,(x,\ \text{edge}\,(k,\ \texttt{0})))$

THEOREM: edge-beyond-max
$((1 + k) < i) \rightarrow (\text{edge}\,(k,\ i) = \text{edge}\,(k,\ 1 + k))$

THEOREM: well-defined-implies-constant
$(\text{exprp}\,(x) \wedge \text{well-defined}\,(x))$
$\rightarrow \quad (\text{val}\,(x,\ \text{edge}\,(\text{width}\,(x),\ i)) = \text{val}\,(x,\ \text{edge}\,(\text{width}\,(x),\ \texttt{0})))$

```
; Here is a related result.  It says that if an expression is defined
; on the vector that assigns x to each input, then the expression is
; constant.
```

THEOREM: non-x-on-nil-implies-constant
$(\text{exprp}\,(x) \wedge (\text{val}\,(x,\ \mathbf{nil}) \neq \text{X})) \rightarrow (\text{val}\,(x,\ \textit{vector}) = \text{val}\,(x,\ \mathbf{nil}))$

```
; Some stronger conjectures are not valid.  Consider for example,
; "When x is well-defined (on all edges) then x is constant."  This
; conjecture is falsified by the expression '(f-and (f-not 0) 1).
; As can be determined by r-loop, (well-defined '(f-and (f-not 0) 1)).
; But (val '(f-and (f-not 0) 1) (list f f)) = f
; while
;     (val '(f-and (f-not 0) 1) (list f t)) = t.
```

```
; The conjecture "When x is well-defined on some set of vectors s, then
; x is constant on s" is invalidated by the expression 0 on the set s =
; (list (list t) (list f)).

; One is tempted to strengthen the hypothesis above by additionally
; requiring that there exist a vector in s that makes each variable
; of x unknown.  That conjecture is invalidated by x =
; (f-and (f-not 0) 1), again, using the test set:
; (list (list f f) (list (x) f) (list t (x)) (list f t)).
; The first and last test vectors produce different well-defined values
; of x.  Observe that the only difference between this test set
; and the set of edges of size 2 is the last vector.

; Note added in proof: On Thursday, May 23, I listened to Matt
; Kaufmann talk about his "reset results" with Bishop and Warren.  His
; results established many monotonicity properties of dual-eval and
; got me once again considering the question "what is X?"  I had been
; trying to think of X as nondeterministically 1 or 0.  I talked with
; Bishop about what X was that afternoon.  Upon leaving Bishop's
; office at 6pm I had resolved to prove that you couldn't build a
; fuzzy edge detector with F-NOT and F-AND.  The suspicion that you
; couldn't do it had come to me while at Oberwulfach (I spent hours
; trying) but I decided to lay the problem aside afterwards and focus
; on the formalization of asynchronous communication.

; It took me a night and a day (Thursday evening and then Friday, May
; 24) to get the result its raw form (in which I used numbers to
; encode views through the window).  Then I set about cleaning it up
; by going to the vectors used here.  Lisa's birthday, the Memorial
; Day weekend, an air conditioning failure at CLI, and Matt Wilding's
; birthday party all prevented useful work on the weekend.  (But I
; thought a lot about the problem -- it was surprisingly hard to
; define (edge k i) in a way that let me easily prove the var-val
; theorem about it.)  On Tuesday, I worked another three hours before
; finally getting the file in the form shown.

; At that point I sent a message about it to Warren, Bishop and Matt.
; In my message I asked Matt whether these results were derivable from
; his monotonicity results.  His reply is below.

; From kaufmann@CLI.COM Tue May 28 14:44:01 1991
; Received: by CLI.COM (4.1/1); Tue, 28 May 91 14:43:59 CDT
; Date: Tue, 28 May 91 14:44:33 CDT
```

```
; From: Matt Kaufmann <kaufmann@CLI.COM>
; To: moore@CLI.COM
; Cc: Brock@CLI.COM, Hunt@CLI.COM
; Subject: Fuzzy Edge Detection

; Nice going.  Yes, I think that the kind of monotonicity results that I
; proved could be used to get your result (though I don't know if that
; would be any easier than whatever proof you gave).  (Maybe "kind of"
; could even be omitted above.)  The informal argument is as follows.
; Consider for example the following two successive rows.

; (r1)  t x f f f f
; (r2)  t t x f f f

; Notice that these both 'approximate' the following row:

; (r3)  t t f f f f

; Now since circuits are monotone, and since (r1) approximates (r3), we
; have that f(r1) approximates f(r3), where f is the candidate fuzzy
; edge detector.  But since f(r1) is well-defined (i.e. T or F), then
; since T and F can only approximate themselves (respectively), we have
; that f(r1) = f(r3).  A similar argument shows f(r2) = f(r3).
; Therefore f(r1) = f(r2).  Arguing in this way we can show that for all
; successive rows r and r', f(r) = f(r'); hence by an easy induction,
; f(r) = f(r') for all rows r, r'.
```

Index