TRANSSCRIPTS FOR THE PROOF OF THE
ALTERNATING BIT PROTOCOL

Gennetta L. DiVito

ICS-CEC-72

June 1981
The sole purpose of this report is to bring together the many pages of verifier transcripts that were generated in our proof of the Alternating Bit Protocol. A summary of this work is contained in a companion report [Divito 91]. Refer to that report for a description of the problem and a basic outline of the approach. The present report is not self-contained; it is primarily a raw collection of transcripts and is not intended to contain explanatory material.

As mentioned in the summary report, the proof was a full mechanical verification effort. In addition, there were two separate verification systems used in this work: the Gypsy system and the AFFIRM system. We present complete transcripts showing the work done on both systems. It is divided into the following four parts:

1. Text of the Gypsy model of the protocol and the intermediate specifications. It includes the supporting lemmas used in the Gypsy part of the proof. It is followed by the output of the verification condition (VC) generation.

2. Prover transcripts for the proofs of the VCs. These were done with the Gypsy prover.

3. Prover transcripts for the proofs of those supporting lemmas that were proved with the Gypsy prover.

4. Transcripts from the AFFIRM system. These include a listing of the type specifications expressed in AFFIRM. They are followed by session transcripts for the AFFIRM prover where proofs of the remaining lemmas can be found.

Documentation on the use of both verification systems can be found in the various manuals listed in the references.
References


Gypsy Model of Protocol

[PHOTO: Recording initiated Tue 5-May-81 9:05AM]

LINK FROM CMP, DIVITO, ITY 20

TOPS-20 Command processor (560)
[PHOTO: Logging disabled Tue 5-May-81 9:05AM]
[PHOTO: Logging enabled Tue 5-May-81 9:13AM]

49
49
49

49; The following is a transcript of the modeling and verification of
49; the Alternating Bit Protocol in Gypsy. We begin by displaying the
49; Gypsy text and the generation of verification conditions. This is
49; followed by the various proof transcripts. Most of the proofs are
49; carried out with the Gypsy prover. However, some are performed
49; with the help of the Affirm system. These are documented in a
49; separate set of transcripts.

49
49
49

49; my abp, gyp, abp, lew
49; Abp, Gyp, 2

scope alt-bit-protocol =
begin

type hit = (zero, one);

type message = sequence of character;

type packet = record (msg : message;
  seqno : bit);

type pkt-buf = buffer of packet;

type clk-buf = buffer of boolean;

type msg-buf = buffer of message;

type msg-seq = sequence of message;

type pkt-seq = sequence of packet;

type bit-seq = sequence of bit;

procedure alt-protocol (var source : msg-buf inout;
  var sink : msg-buf inout) =
Gypsy Model of Protocol

begin
  block msg-queue (outto (sink, myid), infrom (source, myid), 1);
  exit false;
  var pkt-send, pkt-rcv, ack-send, ack-rcv : pkt-buf;
  var clock-in, clock-out : clk-buf;
  copbegin
    sender (source, pkt-send, ack-send, clock-in, clock-out);
    medium (pkt-send, pkt-rcv);
    medium (ack-rcv, ack-send);
    receiver (sink, pkt-rcv, ack-rcv);
    timer (clock-in, clock-out)
  end
end

procedure sender (var source : msg-buf<output>);
  var pkt-send : pkt-buf<output>;
  var ack-send : pkt-buf<input>;
  var start : clk-buf<output>;
  var tick : clk-buf<input> =
begin
  block proper-transmission (infrom (source, myid), outto (pkt-send, myid),
                              infrom (ack-send, myid), 1);
  exit false;
  var pack, ack : packet;
  var b : boolean;
  var next : bit := one;
  loop
    assert proper-transmission (infrom (source, myid),
                               outto (pkt-send, myid),
                               infrom (ack-send, myid), 0)
    & next = next-seqnum (outto (pkt-send, myid))
    & next = next-seqnum (infrom (ack-send, myid))
    receive pack, msg from source;
    pack.seqno := next;
    send pack to pkt-send;
    send true to start;
    loop
      assert proper-transmission (infrom (source, myid),
                               outto (pkt-send, myid),
                               infrom (ack-send, myid), 1)
      & infrom (source, myid)
      = unique-msg (outto (pkt-send, myid))
      & size (unique-msg (outto (pkt-send, myid)))
      = size (unique-msg (infrom (ack-send, myid))) + 1
      & outto (pkt-send, myid) ne null (pkt-send)
      & pack = last (outto (pkt-send, myid))
      & next = next-seqnum (infrom (ack-send, myid))
      & pack.seqno = next;
    await
    on receive ack from ack-send
    then if ack.seqno = next
        then leave
Gypsy Model of Protocol

end
  on receive b from tick
  then send back to pkt_send;
  send true to start
end
next := comp (next)
end
end;

procedure medium (var pkt_in : pkt_buf <input>; var pkt_out : pkt_buf <output>) =
begin
  block outto (pkt_out, myid) sub infrom (pkt_in, myid);
  exit false;
  pending
end;

procedure receiver (var sink : msg_buf<output>; var pkt_rcv : pkt_buf<input>; var ack_rcv : pkt_buf<output>) =
begin
  block proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
  outto (ack_rcv, myid), 1);
  exit false;
  var pack : packet;
  var exp : bit := one;
  loop
    assert proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
    outto (ack_rcv, myid), 0)
    & exp = next_segment (infrom (pkt_rcv, myid))
    & exp = next_segment (outto (ack_rcv, myid));
    receive pack from pkt_rcv;
    if exp = pack_segment
      then send pack, msg to sink;
      exp := comp (exp)
    end;
    send pack to ack_rcv
  end;
end;

procedure timer (var clock_in : clk_buf<input>; var clock_out : clk_buf<output>) =
begin
  block true;
  exit false;
  pending
end;

function comp (b : bit) : bit =
begin
  exit result = if b = zero then one else zero fi;
Gypsy Model of Protocol

result := if b = zero then one else zero fi;
end;

name proper_transmission, proper_reception, next_seqnum, msg_lag, seqnums, last_bit from alt_bit_specs;  

name nchanges, unique_msg, repeats, initial_subseq from alt_bit_specs;  

end;

scope alt_bit_specs =
begin

name bit, message, packet, pkt_buf, msg_buf, clk_buf, pkt_seq, msg_seq, bit_seq, comp from alt_bit_protocol;

function msg_lag (s, t : msg_seq; n : integer) : boolean =
begin
  exit (assume result
    iff initial_subseq (s, t) & size (t) = size (s) in [0..n]);
end;

function initial_subseq (u, v : msg_seq) : boolean = pending;
{begin
  exit (assume result iff some s : msg_seq, u a s = v);
end;}

function proper_transmission (source : msg_seq;
  pkt_send, ack_send : pkt_seq; n : integer) : boolean =
begin
  exit (assume result
    iff msg_lag (unique_msg (pkt_send), source, n) &
    size (unique_msg (pkt_send)) = size (unique_msg (ack_send))
    in [0..n] & size (source) = size (unique_msg (ack_send))
    in [0..n] & repeats (pkt_send));
end;

function proper_reception (sink : msg_seq;
  pkt_rcv, ack_rcv : pkt_seq; n : integer) : boolean =
begin
  exit (assume result
Gypsy Model of Protocol

```
iff nsq-lag (sink, unique_msq (pkt_rcv), n)
  & ack_rcv sub pkt_rcv
  & size (sink) = size (unique_msq (ack_rcv))
in [0..n]);
end;

function next-seqnum (os : pkt_seq) : bit =
begin
  exit (assume result = comp (last-bit (seqnums (os))));
end;

function last-bit (os : bit_seq) : bit =
begin
  exit (assume result = if os = null (bit_seq) then zero else last (bs) fi);
end;

function seqnums (ps : pkt_seq) : bit_seq =
begin
  exit (assume result
    = if ps = null (pkt_seq)
    then null (bit_seq)
    else seqnums (nonlast (ps)) <: last (ps),seqno
    fi);
end;

function nchanges (hs : bit_seq) : integer =
begin
  exit (assume result
    = if hs = null (bit_seq)
    then 0
    else if last (hs) = last-bit (nonlast (hs))
    then nchanges (nonlast (hs))
    else nchanges (nonlast (hs)) + 1
    fi
    fi);
end;

function unique-msq (os : pkt_seq) : msg-seq =
begin
  exit (assume result
    = if os = null (pkt_seq)
    then null (msg-seq)
    else if last (os),seqno
    = next-seqnum (nonlast (os))
    then unique-msq (nonlast (os)) <: last (os),msg
    else unique-msq (nonlast (os))
    fi
    fi
    fi
    fi
```
Gypsy Model of Protocol

end;

function repeats (ps : pkt_seq) : boolean =
begin
  exit (assume result
    iff if ps = null (pkt_seq) or nonlast (ps) = null (pkt_seq)
    then true
    else repeats (nonlast (ps))
    & [ last (ps), seqno
    ne next_seqnum (nonlast (ps))
    -> last (ps) = last (nonlast (ps)) ]
  fi);
end;
end;

Abb. Lem. 5

scope lemmas =
begin

lemma prop_trans_1 (u: msg_seq; x, y: pkt_seq; p: packet; m: message) =
  p, seqno = next_seqnum (x) & p, msg = m
  & proper_transmission (u, x, y, 0)
  -> u <\: m = unique_msg (x <\: p);

lemma prop_trans_2 (u: msg_seq; x, y: pkt_seq; p: packet; m: message) =
  p, seqno = next_seqnum (x) & p, msg = m
  & proper_transmission (u, x, y, 0)
  -> proper_transmission (u <\: m, x <\: p, y, 1);

lemma prop_trans_3 (u: msg_seq; x, y: pkt_seq; p: packet) =
  proper_transmission (u, x, y, 1)
  & x ne null (pkt_seq) & p = last(x)
  -> proper_transmission (u, x <\: p, y, 1);

lemma prop_trans_4 (u: msg_seq; x, y: pkt_seq; p: packet) =
  p, seqno ne next_seqnum (y)
  & proper_transmission (u, x, y, 1)
  -> proper_transmission (u, x, y <\: p, 1);

lemma prop_trans_5 (u: msg_seq; x, y: pkt_seq; m: message) =
  proper_transmission (u, x, y, 0)
  -> proper_transmission (u <\: m, x, y, 1);

lemma prop_trans_6 (u: msg_seq; x, y: pkt_seq) =
  proper_transmission (u, x, y, 0)
Gypsy Model of Protocol

=> proper_transmission (u, x, y, 1);

lemma prop_trans_7 (u: msg_seq; x, y: pkt_seq; p: packet) =
  unique_msg (x) = u
& size (unique_msg (x)) = size (unique_msg (y)) + 1
& p.seano = next_seqnum (y)
& proper_transmission (u, x, y, 1)
=> proper_transmission (u, x, y < p, 0);

lemma prop_rec_1 (u: msg_seq; x, y: pkt_seq; p: packet; m: message) =
  p.seano = next_seqnum (x) & p.mssg = m
& proper_recception (u, x, y, 0)
=> proper_recception (u < m, x < p, y, 1);

lemma prop_rec_2 (u: msg_seq; x, y: pkt_seq; p: packet) =
  proper_recception (u, x, y, 0)
=> proper_recception (u, x < p, y, 1);

lemma prop_rec_3 (u: msg_seq; x, y: pkt_seq) =
  proper_recception (u, x, y, 0) => proper_recception (u, x, y, 1);

lemma prop_rec_4 (u: msg_seq; x, y: pkt_seq; p: packet; m: message) =
  p.seano = next_seqnum (x) & p.seano = next_seqnum (y)
& proper_recception (u, x, y, 0) & p.mssg = m
=> proper_recception (u < m, x < p, y < p, 0);

lemma prop_rec_5 (u: msg_seq; x, y: pkt_seq; p: packet) =
  p.seano ne next_seqnum (x) & p.seano ne next_seqnum (y)
& proper_recception (u, x, y, 0)
=> proper_recception (u, x < p, y < p, 0);

lemma abo_1 (u, v: msg_seq; v, x, y: pkt_seq) =
  proper_transmission (u, x, z, 1)
& proper_recception (v, y, 1)
& x sub w &' z sup y
=> msg_lag (v, u, 1);

lemma main_lemma (s: bit_seq; x, y: pkt_seq) =
  s sub seqnums (x) & x sub y & repeats (y)
& nchanges (seqnums (y)) = nchanges (s) in [0..1]
=> msg_lag (unique_msg (x), unique_msg (y), 1);

lemma interpolate (s: bit_seq; x, y: pkt_seq) =
  s sub seqnums (x) & x sub y & repeats (y)
& nchanges (seqnums (v)) = nchanges (s) in [0..1]
=> nchanges (seqnums (y)) = nchanges (seqnums (x)) in [0..1];

lemma next_comp (x: pkt_seq; p: packet) =
Gypsy Model of Protocol

next_seqnum (x <: p) = comp (p.seqno);

lemma ne_next (x:pkt_seq; p: packet) =
  p.seqno ne next_seqnum (x)
  => next_seqnum (x <: p) = next_seqnum (x);

lemma last_next (x: pkt_seq; p: packet) =
  x ne null (pkt_seq) & p = last (x)
  => p.seqno ne next_seqnum (x);

lemma last_unique (x: pkt_seq; p: packet) =
  x ne null (pkt_seq) & p = last (x)
  => unique_msg (x <: p) = unique_msg (x);

lemma last_repeats (x: pkt_seq; o: packet) =
  x ne null (pkt_seq) & p = last (x) & repeats (x)
  => repeats (x <: o);

lemma eq_msg (u, v: msg_seq) =
  u = v => initial_subseq (u, v);

lemma eq_msg_app (u, v: msg_seq; m: message) =
  u = v => initial_subseq (u <: m, v <: m);

lemma iss_app (u, v: msg_seq; m: message) =
  initial_subseq (u, v) => initial_subseq (u <: m);

lemma iss_trans (u, v, w: msg_seq) =
  initial_subseq (u, v) & initial_subseq (v, w)
  => initial_subseq (u, w);

lemma msg_seq_eq (u, v: msg_seq) =
  msg_seq (u, v, 0) iff u = v;

lemma comp_ne (b1, b2: bit) =
  b1 ne b2 iff comp (b1) = b2;

lemma bit_cases (n: bit) =
  n = zero or n = one;

lemma hist_sub (x, y: pkt_buf) =
  allfrom (x) sub allfrom (y) => allfrom (x) sub allto (y);

lemma app_pktnonnull (x: pkt_seq; p: packet) =
  [(x <: p) = null (pkt_seq)] = false;

lemma app_msgnonnull (u: msg_seq; m: message) =
  [(u <: m) = null (msg_seq)] = false;

lemma app_bitnonnull (s: bit_seq; b: bit) =
  [(s <: b) = null (bit_seq)] = false;
Gypsy Model of Protocol

lemma sub-app (x, y: pkt-seq; o: packet) =
  (assume x sub y -> x <i o sub y <i o);

lemma size-null (u: msg-seq) =
  (assume size (u) = 0) iff u = null (msg-seq);

lemma sub-seqnum (x, y: pkt-seq) =
  (assume x sub y -> seqnums (x) sub seqnums (y));

lemma sub-nchanges (s, t: bit-seq) =
  (assume s sub t -> nchanges (s) le nchanges(t));

lemma nchanges-unique (x: pkt-seq) =
  (assume size (unique-msg (x)) = nchanges (seqnums (x)));

lemma sub-to-lag (x, y: pkt-seq) =
  (assume x sub y & repeats (y) & nchanges (seqnums (y)) = nchanges (seqnums (x)) in [0,1]
  -> initial-subseq (unique-msg (x), unique-msg (y)));

name proper-transmission, proper-reception, next-seqnum, last-bit, seqnums,
  nchanges, unique-msg, repeats, initial-subseq, msg-lag
from alt-bit-secs;

name bit, message, packet, pkt-seq, msg-seq, bit-seq, comp, pkt-buf
from alt-bit-protocol;

end;

4@vsysxxx

[Continuing]

translate app.gyp

No syntax errors detected
No semantic errors detected

Exec-> translate app.lem

No syntax errors detected
No semantic errors detected

Exec-> show status all
Gypsy Model of Protocol

The current design and verification status is:

SCOPE ALT.Bit.PROTOCOL

Waiting for VC generation: AH протокол, Comp, Receiver, Sender
Waiting for pending body to be filled in: Medium, Timer
Types, constants: Bit, Bit_seq, Clkbuf, Message, Msgbuf, Msg_seq, Packet, Pkt_buf, Pkt_seq

SCOPE ALT.Bit.SPECs

Waiting for pending body to be filled in: Initial_subseq
For specifications only: Last_bit, Msg_lag, Mchanges, Next_seqnum, Proper_reception, Proper_transmission, Repeats, Seqnums, Unique_msg

SCOPE Lemmas

Waiting for VC generation: App_1, App_bitnonnull, App_msgnonnull, App_pktnonnull, Bit_cases, Comp_ne, Eq_iss, Eq_iss_app, Hist_sub, Iss_app, Iss_trans, Interpolate, Last_next, Last_unique, Last_repeats, Main_lemma, Msg_lag_eq, Ne_next, Next_comp, Mchanges_unique, Prop_rec_1, Prop_rec_2, Prop_rec_3, Prop_rec_4, Prop_rec_5, Prop_trans_1, Prop_trans_2, Prop_trans_3, Prop_trans_4, Prop_trans_5, Prop_trans_6, Prop_trans_7, Sub_app, Size_null, Sub_seqnum, Sub_to_lag, Sub_mchanges

Exec-> set scope alt.Bit.protocol

Exec-> vcs sender

Generating VCs for procedure sender

Found 1st path
Found 2nd path
Found 3rd path
Found 4th path
Found 5th path
Note: loop has no exit paths
Found 6th path

--------------------
Beginning new path...

Assume (unit entry specification)
Gypsy Model of Protocol

true

initializing local variables

entering loop...

Evaluating next_segment (infrom (ack_send, myid))
Continuing in path...

Evaluating next_segment (outto (pkt_send, myid))
Continuing in path...

Evaluating proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 0)
Continuing in path...

Assert proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 0)

& next = next_segment (outto (pkt_send, myid))
& next = next_segment (infrom (ack_send, myid))

---

Must verify Assert condition

Verification condition sender=1
  1: next_segment (null (*sectype*)) = one
  2: proper_transmission (null (*sectype*), null (*sectype*), null (*sectype*), 0)
---

End of path

---

Beginning new path...
continuing in loop...

Assume (from last assertion)

proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 0)

& next = next_segment (outto (pkt_send, myid))
& next = next_segment (infrom (ack_send, myid))

Receive pack,mssq from source
---

Must verify (Receive blocked) condition

Verification condition senders=5
  H1: empty (source=1)
  H2: next_segment (infrom (ack_send, myid)) = next
  H3: next_segment (outto (pkt_send, myid)) = next
  H4: infrom (source, myid) = infrom (source=1, myid)
Gypsy Model of Protocol

H5: outto (source, myid) = outto (source#1, myid)
H6: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
     infrom (ack_send, myid), 0)

==>
C1: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
     infrom (ack_send, myid), 1)

Continuing in path...
binding due to receive..

pack := pack with (.mssg:=pack1[mssg])
pack := pack with (.seqno:=next)
Send pack to pkt_send

Must verify (Send blocked) condition
Verification condition sender=6
H1: infrom (source, myid) @ {seq: pack1[mssg]} = infrom (source#2, myid)
H2: next_seqnum (infrom (ack_send, myid)) = next
H3: next_seqnum (outto (pkt_send, myid)) = next
H4: infrom (pkt_send, myid) = infrom (pkt_send#1, myid)
H5: outto (pkt_send, myid) = outto (pkt_send#1, myid)
H6: outto (source, myid) = outto (source#2, myid)
H7: full (pkt_send#1)
H8: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
     infrom (ack_send, myid), 0)

==>
C1: proper_transmission (infrom (source, myid) @ {seq: pack1[mssg]},
     outto (pkt_send, myid), infrom (ack_send, myid), 1)

Continuing in path...
Send true to start

Must verify (Send blocked) condition
Verification condition sender=7
H1: infrom (source, myid) @ {seq: pack1[mssg]} = infrom (source#2, myid)
H2: outto (pkt_send, myid)
     @ {seq: pack with (.mssg:=pack1[mssg]; .seqno:=next)}
     = outto (pkt_send#2, myid)
H3: next_seqnum (infrom (ack_send, myid)) = next
H4: next_seqnum (outto (pkt_send, myid)) = next
H5: infrom (pkt_send, myid) = infrom (pkt_send#2, myid)
H6: infrom (start, myid) = infrom (start#1, myid)
H7: outto (source, myid) = outto (source#2, myid)
H8: outto (start, myid) = outto (start#1, myid)
H9: full (start#1)
H10: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
     infrom (ack_send, myid), 0)

==>
C1: proper_transmission (infrom (source, myid) @ {seq: pack1[mssg]},
     outto (pkt_send, myid)
     @ {seq: pack with (.mssg:=pack1[mssg];
     .seqno:=next)})
Gypsy Model of Protocol

infrom (ack-send, myid), 1)
Continuing in path...
entering loop...

Evaluating next_seqnum (infrom (ack-send, myid))
Continuing in path...

Evaluating unique_msg (infrom (ack-send, myid))
Continuing in path...

Evaluating unique_msg (outto (pkt-send, myid))
Continuing in path...

Evaluating unique_msg (outto (pkt-send, myid))
Continuing in path...

Evaluating proper_transmission (infrom (source, myid), outto (pkt-send, myid),
infrom (ack-send, myid), 1)
Continuing in path...

Assert proper_transmission (infrom (source, myid), outto (pkt-send, myid),
infrom (ack-send, myid), 1)
& infrom (source, myid) = unique_msg (outto (pkt-send, myid))
& size (unique_msg (outto (pkt-send, myid)))
= size (unique_msg (infrom (ack-send, myid))) + 1
& outto (pkt-send, myid) ne null (pkt-seq)
& pack = last (outto (pkt-send, myid))
& next = next_seqnum (infrom (ack-send, myid)) & pack.segno = next

Must verify Assert condition

Verification condition sender=13

H1: infrom (source, myid) & [seq: pack1.mssg] = infrom (source*2, myid)
H2: outto (pkt-send, myid)
& [seq: pack with (.mssg:=pack1.mssg; .seqno:=next)]
= outto (pkt-send*2, myid)
H3: next_seqnum (infrom (ack-send, myid)) = next
H4: next_seqnum (outto (pkt-send, myid)) = next
H5: infrom (pkt-send, myid) = infrom (pkt-send*2, myid)
H6: outto (source, myid) = outto (source*2, myid)
H7: proper_transmission (infrom (source, myid), outto (pkt-send, myid),
infrom (ack-send, myid), 0)

-->

C1: infrom (source, myid) & [seq: pack1.mssg]
= unique_msg ( outto (pkt-send, myid)
& [seq: pack with (.mssg:=pack1.mssg; .seqno:=next)])
C2: size (unique_msg ( outto (pkt-send, myid)
& [seq: pack with (.mssg:=pack1.mssg;
& .seqno:=next)])
= size (unique_msg (infrom (ack-send, myid))) + 1
C3: proper_transmission (infrom (source, myid) & [seq: pack1.mssg],
outto (pkt-send, myid)
Gypsy Model of Protocol

```plaintext
>:: (seq: pack with (.msg: =pack=1, msg); 
    .senno: =next)),
    infrom (ack_send, myid), 1)

End of path

```

Beginning new path...
continuing in loop ...

Assume (from last assertion)

```plaintext
proper_transmission (infrom (source, myid), outto (pkt_send, myid),
    infrom (ack_send, myid), 1)
& infrom (source, myid) = unique_msg (outto (pkt_send, myid))
& size (unique_msg (outto (ack_send, myid))) + 1
& outto (pkt_send, myid) ne null (pkt_seq)
& pack = last (outto (pkt_send, myid))
& next = next sennum (infrom (ack_send, myid)) & pack, senno = next
```

Assume (Await blocked)

```plaintext
empty (tick) & empty (ack_send)
```

Blockage Assertion is

```plaintext
proper_transmission (infrom (source, myid), outto (pkt_send, myid),
    infrom (ack_send, myid), 1)
```

End of path

Beginning new path...
continuing in loop ...

Assume (from last assertion)

```plaintext
proper_transmission (infrom (source, myid), outto (pkt_send, myid),
    infrom (ack_send, myid), 1)
& infrom (source, myid) = unique_msg (outto (pkt_send, myid))
& size (unique_msg (outto (ack_send, myid))) + 1
& outto (pkt_send, myid) ne null (pkt_seq)
& pack = last (outto (pkt_send, myid))
```
Gypsy Model of Protocol

& next = next_seqnum (infrom (ack_send, myid)) & pack_seqno = next

Receive ack from ack_send
binding due to receive...

ack := ack#1

Assume (If test failed)

not ack_seqno = next

entering next iteration of loop...

Evaluating next_seqnum (infrom (ack_send, myid))
Continuing in path...

Evaluating unique_msg (infrom (ack_send, myid))
Continuing in path...

Evaluating unique_msg (outto (pkt_send, myid))
Continuing in path...

Evaluating unique_msg (outto (pkt_send, myid))
Continuing in path...

Evaluating proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), )
Continuing in path...

Assert proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 1)
& infrom (source, myid) = unique_msg (outto (pkt_send, myid))
& size (unique_msg (outto (pkt_send, myid)))
& outto (pkt_send, myid) ne null (pkt_seq)
& pack = last (outto (pkt_send, myid))
& next = next_seqnum (infrom (ack_send, myid)) & pack_seqno = next

Must verify Assert condition

Verification condition sender=20
H1: infrom (ack_send, myid) & [seq: ack#1] = infrom (ack_send#3, myid)
H2: pack_seqno = next
H3: next_seqnum (infrom (ack_send, myid)) = next
H4: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
H5: outto (ack_send, myid) = outto (ack_send#3, myid)
H6: outto (pkt_send, myid)[size (outto (pkt_send, myid))] = pack
H7: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1
H8: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 1)
H9: ack#1_seqno ne next
H10: null (pkt_seq) ne outto (pkt_send, myid)
Gypsy Model of Protocol

```plaintext
--> 
C1: next_seqnum (infrom (ack_send, myid) & (seq: ack#1)) = next
C2: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid) & (seq: ack#1))) + 1
C3: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
    infrom (ack_send, myid) & (seq: ack#1), 1)
-----------------------------------
End of path

------------------------------------------------------------------
Beginning new path...
continuing in loop ...

Assume (from last assertion)

proper_transmission (infrom (source, myid), outto (pkt_send, myid),
    infrom (ack_send, myid), 1)
& infrom (source, myid) = unique_msg (outto (pkt_send, myid))
& size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1
& outto (pkt_send, myid) ne null (pkt_send)
& pack = last (outto (pkt_send, myid))
& next = next_seqnum (infrom (ack_send, myid) & pack, seqno = next

Receive b from tick
binding due to receive..

b := #1
Send pack to pkt_send
Continuing in path...
Send true to start

Must verify (Send blocked) condition
Verification condition sender#2?

H1: outto (pkt_send, myid) & (seq: pack) = outto (pkt_send#4, myid)
H2: pack, seqno = next
H3: next_seqnum (infrom (ack_send, myid)) = next
H4: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
H5: infrom (pkt_send, myid) = infrom (pkt_send#4, myid)
H6: infrom (start, myid) = infrom (start#3, myid)
H7: outto (start, myid) = outto (start#3, myid)
H8: outto (pkt_send, myid)[size (outto (pkt_send, myid))] = pack
H9: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1
H10: full (start#3)
H11: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
    infrom (ack_send, myid), 1)
H12: null (pkt_send) ne outto (pkt_send, myid)
```
Gypsy Model of Protocol

-->  
C1: \text{proper-transmission} \ (\text{infrom} \ (\text{source}, \ \text{myid}), \ 
\text{outto} \ (\text{pkt-send}, \ \text{myid}) \ \& \ [\text{seq} : \text{pack}], \ 
\text{infrom} \ (\text{ack-send}, \ \text{myid}), \ 1) \ 

---

Continuing in path...
entering next iteration of loop...

Evaluating \text{next-seqnum} \ (\text{infrom} \ (\text{ack-send}, \ \text{myid}))
Continuing in path...

Evaluating \text{unique-msg} \ (\text{infrom} \ (\text{ack-send}, \ \text{myid}))
Continuing in path...

Evaluating \text{unique-msg} \ (\text{outto} \ (\text{pkt-send}, \ \text{myid}))
Continuing in path...

Evaluating \text{unique-msg} \ (\text{outto} \ (\text{pkt-send}, \ \text{myid}))
Continuing in path...

Evaluating \text{proper-transmission} \ (\text{infrom} \ (\text{source}, \ \text{myid}), \ \text{outto} \ (\text{pkt-send}, \ \text{myid}), \ 
\text{infrom} \ (\text{ack-send}, \ \text{myid}), \ 1)
Continuing in path...

Assert \text{proper-transmission} \ (\text{infrom} \ (\text{source}, \ \text{myid}), \ \text{outto} \ (\text{pkt-send}, \ \text{myid}), \ 
\text{infrom} \ (\text{ack-send}, \ \text{myid}), \ 1)

& \text{infrom} \ (\text{source}, \ \text{myid}) = \text{unique-msg} \ (\text{outto} \ (\text{pkt-send}, \ \text{myid}))
& \text{size} \ (\text{unique-msg} \ (\text{outto} \ (\text{pkt-send}, \ \text{myid})))
= \text{size} \ (\text{unique-msg} \ (\text{infrom} \ (\text{ack-send}, \ \text{myid}))) + 1
& \text{outto} \ (\text{pkt-send}, \ \text{myid}) \ \& \ \text{null} \ (\text{outto} \ (\text{pkt-send}, \ \text{myid}) \ \& \ \text{pack},\text{seqno} = \text{next}

---

Must verify Assert condition

Verification condition \text{sender}=2A
H1: \text{outto} \ (\text{pkt-send}, \ \text{myid}) \ \& \ [\text{seq} : \text{pack}] = \text{outto} \ (\text{pkt-send}^4, \ \text{myid})
H2: \text{pack},\text{seqno} = \text{next}
H3: \text{next-seqnum} \ (\text{infrom} \ (\text{ack-send}, \ \text{myid})) = \text{next}
H4: \text{unique-msg} \ (\text{outto} \ (\text{pkt-send}, \ \text{myid})) = \text{infrom} \ (\text{source}, \ \text{myid})
H5: \text{infrom} \ (\text{pkt-send}, \ \text{myid}) = \text{infrom} \ (\text{pkt-send}^4, \ \text{myid})
H6: \text{outto} \ (\text{pkt-send}, \ \text{myid})[\text{size} \ (\text{outto} \ (\text{pkt-send}, \ \text{myid})))] = \text{pack}
H7: \text{size} \ (\text{unique-msg} \ (\text{outto} \ (\text{pkt-send}, \ \text{myid})))
= \text{size} \ (\text{unique-msg} \ (\text{infrom} \ (\text{ack-send}, \ \text{myid}))) + 1
H8: \text{proper-transmission} \ (\text{infrom} \ (\text{source}, \ \text{myid}), \ \text{outto} \ (\text{pkt-send}, \ \text{myid}), \ 
\text{infrom} \ (\text{ack-send}, \ \text{myid}), \ 1)
H9: \text{null} \ (\text{pkt-send}) \ \& \ \text{outto} \ (\text{pkt-send}, \ \text{myid})

-->  
C1: \text{unique-msg} \ (\text{outto} \ (\text{pkt-send}, \ \text{myid}) \ \& \ [\text{seq} : \text{pack}])
= \text{infrom} \ (\text{source}, \ \text{myid})
C2: \text{size} \ (\text{unique-msg} \ (\text{outto} \ (\text{pkt-send}, \ \text{myid}) \ \& \ [\text{seq} : \text{pack}])
= \text{size} \ (\text{unique-msg} \ (\text{infrom} \ (\text{ack-send}, \ \text{myid}))) + 1
C3: \text{proper-transmission} \ (\text{infrom} \ (\text{source}, \ \text{myid}),


Gypsy Model of Protocol

outto (pkt_send, myid) a [seq: pack],
infrom (ack_send, myid), 1)

End of path

Beginning new path...
continuing in loop...

Assume (from last assertion)

proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 1)
& infrom (source, myid) = unique_msg (outto (pkt_send, myid))
& size (unique_msg (outto (pkt_send, myid)))
= size (unique_msg (infrom (ack_send, myid))) + 1
& outto (pkt_send, myid) ne null (pkt_send)
& pack = last (outto (pkt_send, myid))
& next = next_seqnum (infrom (ack_send, myid)) & pack, seqno = next

Receive ack from ack_send
binding due to receive..

ack := ack#1

Assume (If test succeeded)

ack, seqno = next

Leaving loop...

Evaluating comp (next)
Continuing in path...
next := comp (next)
entering next iteration of loop...

Evaluating next_seqnum (infrom (ack_send, myid))
Continuing in path...

Evaluating next Seqnum (outto (pkt_send, myid))
Continuing in path...

Evaluating proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 0)
Continuing in path...
Assert proper_transmission (infrom (source, myid), outto (pkt_send, myid),
Gypsy Model of Protocol

\[ \text{infrom (ack\_send, myid), 0) } \]
\[ \& \text{next = next\_seqnum (outto (pkt\_send, myid))} \]
\[ \& \text{next = next\_seqnum (infrom (ack\_send, myid))} \]

---

Must verify Assert condition

Verification condition sender=33

H1: infrom (ack\_send, myid) @ \{seq: ack\#1\} = infrom (ack\_send\#3, myid)

H2: ack\#1\_segno = next

H3: pack\_segno = next

H4: next\_seqnum (infrom (ack\_send, myid)) = next

H5: unique\_msg (outto (pkt\_send, myid)) = infrom (source, myid)

H6: outto (ack\_send, myid) = outto (ack\_send\#3, myid)

H7: outto (pkt\_send, myid)\{size (outto (pkt\_send, myid))\} = pack

H8: size (unique\_msg (outto (pkt\_send, myid)))

= size (unique\_msg (infrom (ack\_send, myid))) + 1

H9: proper\_transmission (infrom (source, myid), outto (pkt\_send, myid),
infrom (ack\_send, myid), 1)

H10: null (pkt\_seq) ne outto (pkt\_send, myid)

=>

C1: comp (next)

= next\_seqnum (infrom (ack\_send, myid) @ \{seq: ack\#1\})

C2: comp (next) = next\_seqnum (outto (pkt\_send, myid))

C3: proper\_transmission (infrom (source, myid), outto (pkt\_send, myid),
infrom (ack\_send, myid) @ \{seq: ack\#1\}, 0)

---

End of path

-------------------------------

Exec -> vcs receiver

Generating VCs for procedure receiver

Found 1st path
Vote: loop has no exit paths
Found 2nd path
Found 3rd path

-------------------------------

Beginning new path...

Assume (unit entry specification)
true

initializing local variables

entering loop...
Gypsy Model of Protocol

Evaluating next_seqnum (outto (ack_rcv, myid))
Continuing in path...

Evaluating next_seqnum (infrom (pkt_rcv, myid))
Continuing in path...

Evaluating proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
outto (ack_rcv, myid), 0)
Continuing in path...
Assert proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
outto (ack_rcv, myid), 0)
& exp = next_seqnum (infrom (pkt_rcv, myid))
& exp = next_seqnum (outto (ack_rcv, myid))

Must verify Assert condition
Verification condition receiver14
1: next_seqnum (null ($seqtype$)) = one
2: proper_reception (null ($seqtype$), null ($seqtype$), null ($seqtype$), 0)

End of path

Beginning new path...
continuing in loop ...

Assume (from last assertion)

proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
outto (ack_rcv, myid), 0)
& exp = next_seqnum (infrom (pkt_rcv, myid))
& exp = next_seqnum (outto (ack_rcv, myid))

Receive pack from pkt_rcv

Must verify (Receive blocked) condition
Verification condition receiver5
H1: empty (pkt_rcv1)
H2: next_seqnum (infrom (pkt_rcv, myid)) = exp
H3: next_seqnum (outto (ack_rcv, myid)) = exp
H4: infrom (pkt_rcv, myid) = infrom (pkt_rcv1, myid)
H5: outto (pkt_rcv, myid) = outto (pkt_rcv1, myid)
H6: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
outto (ack_rcv, myid), 0)

C1: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),

Gypsy Model of Protocol

outto (ack_recv, myid), 1)

Continuing in path...

binding due to receive...

pack := pack#1

Assume (If test succeeded)

exp = pack.segno

Send pack.msg to sink

Must verify (Send blocked) condition

Verification condition receiver#8

H1: infrom (pkt_recv, myid) & [seq: pack#1] = infrom (pkt_recv#2, myid)
H2: pack#1.segno = exp
H3: next_seqnum (infrom (pkt_recv, myid)) = exp
H4: next_seqnum (outto (ack_recv, myid)) = exp
H5: infrom (sink, myid) = infrom (sink#1, myid)
H6: outto (pkt_recv, myid) = outto (pkt_recv#2, myid)
H7: outto (sink, myid) = outto (sink#1, myid)
H8: full (sink#1)
H9: proper_reception (outto (sink, myid), infrom (pkt_recv, myid),

outto (ack_recv, myid), 0)

---

C1: proper_reception (outto (sink, myid),

infrom (pkt_recv, myid) & [seq: pack#1],

outto (ack_recv, myid), 1)

Continuing in path...

Evaluating comp (exp)

Continuing in path...

exp := comp (exp)

Send pack to ack_recv

Must verify (Send blocked) condition

Verification condition receiver#8

H1: infrom (pkt_recv, myid) & [seq: pack#1] = infrom (pkt_recv#2, myid)
H2: outto (sink, myid) & [seq: pack#1.msg] = outto (sink#2, myid)
H3: pack#1.segno = exp
H4: next_seqnum (infrom (pkt_recv, myid)) = exp
H5: next_seqnum (outto (ack_recv, myid)) = exp
H6: infrom (ack_recv, myid) = infrom (ack_recv#1, myid)
H7: infrom (sink, myid) = infrom (sink#2, myid)
H8: outto (ack_recv, myid) = outto (ack_recv#1, myid)
H9: outto (pkt_recv, myid) = outto (pkt_recv#2, myid)
H10: full (ack_recv#1)
H11: proper_reception (outto (sink, myid), infrom (pkt_recv, myid),

outto (ack_recv, myid), 0)

-->
Gosy Model of Protocol

C1: proper_reception (out_to (sink, myid) @ [seq: pack1, msg)],
in_from (pkt_rcv, myid) @ [seq: pack1],
out_to (ack_rcv, myid), 1)

Continuing in path...
entering next iteration of loop...

Evaluating next_seqnum (out_to (ack_rcv, myid))
Continuing in path...

Evaluating next_seqnum (in_from (pkt_rcv, myid))
Continuing in path...

Evaluating proper_reception (out_to (sink, myid), in_from (pkt_rcv, myid),
out_to (ack_rcv, myid), 0)
Continuing in path...
Assert proper_reception (out_to (sink, myid), in_from (pkt_rcv, myid),
out_to (ack_rcv, myid), 0)
& exp = next_seqnum (in_from (pkt_rcv, myid))
& exp = next_seqnum (out_to (ack_rcv, myid))

Must verify Assert condition
Verification condition receiver#12
H1: in_from (pkt_rcv, myid) @ [seq: pack1] = in_from (pkt_rcv#2, myid)
H2: out_to (ack_rcv, myid) @ [seq: pack1] = out_to (ack_rcv#2, myid)
H3: out_to (sink, myid) @ [seq: pack1, msg] = out_to (sink#2, myid)
H4: pack1.seqno = exp
H5: next_seqnum (in_from (pkt_rcv, myid)) = exp
H6: next_seqnum (out_to (ack_rcv, myid)) = exp
H7: in_from (ack_rcv, myid) = in_from (ack_rcv#2, myid)
H8: in_from (sink, myid) = in_from (sink#2, myid)
H9: out_to (pkt_rcv, myid) = out_to (pkt_rcv#2, myid)
H10: proper_reception (out_to (sink, myid), in_from (pkt_rcv, myid),
out_to (ack_rcv, myid), 0)

=>
C1: comp (exp)
    = next_seqnum (in_from (pkt_rcv, myid) @ [seq: pack1])
C2: comp (exp) = next_seqnum (out_to (ack_rcv, myid) @ [seq: pack1])
C3: proper_reception (out_to (sink, myid) @ [seq: pack1, msg],
in_from (pkt_rcv, myid) @ [seq: pack1],
out_to (ack_rcv, myid) @ [seq: pack1], 0)

End of path

Beginning new path...
continuing in loop...
Assume (from last assertion)

```
proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
     outto (ack_rcv, myid), 0)
& exp = next_seqnum (infrom (pkt_rcv, myid))
& exp = next_seqnum (outto (ack_rcv, myid))
```

Receive back from pkt_rcv

```
Must verify (Receive blocked) condition
```

```
Verification condition receiver#13
H1: empty (pkt_rcv#1)
H2: next_seqnum (infrom (pkt_rcv, myid)) = exp
H3: next_seqnum (outto (ack_rcv, myid)) = exp
H4: infrom (pkt_rcv, myid) = infrom (pkt_rcv#1, myid)
H5: outto (pkt_rcv, myid) = outto (pkt_rcv#1, myid)
H6: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
     outto (ack_rcv, myid), 0)
==>
C1: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
     outto (ack_rcv, myid), 1)
```

Continuing in path...

binding due to receive..

```
pack := pack#1
```

Assume (If test failed)

```
not exp = pack.seqno
```

Send pack to ack_rcv

```
Must verify (Send blocked) condition
```

```
Verification condition receiver#14
H1: infrom (pkt_rcv, myid) & [seq; pack#1] = infrom (pkt_rcv#2, myid)
H2: next_seqnum (infrom (pkt_rcv, myid)) = exp
H3: next_seqnum (outto (ack_rcv, myid)) = exp
H4: infrom (ack_rcv, myid) = infrom (ack_rcv#3, myid)
H5: outto (ack_rcv, myid) = outto (ack_rcv#3, myid)
H6: outto (pkt_rcv, myid) = outto (pkt_rcv#2, myid)
H7: full (ack_rcv#3)
H8: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
     outto (ack_rcv, myid), 0)
H9: pack#1.seqno ne exp
==>
C1: proper_reception (outto (sink, myid),
     infrom (pkt_rcv, myid) & [seq; pack#1],
     outto (ack_rcv, myid), 1)
```

Continuing in path...
Gypsy Model of Protocol

entering next iteration of loop...

Evaluating next_sequen (outto (ack_rcv, myid))
Continuing in path...

Evaluating next_sequen (infrom (pkt_rcv, myid))
Continuing in path...

Evaluating proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
outto (ack_rcv, myid), 0)
Continuing in path...
Assert proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
outto (ack_rcv, myid), 0)
& exp = next_sequen (infrom (pkt_rcv, myid))
& exp = next_sequen (outto (ack_rcv, myid))

-----------------------------
Must verify Assert condition
-----------------------------
Verification condition receivers
H1: infrom (pkt_rcv, myid) & [seq: pack1] = infrom (pkt_rcv#2, myid)
H2: outto (ack_rcv, myid) & [seq: pack1] = outto (ack_rcv#4, myid)
H3: next_sequen (infrom (pkt_rcv, myid)) = exp
H4: next_sequen (outto (ack_rcv, myid)) = exp
H5: infrom (ack_rcv, myid) = infrom (ack_rcv#4, myid)
H6: outto (pkt_rcv, myid) = outto (pkt_rcv#2, myid)
H7: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
outto (ack_rcv, myid), 0)
H8: pack1.segno ne exp

==>
C1: next_sequen (infrom (pkt_rcv, myid) & [seq: pack1]) = exp
C2: next_sequen (outto (ack_rcv, myid) & [seq: pack1]) = exp
C3: proper_reception (outto (sink, myid),
infrom (pkt_rcv, myid) & [seq: pack1],
outto (ack_rcv, myid) & [seq: pack1], 0)

End of path

Exec-> vcs ab_protocol

Generating VCs for procedure ab_protocol

Found 1st path
Found 2nd path

-----------------------------
Beginning new path...
Gypsy Model of Protocol

Assume (unit entry specification)

true

initializing local variables

entering cobegin...

timer (clock_in, clock_out)
receiver (sink, pkt_rcv, ack_rcv)
medium (ack_rcv, ack_send)
medium (pkt_send, pkt_rcv)
sender (source, pkt_send, ack_send, clock_in, clock_out)
Continuing in path...
assume cobegin blocks.

Assume (blockage specification for subprocess sender#1)

if isblocked (sender#1)
then proper_transmission (infrom (source#1, sender#1),
outto (pkt_send#1, sender#1),
infrom (ack_send#1, sender#1), 1)
else false
fi

Assume (blockage specification for subprocess medium#1)

if isblocked (medium#1)
then outto (pkt_rcv#1, medium#1) sub infrom (pkt_send#1, medium#1)
else false
fi

Assume (blockage specification for subprocess medium#2)

if isblocked (medium#2)
then outto (ack_send#1, medium#2) sub infrom (ack_rcv#1, medium#2)
else false
fi

Assume (blockage specification for subprocess receiver#1)

if isblocked (receiver#1)
then proper_reception (outto (sink#1, receiver#1),
infrom (pkt_rcv#1, receiver#1),
outto (ack_rcv#1, receiver#1), 1)
Gypsy Model of Protocol

if false
fi
Assume (blockage specification for subprocess timer1)

if isblocked (timer1) then true else false fi
Assume (blockage specification for unit an-protocol)

isblocked (sender1) or isblocked (medium1) or isblocked (medium2)
or isblocked (receiver1) or isblocked (timer1)

Blockage Assertion is
msg-lag (outto (sink, myid), infrom (source, myid), 1)

Must verify (process blockage) condition
Verification condition an-protocol#2
H1: proper-reception (outto (sink1, receiver1), allfrom (pkt_rcv1),
    allto (ack_rcv1), 1)
H2: proper-transmission (infrom (source1, sender1), allto (pkt_send1),
    allfrom (ack_send1), 1)
H3: isblocked (medium1)
H4: isblocked (medium2)
H5: isblocked (receiver1)
H6: isblocked (sender1)
H7: isblocked (timer1)
H8: allto (ack_send1) sub allfrom (ack_rcv1)
H9: allto (pkt_rcv1) sub allfrom (pkt_send1)

=>
C1: msg-lag (outto (sink1, receiver1), infrom (source1, sender1), 1)

End of path

Beginning new path...

Assume (unit entry specification)

true

initializing local variables

entering cobegin...

timer (clock_in, clock_out)
receiver (sink, pkt_rcv, ack_rcv)
Gypsy Model of Protocol

medium (ack_rcv, ack_send)
medium (pkt_send, pkt_rcv)
sender (source, pkt_send, ack_send, clock_in, clock_out)
Continuing in path...

Assume Procedure Exit Specifications

false & false & false & false & false & false

assume all procedure activations terminate normally.

Leaving Unit ao_protocol
Assert false
End of path

--------------------------------------------------------

Exec-> vcs comp

Generating VCs for function comp

Found 1st path

--------------------------------------------------------

Beginning new path...

Assume (unit entry specification)

true

result := if b = zero then one else zero fi
Leaving Unit comp
Assert result = if b = zero then one else zero fi
End of path

--------------------------------------------------------

Exec-> show status all

The current design and verification status is:

SCOPE ALT_BIT_PROTOCOL

Waiting for pending body to be filled in: "MEDIUM, TIMER"
Proved: COMP
Gypsy Model of Protocol

Types, constants: BIT, BIT_SEQ, CLK_BUF, MESSAGE, MSG_BUF, MSG_SEQ, PACKET, PKT_BUF, PKT_SEQ
Check VCs (not fully checked): AR_PROTOCOL, RECEIVER, SENDER
Check VCs (path pending): AR_PROTOCOL, RECEIVER, SENDER

ap_protocol
waiting to be proved: AR_PROTOCOL#2
proved in vc generator: AR_PROTOCOL#1, AR_PROTOCOL#3, AR_PROTOCOL#4
receiver
waiting to be proved: RECEIVER#4, RECEIVER#5, RECEIVER#6, RECEIVER#8, RECEIVER#12, RECEIVER#13, RECEIVER#14, RECEIVER#18
proved in vc generator: RECEIVER#1, RECEIVER#2, RECEIVER#3, RECEIVER#7, RECEIVER#9, RECEIVER#10, RECEIVER#11, RECEIVER#15, RECEIVER#16, RECEIVER#17
sender
waiting to be proved: SENDER#4, SENDER#5, SENDER#6, SENDER#7, SENDER#13, SENDER#29, SENDER#32
proved in vc generator: SENDER#1, SENDER#2, SENDER#3, SENDER#8, SENDER#9, SENDER#10, SENDER#11, SENDER#12, SENDER#14, SENDER#15, SENDER#16, SENDER#17, SENDER#18, SENDER#19, SENDER#21, SENDER#23, SENDER#24, SENDER#25, SENDER#27, SENDER#29, SENDER#30, SENDER#31, SENDER#32

SCOPE ALT-BIT-SPECs

Waiting for pending body to be filled in: INITIAL-SUBSEQ
For specifications only: LAST-BIT, MSG-LAG, NCHANGES, NEXT-SEQNUM, PROPER-RECEPTION, PROPER-TRANSMISSION, REPEATS, SEQUENCES, UNIQUE-MSG

SCOPE LEMMAS


Exec-> save app.dmp

File APP.DMP already exists. Rewrite it? -> y

Saving...
Gypsy Model of Protocol

*****

[PHOTO: Logging disabled Tue 5-May-81 9:41AM]

[PHOTO: Recording terminated Tue 5-May-81 9:43AM]
Proofs of the VCs in Gypsy

[PHOTO: Recording initiated Thu 7-May-81 7:52AM]

LINK FROM CMP, DIVINO, TTY 20

TOPS-20 Command processor 4(SHO)

[PHOTO: Logging disabled Thu 7-May-81 7:52AM]

[PHOTO: Logging enabled Thu 7-May-81 7:55AM]

% we are now ready to do the proofs. We will first knock off the VCs
% and then take on the various lemmas. These will go in roughly a
% top-down order. The lemmas which get proved in the Affirm system
% will simply be assumed here.

show status all

The current design and verification status is:

SCOPE ALT-BIT-PROTOCOL

waiting for pending body to be filled in: MEDIUM, TIMER

Proved: COMP

Types, constants: BIT, BIT-SEQ, CLK-BUF, MESSAGE, MSG-BUF, MSG-SEQ, PACKET,

PKT-BUF, PKT-SEQ

Check VCs (not fully checked): AB-PROTOCOL, RECEIVER, SENDER

Check VCs (path pending): AB-PROTOCOL, RECEIVER, SENDER

ab_protocol

• waiting to be proved: AB-PROTOCOL#2

Proved in VC generator: AB-PROTOCOL#1, AB-PROTOCOL#3, AB-PROTOCOL#4

receiver

• waiting to be proved: RECEIVER#4, RECEIVER#5, RECEIVER#6, RECEIVER#8,

RECEIVER#12, RECEIVER#13, RECEIVER#14, RECEIVER#18

Proved in VC generator: RECEIVER#1, RECEIVER#2, RECEIVER#3, RECEIVER#7,

RECEIVER#9, RECEIVER#10, RECEIVER#11, RECEIVER#15, RECEIVER#16,

RECEIVER#17

sender

• waiting to be proved: SENDER#4, SENDER#5, SENDER#6, SENDER#7, SENDER#13,

SENDER#20, SENDER#22, SENDER#28, SENDER#33

Proved in VC generator: SENDER#1, SENDER#2, SENDER#3, SENDER#8,

SENDER#9, SENDER#10, SENDER#11, SENDER#12, SENDER#14, SENDER#15,

SENDER#16, SENDER#17, SENDER#18, SENDER#19, SENDER#21, SENDER#23,

SENDER#24, SENDER#25, SENDER#26, SENDER#27, SENDER#29, SENDER#30,

SENDER#31, SENDER#32
Proofs of the VCs in Gopsy

SCOPE ALT_BIT_SPECS

Waiting for pending body to be filled in: INITIAL_SUBSEQ
For specifications only: LAST_BIT, MSG_LAG, NCHANGES, NEXT_SEQNUM, PROPER_RECEPTION, PROPER_TRANSMISSION, REPEATS, SEQNUMS, UNIQUE_MSG

SCOPE LEMMAS

Waiting to be proved: ABP_1, APP_BIT_NONNULL, APP_MSG_NONNULL, APP_PKT_NONNULL, BIT_CASES, COMP_EQ, EQ_ISS, EQ_ISS_APP, HTST_SUB, ISS_APP, ISS_TRANS, INTERPOLATE, LAST_NEXT, LAST_UNIQUE, LAST_REPEATS, MAIN_LEMMA, MSG_LAG_EQ, NE_NEXT, NEXT_COMP, NCHANGES_UNIQUE, PROP_REC_1, PROP_REC_2, PROP_REC_3, PROP_REC_4, PROP_REC_5, PROP_TRANS_1, PROP_TRANS_2, PROP_TRANS_3, PROP_TRANS_4, PROP_TRANS_5, PROP_TRANS_6, PROP_TRANS_7, SUB_APP, SIZE_NULL, SUB_SEQNUM, SUB_TO_LAG, SUB_NCHANGES

Exec-> prove sender#4
Entering Prover with verification condition sender#4

C1: next_seqnum (null (#sectype#)) = one
C2: proper_transmission (null (#sectype#), null (#sectype#), null (#sectype#), 0)

Backup point
(.,)

Prv-> $Proceeding
Backup point
(., 1.,)

Prv-> expand next_seqnum
Backup point
(., 1., E.,)

Prv-> theorem

C1: comp (last_bit (seqnums (null (#sectype#)))) = one

Prv-> expand last_bit
Backup point
(., 1., E., E.,)

Prv-> expand seqnums
Backup point
(., 1., E., E., E.,)

Prv-> theorem

C1: comp (if null (bit_seq)) = null (bit_seq)
then zero
Proofs of the VCs in Gypsy

```plaintext
else null (bit_seq)[size (null (bit_seq))] fi
= one

Prvr→ simplify theorem

Prvr→ expand comp
Backup point
(• 1, E, E, E, E, E,)

Prvr→ $proceeding
(• 1,)

next_seqnum (null (#seqltype$)) = one

Proved

Prvr→ $proceeding
Backup point
(• 2,)

Prvr→ theorem

C1: proper_transmission (null (#seqltype$), null (#seqltype$),
null (#seqltype$), 0)

Prvr→ expand proper_transmission
Backup point
(• 2, E, E,)

Prvr→ theorem

C1: 0 = size (unique_msg (null (#seqltype$)))
C2: msg_seq (unique_msg (null (#seqltype$)), null (#seqltype$), 0)
C3: repeats (null (#seqltype$))

Prvr→ expand unique_msg
Backup point
(• 2, E, E, E,)

Prvr→ expand repeats
Backup point
(• 2, E, E, E, F,)

Prvr→ simplify theorem

Prvr→ theorem

C1: msg_seq (null (msg_seq), null (#seqltype$), 0)

Prvr→ expand msg_seq
Backup point
(• 2, E, E, E, E, F,)
```
Proves of the VC's in Gypsy

Prvr→ theorem

C1: initial_subseq (null (msg_seq), null (seqtype))

Prvr→ use eq_isss

**** eq_isss is not known in scope ALT-SIT-PROTOCOL
Illegal argument to USE

Prvr→ use eq_isss::lemmas
Backup point
(* 2 * E E E E E U *)

Prvr→ qed
(* 2 * E E E E E E U QED *)
(* 2 * E E E E E E U QED AC *)
(* 2 * E E E E E E U QED BC *)

null (msg_seq) = null (seqtype)
Proved
QED

Prvr→ sProceeding
(* 2 *)

proper_transmission (null (seqtype), null (seqtype), null (seqtype), 0)
Proved

Prvr→ sProceeding
sender#4
proved in theorem prover.

Exec→ prove sender#5
Entering Prover with verification condition sender#5

H1: empty (source#1)
H2: next_seqnum (infrom (ack_send, myid)) = next
H3: next_seqnum (outto (pkt_send, myid)) = next
H4: infrom (source, myid) = infrom (source#1, myid)
H5: outto (source, myid) = outto (source#1, myid)
H6: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 0)

→

C1: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 1)

Backup point
(* )

Prvr→ retain h6
Backup point
Proofs of the VCs in Gypsy

(, D ,)

Prvr=> use prop_trans::lemmas
Backup point
(, D , U ,)

Prvr=> qed
(, D , U , QED)
::(, D , U , QED BC)
(, D , U , QED BC)

proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 0)
Proved
QED

Prvr=> sProceeding
sender#5
proved in theorem prover.

Exec=> prove sender#6
Entering Prover with verification condition sender#6

H1: infrom (source, myid) \& (seq: pack1, mssgl) = infrom (source#2, myid)
H2: next_seqnum (infrom (ack_send, myid)) = next
H3: next_seqnum (outto (pkt_send, myid)) = next
H4: infrom (pkt_send, myid) = infrom (pkt_send#1, myid)
H5: outto (pkt_send, myid) = outto (pkt_send#1, myid)
H6: outto (source, myid) = outto (source#2, myid)
H7: full (pkt_send#1)
H8: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 0)
 =>
C1: proper_transmission (infrom (source, myid) \& (seq: pack1, mssgl),
outto (pkt_send, myid), infrom (ack_send, myid), 1)

Backup point
(,)

Prvr=> retain H8
Backup point
(, D ,)

Prvr=> use prop_trans::lemmas
Backup point
(, D , U ,)

Prvr=> qed
(, D , U , QED)
::(, D , U , QED BC)
(, D , U , QED BC)

proper_transmission (infrom (source, myid), outto (pkt_send, myid),
Proofs of the VCs in Gypsy

\text{infrom}(\text{ack\_send}, \text{myid}), 0)

Proved
QED

\text{Prvr} \rightarrow \text{sProceeding}
\text{sender\#6}
proved in theorem prover.

\text{Exec} \rightarrow \text{prove sender\#7}
\text{Entering Prover with verification condition sender\#7}

H1: infrom(source, myid) \& [seq: pack#1, mssg] = infrom(source#2, myid)
H2: outto(pkt_send, myid)
    \& [seq: pack with (.mssg=pack#1.mssg; .seqno=next)]
    = outto(pkt_send#2, myid)
H3: next_seqnum(infrom(ack_send, myid)) = next
H4: next_seqnum(outto(pkt_send, myid)) = next
H5: infrom(pkt_send, myid) = infrom(pkt_send#2, myid)
H6: infrom(start, myid) = infrom(start#1, myid)
H7: outto(source, myid) = outto(source#2, myid)
H8: outto(start, myid) = outto(start#1, myid)
H9: full(start#1)
H10: proper_transmission(infrom(source, myid), outto(pkt_send, myid),
    infrom(ack_send, myid), 0)

\Rightarrow

C1: proper_transmission(infrom(source, myid) \& [seq: pack#1, mssg],
    outto(pkt_send, myid)
    \& [seq: pack with (.mssg=pack#1.mssg;
    .seqno=next)],
    infrom(ack_send, myid), 1)

backup point
(.)

Prvr \rightarrow \text{retain n4, n10}
backup point
(., D.,)

Prvr \rightarrow \text{use prop\_trans\_2::lemmas}
backup point
(., D., !.,)

Prvr \rightarrow \text{qed}
(., D., U., QED)
:(:(., D., U., QED BC)
(., D., U., QED BC 1)
(., D., U., QED BC 1)

pack with (.mssg=pack#1.mssg; .seqno=next), mssg = pack#1, mssg

proved
(., D., U., QED BC 2)
(., D., U., QED BC 2)
Proofs of the VCs in Gopsy

pack with (\textit{mssq} := \textit{pack1}, \textit{mssq}; \textit{seqno} := \textit{next}), \textit{seqno} = \textit{next-seqnum} (outto (\textit{pkt-send}, \textit{myid}))
Proved

(. \textit{D} . \textit{U} . \textit{QED} RC 3)
(. \textit{D} . \textit{U} . \textit{QED} RC 3)

\textit{proper-transmission} (\textit{infrom} (\textit{source}, \textit{myid}), outto (\textit{pkt-send}, \textit{myid}),
infrom (\textit{ack-send}, \textit{myid}), 0)
Proved

(. \textit{D} . \textit{U} . \textit{QED} RC)

pack with (\textit{mssq} := \textit{pack1}, \textit{mssq}; \textit{seqno} := \textit{next}), \textit{mssq} = \textit{pack1}, \textit{mssq}
& pack with (\textit{mssq} := \textit{pack1}, \textit{mssq}; \textit{seqno} := \textit{next}), \textit{seqno}
= \textit{next-seqnum} (outto (\textit{pkt-send}, \textit{myid}))
& \textit{proper-transmission} (\textit{infrom} (\textit{source}, \textit{myid}), outto (\textit{pkt-send}, \textit{myid}),
infrom (\textit{ack-send}, \textit{myid}), 0)
Proved

QED

Prvr-> sProceeding
sender#7
proved in theorem prover.

Exec-> save

Enter file name-> abp.dmp

File ABP.DMP already exists. Rewrite it? -> y

Saving.................................................................
..............................................................................
..............................................................................
..............................................................................
..............................................................................
..............................................................................
Exec-> show status sender

Check VCs (not fully checked): SENDER
Check VCs (path sending): SENDER

sender

*alting to be proved: SENDER#13, SENDER#20, SENDER#22, SENDER#28, SENDER#33

Proved in VC generator: SENDER#1, SENDER#2, SENDER#3, SENDER#8, SENDER#9, SENDER#10, SENDER#11, SENDER#12, SENDER#14, SENDER#15, SENDER#16, SENDER#17, SENDER#18, SENDER#19, SENDER#21, SENDER#23, SENDER#24, SENDER#25, SENDER#26, SENDER#27, SENDER#29, SENDER#30, SENDER#31, SENDER#32

Proved in theorem prover: SENDER#4, SENDER#5, SENDER#6, SENDER#7

Exec-> prove sender#13
Proofs of the VCs in Gypsy

Entering Prover with verification condition sender

H1: infrom (source, myid) @ [seq: pack1, mssg] = infrom (source2, myid)
H2: outto (pkt_send, myid)
   @ [seq: pack with (.mssg:=pack1, mssg; .seqno:=next)]
   = outto (pkt_send2, myid)
H3: next_seqnum (infrom (ack_send, myid)) = next
H4: next_seqnum (outto (pkt_send, myid)) = next
H5: infrom (pkt_send, myid) = infrom (pkt_send2, myid)
H6: outto (source, myid) = outto (source2, myid)
H7: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
   infrom (ack_send, myid), 0)

=>

C1: infrom (source, myid) @ [seq: pack1, mssg]
   = unique_mssg ( outto (pkt_send, myid)
   @ [seq: pack with (.mssg:=pack1, mssg; .seqno:=next)])
C2: size (unique_mssg ( outto (pkt_send, myid)
   @ [seq: pack with (.mssg:=pack1, mssg; .seqno:=next)])
   = size (unique_mssg (infrom (ack_send, myid))) + 1
C3: proper_transmission (infrom (source, myid) @ [seq: pack1, mssg],
   outto (pkt_send, myid)
   @ [seq: pack with (.mssg:=pack1, mssg; .seqno:=next)],
   infrom (ack_send, myid), 1)

Backup point
(*)

Prvr=> retain h4, h7
Backup point
(* D *)

Prvr=> sProceeding
Backup point
(* D 1 *)

Prvr=> theorem

H1: next_seqnum (outto (pkt_send, myid)) = next
H2: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
   infrom (ack_send, myid), 0)

=>

C1: infrom (source, myid) @ [seq: pack1, mssg]
   = unique_mssg ( outto (pkt_send, myid)
   @ [seq: pack with (.mssg:=pack1, mssg; .seqno:=next)])

Prvr=> use prop_trans_1::lemmas
Backup point
(* D 1 U *)

Prvr=> qed
(* D 1 UQed)
Proofs of the VCs in Gypsy

:::Conclusion simplified to:

\[
\text{infrom (source, myid) \& \{seq: pack\#1,mssg\}}
\]

\[
= \text{unique_mssg ( outto (pkt_send, myid)}
\]

\[
\text{\& \{seq: pack with (\{mssg:=pack\#1,mssg\}; \{seqno:=next\}\})}. (D \& 1)
\]

U \& QED BC

(D \& 1 \& U \& QED BC 1)

(D \& 1 \& U \& QED BC 1)

pack with (\{mssg:=pack\#1,mssg\}; \{seqno:=next\}), mssg = pack\#1,mssg

Proved

(D \& 1 \& U \& QED BC 2)

(D \& 1 \& U \& QED BC 2)

pack with (\{mssg:=pack\#1,mssg\}; \{seqno:=next\}), seqno

= \text{next-seqnum (outto (pkt_send, myid))}

Proved

(D \& 1 \& U \& QED BC 3)

(D \& 1 \& U \& QED BC 3)

\text{proper-transmission (infrom (source, myid), outto (pkt_send, myid), y#s, 0)}

Proved

(D \& 1 \& U \& QED BC)

pack with (\{mssg:=pack\#1,mssg\}; \{seqno:=next\}), mssg = pack\#1,mssg

& pack with (\{mssg:=pack\#1,mssg\}; \{seqno:=next\}), seqno

= \text{next-seqnum (outto (pkt_send, myid))}

& proper-transmission (infrom (source, myid), outto (pkt_send, myid), y#s, 0)

Proved

QED

Prvr=> sProceeding

(D \& 1 \& 1)

infrom (source, myid) \& \{seq: pack\#1,mssg\}

= \text{unique_mssg ( outto (pkt_send, myid)}

\text{\& \{seq: pack with (\{mssg:=pack\#1,mssg\}; \{seqno:=next\}\})}

Proved

Prvr=> sProceeding

Backup point

(D \& 2 \& *)

Prvr=> theorem

H1: \text{next-seqnum (outto (pkt_send, myid)) = next}

H2: \text{proper-transmission (infrom (source, myid), outto (pkt_send, myid), infrom (ack_send, myid), 0)}

=>

C1: \text{size (unique_mssg ( outto (pkt_send, myid)}

\text{\& \{seq: pack with (\{mssg:=pack\#1,mssg\}; \{seqno:=next\}\})})
Proofs of the VCs in Gypsy

= size (unique_msg (infrom (ack_send, myid))) + 1

Prv-> expand unique_msg
More than one to expand?
Which do you want to expand? show=choices

1: unique_msg (outto (pkt_send, myid)
   * [seq: pack with [.msg:pack=1, msg; .seqno:=next]])
2: unique_msg (infrom (ack_send, myid))

Which do you want to expand? 1
Backup point
(*, D, E *)

Prv-> simplify theorem

Prv-> theorem

H1: next_seqnum (outto (pkt_send, myid)) = next
H2: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 0)

→

C1: size (unique_msg (outto (pkt_send, myid)))
   = size (unique_msg (infrom (ack_send, myid)))

Prv-> expand proper_transmission
Backup point
(*, D, E *)

Prv-> simplify hypothesis

Prv-> theorem

H1: next_seqnum (outto (pkt_send, myid)) = next
H2: size (unique_msg (infrom (ack_send, myid)))
   = size (unique_msg (outto (pkt_send, myid)))
H3: size (unique_msg (infrom (ack_send, myid))) = size (infrom (source,
   myid))
H4: msg_len (unique_msg (outto (pkt_send, myid)), infrom (source, myid), 0)
H5: repeats (outto (pkt_send, myid))

→

C1: size (unique_msg (outto (pkt_send, myid)))
   = size (unique_msg (infrom (ack_send, myid)))

Prv-> equality substitute H2
H2 can be solved for:
   T1: size (unique_msg (infrom (ack_send, myid)))
   T2: size (unique_msg (outto (pkt_send, myid)))
Which term (by label) do you want to substitute for?
T1
Proofs of the VCs in Gypsy

\[
\text{size (unique\_msg (infrom (ack\_send, myid))) = size (unique\_msg (outto (pkt\_send, myid)))}
\]

OK??

\text{Typelist equalities}

\[
\text{size (infrom (source, myid)) = size (unique\_msg (outto (pkt\_send, myid)))}
\]

Backup point
\[
(*\ D\ \ ?\ .\ E\ .\ =S\ .)
\]

Prvr\rightarrow \$\text{Proceeding}
\[
(*\ D\ \ ?\ .)
\]

\[
\text{size (unique\_msg (outto (pkt\_send, myid))}
\]
\[
\text{\quad \&\ \{seq: pack with (\_mssg:=pack\#1\_mssg; \_seqno:=next)\})}
\]
\[
= \text{size (unique\_msg (infrom (ack\_send, myid))) + 1}
\]

Proved

Prvr\rightarrow \$\text{Proceeding}

Backup point
\[
(*\ D\ .\ 3\ .)
\]

Prvr\rightarrow \text{theorem}

\[
\text{H1: next\_seqnum (outto (pkt\_send, myid)) = next}
\]
\[
\text{H2: proper\_transmission (infrom (source, myid), outto (pkt\_send, myid),}
\]
\[
\quad \text{infrom (ack\_send, myid), 0)}
\]

\rightarrow

\[
\text{C1: proper\_transmission (infrom (source, myid) \& \{seq: pack\#1\_mssg},}
\]
\[
\quad \text{outto (pkt\_send, myid)}
\]
\[
\quad \text{\& \{seq: pack with (\_mssg:=pack\#1\_mssg;}
\]
\[
\quad \quad \text{\_seqno:=next)\},}
\]
\[
\quad \text{infrom (ack\_send, myid), 1)}
\]

Prvr\rightarrow \text{use prop\_trans\_2::lemmas}

Backup point
\[
(*\ D\ .\ 3\ .\ U\ .)
\]

Prvr\rightarrow \text{QED}
\[
(*\ D\ .\ 3\ .\ U\ .\ \text{QED})
\]
\[
(*\ D\ .\ 3\ .\ U\ .\ \text{QED AC})
\]
\[
(*\ D\ .\ 3\ .\ U\ .\ \text{QED AC} 1)
\]
\[
(*\ D\ .\ 3\ .\ U\ .\ \text{QED AC} 1)
\]

\[
\text{pack with (\_mssg:=pack\#1\_mssg; \_seqno:=next), mssg = pack\#1\_mssg}
\]

Proved
\[
(*\ D\ .\ 3\ .\ U\ .\ \text{QED AC} 2)
\]
\[
(*\ D\ .\ 3\ .\ U\ .\ \text{QED AC} 2)
\]

\[
\text{pack with (\_mssg:=pack\#1\_mssg; \_seqno:=next), seqno}
\]
\[
\quad = \text{next\_seqnum (outto (pkt\_send, myid))}
\]

Proved
Proofs of the VCs in Gypsy

(* D . 3 . U . QED AC 3 *)
(* D . 3 . U . QED AC 3 *)

proper_transmission (in from (source, myid), out to (pkt_send, myid),
in from (ack_send, myid), 0)

Proved
(* D . 3 . U . QED AC *)

pack with (.mssq := pack#1.mssq; .seqno := next), mssq = pack#1.mssq & pack with (.mssq := pack#1.mssq; .seqno := next), seqno = next-seqnum (out to (pkt_send, myid))

& proper_transmission (in from (source, myid), out to (pkt_send, myid),
in from (ack_send, myid), 0)

Proved

QED

Prv ⇒ sProceeeding
(* D . 3 . *)

proper_transmission (in from (source, myid) & [seq: pack#1.mssq],
out to (pkt_send, myid) & [seq: pack with (.mssq := pack#1.mssq; .seqno := next))],
in from (ack_send, myid), 1)

Proved

Prv ⇒ sProceeeding

sender#13

proved in theorem prover.

Exec ⇒ prove sender#20

Entering Prover with verification condition sender#20

H1: in from (ack_send, myid) & [seq: ack#1] = in from (ack_send#3, myid)
H2: pack, seqno = next
H3: next-seqnum (in from (ack_send, myid)) = next
H4: unique_msa (out to (pkt_send, myid)) = in from (source, myid)
H5: out to (ack_send, myid) = out to (ack_send#3, myid)
H6: out to (pkt_send, myid) [size (out to (pkt_send, myid))] = pack
H7: size (unique_msa (out to (pkt_send, myid))) = size (unique_msa (in from (ack_send, myid))) + 1
H8: proper_transmission (in from (source, myid), out to (pkt_send, myid),
in from (ack_send, myid), 1)
H9: ack#1, seqno ne next
H10: null (out to) ne out to (pkt_send, myid)

⇒

C1: next-seqnum (in from (ack_send, myid) & [seq: ack#1]) = next
C2: size (unique_msa (out to (pkt_send, myid))) = size (unique_msa (in from (ack_send, myid) & [seq: ack#1])) + 1
C3: proper_transmission (in from (source, myid), out to (pkt_send, myid),
in from (ack_send, myid) & [seq: ack#1], 1)

Typelist equalities
Proofs of the VCs in Gynsy

\[
\text{size (unique_msg (infrom (ack_send, myid))) + 1 = size (unique_msg (outto (pkt_send, myid)))}
\]

Backup point
(., D .)

Prv⇒ retain n3,n7,n8,n9
Typelist equalities

\[
\text{size (unique_msg (infrom (ack_send, myid))) + 1 = size (unique_msg (outto (pkt_send, myid)))}
\]

Backup point
(., D .)

Prv⇒ sProceeding
Backup point
(., D . 1 .)

Prv⇒ theorem

H1: \[\text{size (unique_msg (infrom (ack_send, myid))) + 1 = size (unique_msg (outto (pkt_send, myid)))}\]
H2: \[\text{next_seqnum (infrom (ack_send, myid)) = next}\]
H3: \[\text{size (unique_msg (outto (pkt_send, myid))) = size (unique_msg (infrom (ack_send, myid))) + 1}\]
H4: \[\text{proper_transmission (infrom (source, myid), outto (pkt_send, myid), infrom (ack_send, myid), 1)}\]
H5: \[\text{ack#1, seqno ne next}\]

⇒

C1: \[\text{next_seqnum (infrom (ack_send, myid) \& [seq: ack#1]) = next}\]

Prv⇒ retain n2,n5
Backup point
(., D . 1 . D .)

Prv⇒ use ne_next::lemmas
Typelist equalities

\[
\text{size (unique_msg (infrom (ack_send, myid))) + 1 = size (unique_msg (outto (pkt_send, myid)))}
\]

\& \[\text{size (unique_msg (outto (pkt_send, myid))) = size (unique_msg (infrom (ack_send, myid))) + 1}\]

Backup point
(., D . 1 . D . U .)

Prv⇒ hypothesis

H1: \[\text{size (unique_msg (infrom (ack_send, myid))) + 1 = size (unique_msg (outto (pkt_send, myid)))}\]
H2: \[\text{size (unique_msg (outto (pkt_send, myid))) = size (unique_msg (infrom (ack_send, myid))) + 1}\]
H3: \[\text{p#8, seqno ne next_seqnum (x#12s)}\]

⇒ \[\text{next_seqnum (x#12s \& [seq: p#8]) = next_seqnum (x#12s)}\]
Proofs of the VCs in Gypsy

H4: next_sequnum (infrom (ack_send, myid)) = next
H5: ack#1.seqno ne next

Prvr→ reorder h4,n3,h5
Backup point
(* D . 1 . D . U . REORDER .)

Prvr→ qed
(* D . 1 . D . U . REORDER . QED)

next := next_sequnum (infrom (ack_send, myid))
(* D . 1 . D . U . REORDER . QED =S)

size (unique_msg (infrom (ack_send, myid))) :=
  size (unique_msg (outto (pkt_send, myid))) = 1
(* D . 1 . D . U . REORDER . QED =S =S)

ack#1.seqno ne next_sequnum (infrom (ack_send, myid))
proved
QED

Prvr→ sProceeding
(* D . 1 .)

next_sequnum (infrom (ack_send, myid) a [seq: ack#1]) = next
proved

Prvr→ sProceeding
Backup point
(* D . 2 .)

Prvr→ theorem

H1: size (unique_msg (infrom (ack_send, myid))) + 1
  = size (unique_msg (outto (pkt_send, myid)))
H2: next_sequnum (infrom (ack_send, myid)) = next
H3: size (unique_msg (outto (pkt_send, myid)))
  = size (unique_msg (infrom (ack_send, myid))) + 1
H4: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 1)

H5: ack#1.seqno ne next

→
C1: size (unique_msg (outto (pkt_send, myid)))
  = size (unique_msg (infrom (ack_send, myid)) a [seq: ack#1]) + 1

Prvr→ expand unique_msg
More than one to expand.
Which do you want to expand? show-choices
1: unique_msg (infrom (ack_send, myid))
Proofs of the VCs in Gypsy

2: unique_msg (outto (pkt_send, myid))
3: unique_msg (infrom (ack_send, myid) & (seq: ack#1))

which do you want to expand? 3
Backup point
(, D, 2, E, )

Prv-> simplify theorem

Prv-> theorem

H1: next_segunum (infrom (ack_send, myid)) = next
H2: size (unique_msg (infrom (ack_send, myid))) + 1
    = size (unique_msg (outto (pkt_send, myid)))
H3: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1
H4: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
    infrom (ack_send, myid), 1)
H5: ack#1.segno ne next

=>
C1: size (unique_msg (outto (pkt_send, myid)))
    = size (if ack#1.segno = next_segunum (infrom (ack_send, myid))
        then unique_msg (infrom (ack_send, myid))
        else unique_msg (infrom (ack_send, myid))
    + t1) + 1

Prv-> equality substitute h1
H1 yields
next_segunum (infrom (ack_send, myid)) := next
Backup point
(, D, 2, E, =S, )

Prv-> simplify theorem

Prv-> theorem

C1: true

Prv-> sProceeding
(, D, 2, )

size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid) & [seq: ack#1])) + 1
Proved

Prv-> sProceeding
Backup point
(, D, 3, )

Prv-> theorem
Proofs of the VCs in Gypsy

H1: size (unique_msg (infom (ack-send, myid))) + 1
    = size (unique_msg (outto (pkt-send, myid)))
H2: next-seqnum (infom (ack-send, myid)) = next
H3: size (unique_msg (outto (pkt-send, myid)))
    = size (unique_msg (infom (ack-send, myid))) + 1
H4: proper-transmission (infom (source, myid), outto (pkt-send, myid),
    infom (ack-send, myid), 1)
H5: ack#1, seqno ne next

⇒
C1: proper-transmission (infom (source, myid), outto (pkt-send, myid),
    infom (ack-send, myid) & [seq: ack#1], 1)

Prv⇒ use prop_trans-4::lemmas
Backup point
(* D * 3 * U *)

Prv⇒ qed
(* D * 3 * U * QED)
↓↓↓↓(* D * 3 * U * QED BC)
(* D * 3 * U * QED BC 1)
(* D * 3 * U * QED BC 1)

proper-transmission (infom (source, myid), outto (pkt-send, myid),
    infom (ack-send, myid), 1)

Proved
(* D * 3 * U * QED BC 2)
↓↓↓↓
size (unique_msg (infom (ack-send, myid))) :=
    size (unique_msg (outto (pkt-send, myid))) - 1
(* D * 3 * U * QED BC 2 =S)
↑↑↑↑
next-seqnum (infom (ack-send, myid)) := next
(* D * 3 * U * QED BC 2 =S =S)
(* D * 3 * U * QED BC 2)

ack#1, seqno ne next-seqnum (infom (ack-send, myid))

Proved
(* D * 3 * U * QED BC)

proper-transmission (infom (source, myid), outto (pkt-send, myid),
    infom (ack-send, myid), 1)
& ack#1, seqno ne next-seqnum (infom (ack-send, myid))

Proved
QED

Prv⇒ $Proceeding
(* D * 3 *)

proper-transmission (infom (source, myid), outto (pkt-send, myid),
    infom (ack-send, myid) & [seq: ack#1], 1)

Proved
Proofs of the VCs in Gypsy

Provr-> Proceeding
sender#20
proved in theorem prover.

Exec->
save

Enter file name-> abp.dmp

File ABP.DMP already exists. Rewrite it? => y

Saving.................................................................
...........................................................................
...........................................................................
...........................................................................
...........................................................................
...........................................................................
...........................................................................
...........................................................................

Exec-> prove sender#22
Entering Prover with verification condition sender#22

H1: outto (pkt_send, myid) @ [seq: pack] = outto (pkt_send#4, myid)
H2: pack, seqno = next
H3: next_seqnum (infrom (ack_send, myid)) = next
H4: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
H5: infrom (pkt_send, myid) = infrom (pkt_send#4, myid)
H6: infrom (start, myid) = infrom (start#3, myid)
H7: outto (start, myid) = outto (start#3, myid)
H8: outto (pkt_send, myid)(size (outto (pkt_send, myid))) = pack
H9: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1
H10: full (start#3)
H11: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
    infrom (ack_send, myid), 1)
H12: null (pkt_seq) ne outto (pkt_send, myid)

=>
   C1: proper_transmission (infrom (source, myid),
       outto (pkt_send, myid) @ [seq: pack],
       infrom (ack_send, myid), 1)

Typelist equalities

size (unique_msg (infrom (ack_send, myid))) + 1
= size (unique_msg (outto (pkt_send, myid)))

Backup point
(. )

Provr-> retain H8,h11,h12
Backup point
(., . ,)

Provr-> use prop_trans_3::lemmas
Typelist equalities
Proofs of the VCs in Gypsy

\[\text{size (unique_msg (infrom (ack_send, myid))) + 1} = \text{size (unique_msg (outto (pkt_send, myid)))} \]
\& \text{size (unique_msg (outto (pkt_send, myid)))} = \text{size (unique_msg (infrom (ack_send, myid))) + 1}

Backup point

(* D U *)

Prv-> qed

(* D U QED *)

: : : : : : :
\text{size (unique_msg (infrom (ack_send, myid)))} := \text{size (unique_msg (outto (pkt_send, myid)))} = 1

(* D U QED = S *)


(* D U QED = S = S BC 1 *)

(* D U QED = S = S BC 1 *)

\text{outto (pkt_send, myid)[size (outto (pkt_send, myid))]

\text{outto (pkt_send, myid)[size (outto (pkt_send, myid))]

Proved

(* D U QED = S = S BC 2 *)

(* D U QED = S = S BC 2 *)

\text{proper_transmission (infrom (source, myid), outto (pkt_send, myid), infrom (ack_send, myid), 1)}

Proved

(* D U QED = S = S BC 3 *)

(* D U QED = S = S BC 3 *)

\text{null (pkt_seq) ne outto (pkt_send, myid)}

Proved

(* D U QED = S = S BC *)

\text{outto (pkt_send, myid)[size (outto (pkt_send, myid))]

\text{outto (pkt_send, myid)[size (outto (pkt_send, myid))]

\& \text{proper_transmission (infrom (source, myid), outto (pkt_send, myid), infrom (ack_send, myid), 1)}

\& \text{null (pkt_seq) ne outto (pkt_send, myid)}

Proved

QED

Prv-> $Proceeding$

sender#22

proved in theorem prover.

Exec-> prove sender#28

Entering Prover with verification condition sender#28

H1: \text{outto (pkt_send, myid) \& [seq: pack] = outto (pkt_send\#4, myid)}
Proofs of the VCs in Gypsy

H2: pack.send = next
H3: next_seqnum (infrom (ack.send, myid)) = next
H4: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
H5: infrom (pkt_send, myid) = infrom (pkt_send, myid)
H6: outto (pkt_send, myid) = size (unique_msg (infrom (ack.send, myid))) = pack
H7: size (unique_msg (outto (pkt_send, myid))) = size (unique_msg (infrom (ack.send, myid))) + 1
H8: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
    infrom (ack.send, myid), 1)
H9: null (pkt_seq) ne outto (pkt_send, myid)

=>
C1: unique_msg (outto (pkt_send, myid)) infrom (source, myid)
    = size (unique_msg (outto (pkt_send, myid))) = 1
C2: size (unique_msg (outto (pkt_send, myid))) = size (unique_msg (infrom (ack.send, myid))) + 1
C3: proper_transmission (infrom (source, myid),
    outto (pkt_send, myid) = [seq: pack],
    infrom (ack.send, myid), 1)

Typelist equalities

    size (unique_msg (infrom (ack.send, myid))) + 1
    = size (unique_msg (outto (pkt_send, myid)))

Backup point
(.*

Prvr=> retain h4,h6,h7,h8,h9

Typelist equalities

    size (unique_msg (infrom (ack.send, myid))) + 1
    = size (unique_msg (outto (pkt_send, myid)))

Backup point
(.* B *).

Prvr=> sProceeding
Backup point
(.* D, 1 *).

Prvr=> theorem

H1: size (unique_msg (infrom (ack.send, myid))) + 1
    = size (unique_msg (outto (pkt_send, myid)))
H2: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
H3: outto (pkt_send, myid) = size (outto (pkt_send, myid))) = pack
H4: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack.send, myid))) + 1
H5: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
    infrom (ack.send, myid), 1)
H6: null (pkt_seq) ne outto (pkt_send, myid)

=>
C1: unique_msg (outto (pkt_send, myid)) infrom (source, myid)
Proofs of the VCs in Gypsy

Prvr => retain h2, h3, h6
Backup point
( . D . 1 . D )

Prvr => use last_unique::lemmas
Type list equalities

size (unique_msg (infrom (ack_send, myid))) + 1
= size (unique_msg (outto (pkt_send, myid)))
& size (unique_msg (outto (pkt_send, myid)))
= size (unique_msg (infrom (ack_send, myid))) + 1

Backup point

Prvr => hypothesis

H1: size (unique_msg (infrom (ack_send, myid))) + 1
  = size (unique_msg (outto (pkt_send, myid)))
H2: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1
H3: x#18s (size (x#18s)) = n#14s & null (pkt_seq) ne x#18s
    => unique_msg (x#18s a [seq; n#14s]) = unique_msg (x#18s)
H4: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
H5: outto (pkt_send, myid)[size (outto (pkt_send, myid))] = pack
H6: null (pkt_seq) ne outto (pkt_send, myid)

Prvr => delete h1, h2
Backup point

Prvr => qed
:::
    infrom (source, myid) := unique_msg (outto (pkt_send, myid))
:::
    pack := outto (pkt_send, myid)[size (outto (pkt_send, myid))]

    outto (pkt_send, myid)[size (outto (pkt_send, myid))]
    = outto (pkt_send, myid)[size (outto (pkt_send, myid))]
Proved

    null (pkt_seq) ne outto (pkt_send, myid)
Proved
Proofs of the VCs in Gyvus

outto (pkt_send, myid)[size (outto (pkt_send, myid))] = outto (pkt_send, myid)[size (outto (pkt_send, myid))] & null (pkt_seq) ne outto (pkt_send, myid)

Proved
QED

PRV => sProceeding
(* D . 1 .)

unique_msg (outto (pkt_send, myid) @ [seq; pack]) = infrom (source, myid)

Proved

PRV => sProceeding
Backup point
(* D . 2 .)

PRV => theorem

H1: size (unique_msg (infrom (ack_send, myid))) + 1
    = size (unique_msg (outto (pkt_send, myid)))

H2: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)

H3: outto (pkt_send, myid)[size (outto (pkt_send, myid))] = pack

H4: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1

H5: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 1)

H6: null (pkt_seq) ne outto (pkt_send, myid)

=>

C1: size (unique_msg (outto (pkt_send, myid) @ [seq; pack]))
    = size (unique_msg (infrom (ack_send, myid))) + 1

PRV => delete H4, H5

Typelist equalities

size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1

Backup point
(* D . 2 . D .)

PRV => use last_unique::lemmas
Backup point
(* D . 2 . D . H .)

PRV => hypothesis

H1: x#20s[size (x#20s)] = p#16s & null (pkt_seq) ne x#20s
    => unique_msg (x#20s @ [seq: p#16s]) = unique_msg (x#20s)

H2: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1

H3: size (unique_msg (infrom (ack_send, myid))) + 1
    = size (unique_msg (outto (pkt_send, myid)))

H4: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
Proofs of the VCs in Gvrsy

H5: \text{outto (pkt\_send, myid)[size (outto (pkt\_send, myid))] = pack}
H6: null (pkt\_send) ne outto (pkt\_send, myid)

Prvr \rightarrow \text{forwardchain n1}

\begin{align*}
\text{unique\_msg (outto (pkt\_send, myid) a [seq: pack])} \\
= \text{unique\_msg (outto (pkt\_send, myid))}
\end{align*}

Backup point
(*) D . 2 . D . U . FC .)

Prvr \rightarrow \text{theorem}

H1: \text{unique\_msg (outto (pkt\_send, myid) a [seq: pack])} \\
= \text{unique\_msg (outto (pkt\_send, myid))}

H2: \text{size (size (x\_20s)) = x\_16s & null (pkt\_seq) ne x\_20s} \\
\rightarrow \text{unique\_msg (x\_20s a [seq: x\_16s]) = unique\_msg (x\_20s)}

H3: \text{size (unique\_msg (outto (pkt\_send, myid)))} \\
= \text{size (unique\_msg (infrom (ack\_send, myid)))} + 1

H4: \text{size (unique\_msg (infrom (ack\_send, myid)))} + 1 \\
= \text{size (unique\_msg (outto (pkt\_send, myid)))}

H5: \text{unique\_msg (outto (pkt\_send, myid)) = infrom (source, myid)}

H6: \text{outto (pkt\_send, myid)[size (outto (pkt\_send, myid))] = pack}

H7: null (pkt\_seq) ne outto (pkt\_send, myid)

\rightarrow

C1: \text{size (unique\_msg (outto (pkt\_send, myid) a [seq: pack]))} \\
= \text{size (unique\_msg (infrom (ack\_send, myid)))} + 1

Prvr \rightarrow \text{equality substitute n1}
H1 yields

\text{unique\_msg (outto (pkt\_send, myid) a [seq: pack]) :=} \\
\text{unique\_msg (outto (pkt\_send, myid))}

Backup point

Prvr \rightarrow \text{sProceeding}

(*) D . 2 .

\text{size (unique\_msg (outto (pkt\_send, myid) a [seq: pack]))} \\
= \text{size (unique\_msg (infrom (ack\_send, myid)))} + 1

Proved

Prvr \rightarrow \text{sProceeding}

Backup point

(*) D . 3 .

Prvr \rightarrow \text{theorem}

H1: \text{size (unique\_msg (infrom (ack\_send, myid)))} + 1 \\
= \text{size (unique\_msg (outto (pkt\_send, myid)))}

H2: \text{unique\_msg (outto (pkt\_send, myid)) = infrom (source, myid)}

H3: \text{outto (pkt\_send, myid)[size (outto (pkt\_send, myid))] = pack}
Proofs of the VCs in GyRSV

H4: size (unique_msg (outto (pkt_send, myid)))
   = size (unique_msg (infrom (ack_send, myid))) + 1

H5: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
   infrom (ack_send, myid), 1)

H6: null (pkt_seq) ne outto (pkt_send, myid).

\rightarrow

C1: proper_transmission (infrom (source, myid),
   outto (pkt_send, myid) & (seq= pack),
   infrom (ack_send, myid), 1)

Prvt-> use prop_trans_3::lemmas

Backup point

( . D . 3 . U . )

Prvt-> qed

( . D . 3 . U . QED )

( . D . 3 . U . QED RC 1 )

( . D . 3 . U . QED RC 1 )

outto (pkt_send, myid)[size (outto (pkt_send, myid))] = pack

Proved

( . D . 3 . U . QED RC 2 )

( . D . 3 . U . QED RC 2 )

proper_transmission (infrom (source, myid), outto (pkt_send, myid),
   infrom (ack_send, myid), 1)

Proved

( . D . 3 . U . QED RC 3 )

( . D . 3 . U . QED RC 3 )

null (pkt_seq) ne outto (pkt_send, myid)

Proved

( . D . 3 . U . QED RC )

outto (pkt_send, myid)[size (outto (pkt_send, myid))] = pack

& proper_transmission (infrom (source, myid), outto (pkt_send, myid),
   infrom (ack_send, myid), 1)

& null (pkt_seq) ne outto (pkt_send, myid)

Proved

QED

Prvt-> $Proceeding$

( . D . 3 . )

proper_transmission (infrom (source, myid),
   outto (pkt_send, myid) & (seq= pack),
   infrom (ack_send, myid), 1)

Proved

Prvt-> $Proceeding$

sender=28
Proofs of the VCs in Gypsy

proved in the theorem prover.

[PHOTO: Recording initiated Sat 9-May-81 9:55AM]

LINK FROM CP, DIVITO, TTY 20

TOPS-20 Command processor 4(560)

[PHOTO: Logging disabled Sat 9-May-81 9:55AM]

@[PHOTO: Logging enabled Sat 9-May-81 10:04AM]

prove sender#33

Entering Prover with verification condition sender#33

H1: infrom (ack_send, myid) @ [seq: ack#1] = infrom (ack_send#3, myid)
H2: ack#1.seqno = next
H3: pack.seqno = next
H4: next_seqnum (infrom (ack_send, myid)) = next
H5: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
H6: outto (ack_send, myid) = outto (ack_send#3, myid)
H7: outto (pkt_send, myid)[size (outto (pkt_send, myid))] = pack

= size (unique_msg (infrom (ack_send, myid))) + 1
H9: proper_transmission (infrom (source, myid), outto (pkt_send, myid),

infrom (ack_send, myid), 1)

H10: null (pkt_seq) ne outto (pkt_send, myid)

=>

C1: comp (next)

= next_seqnum (infrom (ack_send, myid) @ [seq: ack#1])
C2: comp (next) = next_seqnum (outto (pkt_send, myid))
C3: proper_transmission (infrom (source, myid), outto (pkt_send, myid),

infrom (ack_send, myid) @ [seq: ack#1], 0)

Typelist equalities

= size (unique_msg (infrom (ack_send, myid))) + 1

Back up point
(.*

Prv-> delete h1,h6
Back up point
(.* D .)

Prv-> $Proceeding
Back up point
(.* D .1 .)

Prv-> theorem
Proofs of the VCs in Gypsy

H1: size (unique_msg (infrom (ack_send, myid))) + 1
    = size (unique_msg (outto (pkt_send, myid)))
H2: ack#1.segno = next
H3: pkt.segno = next
H4: next_seqnum (infrom (ack_send, myid)) = next
H5: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
H6: outto (pkt_send, myid)[size (outto (pkt_send, myid))] = pack
H7: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1
H8: proper_transmission (infrom (source, myid), outto (pkt_send, myid),
  infrom (ack_send, myid), 1)
H9: null (pkt_seq) ne outto (pkt_send, myid)

--> C1: comp (next)
    = next_seqnum (infrom (ack_send, myid) @ [seq: ack#1])

Prv⇒ retain n2
Backup point
(. D . 1 . D .)

Prv⇒ use next-comp:lemmas
Typelist equalities

  size (unique_msg (outto (pkt_send, myid)))
  = size (unique_msg (infrom (ack_send, myid))) + 1
& size (unique_msg (infrom (ack_send, myid))) + 1
  = size (unique_msg (outto (pkt_send, myid)))

Backup point
(. D . 1 . D . D .)

Prv⇒ theorem

H1: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1
H2: size (unique_msg (infrom (ack_send, myid))) + 1
    = size (unique_msg (outto (pkt_send, myid)))
H3: comp (p#208.segno) = next_seqnum (x#248 & [seq: p#208])
H4: ack#1.segno = next

--> C1: comp (next)
    = next_seqnum (infrom (ack_send, myid) @ [seq: ack#1])

Prv⇒ delete h1,n2
Backup point
(. D . 1 . D . U . D .)

Prv⇒ qed
(. D . 1 . D . U . D . QED)
:::
next := ack#1.segno
(. D . 1 . D . U . D . QED =S)
QED
Proofs of the VCs in Gynsy

PrvR-> $\text{Proceeding}$
( * D . 1 . )

comp (next) = next_seqnum (infrom (ack_send, myid) & [seq: ack#1])
proved

PrvR-> $\text{Proceeding}$
Backup point
( * D . 2 . )

PrvR-> $\text{Theorem}$

H1: size (unique_msg (infrom (ack_send, myid))) + 1
    = size (unique_msg (outto (pkt_send, myid)))
H2: ack#1_seqno = next
H3: pack_seqno = next
H4: next_seqnum (infrom (ack_send, myid)) = next
H5: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
H6: outto (pkt_send, myid)size (outto (pkt_send, myid))) = pack
H7: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1
H8: properly_transmission (infrom (source, myid), outto (pkt_send, myid),
    infrom (ack_send, myid), 1)
H9: null (pkt_seq) ne outto (pkt_send, myid)

$\Rightarrow$
C1: comp (next) = next_seqnum (outto (pkt_send, myid))

PrvR-> $\text{Retain h3, h6, h9}$
Backup point
( * D . 2 . D . )

PrvR-> $\text{Use last-next::lemmas}$
Typelist equalities

size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1
& size (unique_msg (infrom (ack_send, myid))) + 1
    = size (unique_msg (outto (pkt_send, myid)))

Backup point
( * D . 2 . D . U . )

PrvR-> $\text{Theorem}$

H1: size (unique_msg (outto (pkt_send, myid)))
    = size (unique_msg (infrom (ack_send, myid))) + 1
H2: size (unique_msg (infrom (ack_send, myid))) + 1
    = size (unique_msg (outto (pkt_send, myid)))
H3: x#26s[size (x#26s)] = p#22s & null (pkt_seq) ne x#26s
    $\Rightarrow$ p#22s_seqno ne next_seqnum (x#26s)
H4: pack_seqno = next
H5: outto (pkt_send, myid)[size (outto (pkt_send, myid))] = pack
Proofs of the VCs in Gvnsv

H6: null (pkt_seq) ne outto (pkt_send, myid)
=>
C1: comp (next) = next_seenum (outto (pkt_send, myid))

Prvr=> forwardchain h3
::::::Forward chaining gives
pack.seanno ne next_seenum (outto (pkt_send, myid))
Backup point
(* D . 2 . D . U . FC . D *)

Prvr=> hypothesis

H1: pack.seanno ne next_seenum (outto (pkt_send, myid))
H2: size (unique_msg (outto (pkt_send, myid)))
   = size (unique_msg (infrom (ack_send, myid))) + 1
H3: size (unique_msg (infrom (ack_send, myid))) + 1
   = size (unique_msg (outto (pkt_send, myid)))
H4: x#26s[size (x#26s)] = p#22s & null (pkt_seq) ne x#26s
   => p#22s.seanno ne next_seenum (x#26s)
H5: pack.seanno = next
H6: outto (pkt_send, myid)[size (outto (pkt_send, myid))] = pack
H7: null (pkt_seq) ne outto (pkt_send, myid)

Prvr=> retain h1,n5
Backup point
(* D . 2 . D . U . FC . D *)

Prvr=> hypothesis

H1: pack.seanno ne next_seenum (outto (pkt_send, myid))
H2: pack.seanno = next

Prvr=> equality substitute h2
H2 yields
next := pack.seanno
Backup point

Prvr=> theorem

H1: pack.seanno ne next_seenum (outto (pkt_send, myid))
=>
C1: comp (pack.seanno) = next_seenum (outto (pkt_send, myid))

Prvr=> use comp_ne::lemmas
Typelist equalities

size (unique_msg (outto (pkt_send, myid)))
   = size (unique_msg (infrom (ack_send, myid))) + 1
& size (unique_msg (infrom (ack_send, myid))) + 1
   = size (unique_msg (outto (pkt_send, myid)))
Proofs of the VCs in Gypsy

Backup point

Proof-> theorem

H1: size (unique_msg (outto (pkt_send, myid))) = size (unique_msg (infrom (ack_send, myid))) + 1
H2: size (unique_msg (infrom (ack_send, myid))) + 1 = size (unique_msg (outto (pkt_send, myid)))
H3: comp (b1 #2s) = b2 #2s iff b1 #2s ne b2 #2s
H4: pack_seqno ne next_seqnum (outto (pkt_send, myid))

->
C1: comp (pack_seqno) = next_seqnum (outto (pkt_send, myid))

Proof-> forward chaining
which way? (left or right)
left
:::Forward chaining gives

comp (pack_seqno) = next_seqnum (outto (pkt_send, myid))

Backup point

Proof-> sProceeding
( . D . 2 . )

comp (next) = next_seqnum (outto (pkt_send, myid))

Proved

Proof-> sProceeding
Backup point
( . D . 3 . )

Proof-> theorem

H1: size (unique_msg (infrom (ack_send, myid))) + 1 = size (unique_msg (outto (pkt_send, myid)))
H2: ack1_seqno = next
H3: pack_seqno = next
H4: next_seqnum (infrom (ack_send, myid)) = next
H5: unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
H6: outto (pkt_send, myid)[size (outto (pkt_send, myid))] = pack
H7: size (unique_msg (outto (pkt_send, myid))) = size (unique_msg (infrom (ack_send, myid))) + 1
H8: proper_transmission (infrom (source, myid), outto (pkt_send, myid), infrom (ack_send, myid), 1)
H9: null (pkt_seq) ne outto (pkt_send, myid)

->
C1: proper_transmission (infrom (source, myid), outto (pkt_send, myid), infrom (ack_send, myid) # [seq: ack1_seqno], 0)

Proof-> use prop_trans_7::lemmas
Prove point

\texttt{Prv \rightarrow qed}

\begin{verbatim}
(* D . 3 . U . QED *)

size (unique_msg (infrom (ack_send, myid))) :=
size (unique_msg (outto (pkt_send, myid))) - 1
(* D . 3 . U . QED BC 1 =S *)

ack\#1.seqno := next
(* D . 3 . U . QED BC 1 =S =S *)
(* D . 3 . U . QED BC 1 *)

ack\#1.seqno = next sequence (infrom (ack_send, myid))

Proved
(* D . 3 . U . QED BC 2 *)
(* D . 3 . U . QED BC 2 *)

unique_msg (outto (pkt_send, myid)) = infrom (source, myid)

Proved
(* D . 3 . U . QED BC 3 *)
(* D . 3 . U . QED BC 3 *)

size (unique_msg (outto (pkt_send, myid)))
= size (unique_msg (infrom (ack_send, myid))) + 1

Proved
(* D . 3 . U . QED BC 4 *)
(* D . 3 . U . QED BC 4 *)

proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 1)

Proved
(* D . 3 . U . QED BC *)

ack\#1.seqno = next sequence (infrom (ack_send, myid))
& unique_msg (outto (pkt_send, myid)) = infrom (source, myid)
& size (unique_msg (outto (pkt_send, myid)))
= size (unique_msg (infrom (ack_send, myid))) + 1
& proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid), 1)

Proved
QED

Prv \rightarrow $Proceeding$

(* D . 3 . *)

proper_transmission (infrom (source, myid), outto (pkt_send, myid),
infrom (ack_send, myid) a [seq: ack\#1], 0)

Proved
Proofs of the VCs in Gypsy

Prvr=> sProceeding
   sender#33
   proved in theorem prover.

Exec=> show status sender

   Proved: SENDER
   sender#1 . . . sender#33 proved.

Exec=> show status receiver

   receiver
   waiting to be proved: RECEIVER#4, RECEIVER#5, RECEIVER#6, RECEIVER#8,
   RECEIVER#12, RECEIVER#13, RECEIVER#14, RECEIVER#18
   Proved in VC generator: RECEIVER#1, RECEIVER#2, RECEIVER#3, RECEIVER#7,
   RECEIVER#9, RECEIVER#10, RECEIVER#11, RECEIVER#15, RECEIVER#16,
   RECEIVER#17

Exec=> prove receiver#4
Entering Prover with verification condition receiver#4

   C1: next-seqnum (null (#sectype#)) = one
   C2: proper-reception (null (#sectype#), null (#sectype#), null (#sectype#),
       0)

Backup point
(.,)

Prvr=> sProceeding
Backup point
(., 1.,)

Prvr=> expand next-seqnum
Backup point
(., 1., E.,)

Prvr=> theorem

   C1: comp (last-bit (seqnums (null (#sectype#)))) = one

Prvr=> expand seqnums
Backup point
(., 1., E., E.,)

Prvr=> expand last-bit
Backup point
(., 1., E., E., E.,)

Prvr=> expand comp
Proofs of the VCs in Gvisy

Backup point
(· 1 · E · E · E · E ·)

Prvrr→ simplify theorem

Prvrr→ sProceeding
(· 1 ·)

next-seqnum (null (#seqtype#)) = one

Proved

Prvrr→ sProceeding
Backup point
(· 2 ·)

Prvrr→ theorem

C1: proper-reception (null (#seqtype#), null (#seqtype#), null (#seqtype#), 0)

Prvrr→ expand proper-reception
Backup point
(· 2 · E ·)

Prvrr→ theorem

C1: 0 = size (unique_msg (null (#seqtype#)))
C2: msg-lag (null (#seqtype#), unique_msg (null (#seqtype#)), 0)

Prvrr→ expand unique_msg
Backup point
(· 2 · E · E ·)

Prvrr→ expand msg-lag
Backup point
(· 2 · E · E · E ·)

Prvrr→ theorem

C1: 0 = size (null (msg-seq))
C2: initial-subseq (null (#seqtype#), null (msg-seq))

Prvrr→ simplify theorem

Prvrr→ use eq-iss:lemmas
Backup point
(· 2 · E · E · E · E ·)

Prvrr→ qed
(· 2 · E · E · E · E · U · QED)
(· 2 · E · E · E · U · QED BC)
(· 2 · E · E · E · U · QED BC)
Proofs of the VC\textsubscript{s} in Gypsy

\begin{verbatim}
null (#seqtype\#) = null (#msg-seq)
Proved
QED

Prvr\rightarrow sProceeding
(\cdot 2 , )

proper-reception (null (#seqtype\#), null (#seqtype\#), null (#seqtype\#), 0)
Proved

Prvr\rightarrow sProceeding
receiver\#4
proved in theorem prover.

Exec\rightarrow prove receiver\#5
Entering Prover with verification condition receiver\#5

H1: empty (pkt_rcv\#1)
H2: next_seqnum (infrom (pkt_rcv, myid)) = exp
H3: next_seqnum (outto (ack_rcv, myid)) = exp
H4: infrom (pkt_rcv, myid) = infrom (pkt_rcv\#1, myid)
H5: outto (pkt_rcv, myid) = outto (pkt_rcv\#1, myid)
H6: proper-reception (outto (sink, myid), infrom (pkt_rcv, myid),
     outto (ack_rcv, myid), 0)

\rightarrow

C1: proper-reception (outto (sink, myid), infrom (pkt_rcv, myid),
     outto (ack_rcv, myid), 1)

Backup point
(\cdot )

Prvr\rightarrow retain n\#6
Backup point
(\cdot D , )

Prvr\rightarrow use prop_rec_3::lemmas
Backup point
(\cdot D , U , )

Prvr\rightarrow qed
(\cdot D , U , QED)
::(\cdot D , U , QED RC)
(\cdot D , U , QED RC)

proper-reception (outto (sink, myid), infrom (pkt_rcv, myid),
     outto (ack_rcv, myid), 0)

Proved
QED

Prvr\rightarrow sProceeding
receiver\#5
proved in theorem prover.
\end{verbatim}
Proofs of the VCs in Gypsy

\textbf{Exec-> prove receiver\#6}

\textbf{Entering Prover with verification condition receiver\#6}

H1: `infrom (pkt_rcv, myid) \& (seq: pack\#1) = infrom (pkt_rcv\#2, myid)`
H2: `pack\#1.seqno = exp`
H3: `next_seqnum (`infrom (pkt_rcv, myid)) = exp`
H4: `next_seqnum (outto (ack_rcv, myid)) = exp`
H5: `infrom (sink, myid) = infrom (sink\#1, myid)`
H6: `outto (pkt_rcv, myid) = outto (pkt_rcv\#2, myid)`
H7: `outto (sink, myid) = outto (sink\#1, myid)`
H8: `full (sink\#1)`
H9: `proper_reception (outto (sink, myid), infrom (pkt_rcv, myid), outto (ack_rcv, myid), 0)`

\[\Rightarrow\]

C1: `proper_reception (outto (sink, myid), infrom (pkt_rcv, myid) \& [seq: pack\#1], outto (ack_rcv, myid), 1)`

\textbf{Backup point}

( . )

\textbf{Prvr-> retain h9}

\textbf{Backup point}

( . D . )

\textbf{Prvr-> use prop_rec_2::lemmas}

\textbf{Backup point}

( . D . U . )

\textbf{Prvr-> qed}

( . D . U . QED )

\[\Rightarrow ( . D . U . QED \text{ RC})

( . D . U . QED \text{ RC})

\textbf{proper_reception (outto (sink, myid), infrom (pkt_rcv, myid), outto (ack_rcv, myid), 0)}

\textbf{Proved}

\textbf{QED}

\textbf{Prvr-> $\text{Proceeding}$}

\textbf{receiver\#6}

proved in theorem prover.

\textbf{Exec-> prove receiver\#8}

\textbf{Entering Prover with verification condition receiver\#8}

H1: `infrom (pkt_rcv, myid) \& [seq: pack\#1] = infrom (pkt_rcv\#2, myid)`
H2: `outto (sink, myid) \& [seq: pack\#1,mss\#] = outto (sink\#2, myid)`
H3: `pack\#1,seqno = exp`
H4: `next_seqnum (`infrom (pkt_rcv, myid)) = exp`
H5: `next_seqnum (outto (ack_rcv, myid)) = exp`
H6: `infrom (ack_rcv, myid) = infrom (ack_rcv\#1, myid)"
Proofs of the VCs in Gypsy

H7: infrom (sink, myid) = infrom (sink=2, myid)
H8: outto (ack_rcv, myid) = outto (ack_rcv=1, myid)
H9: outto (pkt_rcv, myid) = outto (pkt_rcv=2, myid)
H10: full (ack_rcv=1)
H11: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
    outto (ack_rcv, myid), 0)

=>

C1: proper_reception (outto (sink, myid) a [seq: pack#1.mssq],
    infrom (pkt_rcv, myid) a [seq: pack#1],
    outto (ack_rcv, myid), 1)

backup point
()

Prvr=> retain h3,h4,h11
Backup point
( . D . )

Prvr=> use prop_rec_1::lemmas
Backup point
( . D . U . )

Prvr=> qed
( . D . U . QED )
( . D . U . QED BC 1 )
( . D . U . QED BC 1 )

pack#1.mssq = pack#1.mssq

Proved
( . D . U . QED BC 2 )
: : : :
pack#1.seqno := exp
( . D . U . QED BC 2 =S )
( . D . U . QED BC 2 )

pack#1.seqno = next_seqnum (infrom (pkt_rcv, myid))

Proved
( . D . U . QED BC 3 )
( . D . U . QED BC 3 )

proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
    outto (ack_rcv, myid), 0)

Proved
( . D . U . QED BC )

pack#1.mssq = pack#1.mssq
& pack#1.seqno = next_seqnum (infrom (pkt_rcv, myid))
& proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
    outto (ack_rcv, myid), 0)

Proved
QED
Proofs of the VCs in Gyrcv

Prvr-> sProceeding
receiver=13
proved in theorem prover.

Exec-> prove receiver=13
Entering Prover with verification condition receiver=13

H1: empty (pkt_rcv#1)
H2: next_seqnum (infrom (pkt_rcv, myid)) = exp
H3: next_seqnum (outto (ack_rcv, myid)) = exp
H4: infrom (pkt_rcv, myid) = infrom (pkt_rcv#1, myid)
H5: outto (pkt_rcv, myid) = outto (pkt_rcv#1, myid)
H6: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
    outto (ack_rcv, myid), 0)

->

C1: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
    outto (ack_rcv, myid), 1)

Backup point
(*)

Prvr-> retain n6
Backup point
(* D *)

Prvr-> use prop_rec_3::lemmas
Backup point
(* D U *)

Prvr-> qed
(* D U QED)
:<=: (* D U QED BC)
(* D U QED BC)

proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
    outto (ack_rcv, myid), 0)

Proved

QED

Prvr-> sProceeding
receiver=13
proved in theorem prover.

Exec-> prove receiver=14
Entering Prover with verification condition receiver=14

H1: infrom (pkt_rcv, myid) @ [seq: pack#1] = infrom (pkt_rcv#2, myid)
H2: next_seqnum (infrom (pkt_rcv, myid)) = exp
H3: next_seqnum (outto (ack_rcv, myid)) = exp
H4: infrom (ack_rcv, myid) = infrom (ack_rcv#3, myid)
H5: outto (ack_rcv, myid) = outto (ack_rcv#3, myid)
H6: outto (pkt_rcv, myid) = outto (pkt_rcv#2, myid)
H7: full (ack_rcv#3)
Proofs of the VCs in Gypsy

H8: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
outto (ack_rcv, myid), 0)
H9: pack#1, seqno re exp

\[ \Rightarrow \]
C1: proper_reception (outto (sink, myid),
infrom (pkt_rcv, myid) + [seq: pack#1],
outto (ack_rcv, myid), 1)

Backup point
(
)

\textbf{Prvr-> retain H8}
Backup point
(, D , )

\textbf{Prvr-> use prop_rec_2::lemmas}
Backup point
(, D , U , )

\textbf{Prvr-> qed}
(, D , U , QED)
::(, D , U , QED BC)
(, D , U , QED RC)

\textbf{proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),}
outto (ack_rcv, myid), 0)

Proved
QED

\textbf{Prvr-> sProceeding}
receiver#14
proved in theorem prover.

\textbf{Exec-> save app.dmp}

File \texttt{APP.DMP} already exists. Rewrite it? \[ \Rightarrow \] y

\textbf{Saving..........................}

\textbf{Exec-> show status receiver}

receiver
\# failing to be proved: RECEIVER#12, RECEIVER#18
Proved in VC generator: RECEIVER#1, RECEIVER#2, RECEIVER#3, RECEIVER#7,
RECEIVER#9, RECEIVER#10, RECEIVER#11, RECEIVER#15, RECEIVER#16,
RECEIVER#17
Proved in theorem prover: RECEIVER#4, RECEIVER#5, RECEIVER#6,
RECEIVER#8, RECEIVER#13, RECEIVER#14
Proofs of the VCs in Gvrsy

Exec => prove receiver#12
Entering Prover with verification condition receiver#12

H1: infom (pkt_rcv, myid) @ [seq: pack#1] = infom (pkt_rcv#2, myid)
H2: outto (ack_rcv, myid) @ [seq: pack#1] = outto (ack_rcv#2, myid)
H3: outto (sink, myid) @ [seq: pack#1, mssg] = outto (sink#2, myid)
H4: pack#1, seqno = exp
H5: next_seqnum (infom (pkt_rcv, myid)) = exp
H6: next_seqnum (outto (ack_rcv, myid)) = exp
H7: infom (ack_rcv, myid) = infom (ack_rcv#2, myid)
H8: infom (sink, myid) = infom (sink#2, myid)
H9: outto (pkt_rcv, myid) = outto (pkt_rcv#2, myid)
H10: proper_reception (outto (sink, myid), infom (pkt_rcv, myid),
     outto (ack_rcv, myid), 0)

=>
C1: comp (exp)
   = next_seqnum (infom (pkt_rcv, myid) @ [seq: pack#1])
C2: comp (exp) = next_seqnum (outto (ack_rcv, myid) @ [seq: pack#1])
C3: proper_reception (outto (sink, myid) @ [seq: pack#1, mssg],
   infom (pkt_rcv, myid) @ [seq: pack#1],
   outto (ack_rcv, myid) @ [seq: pack#1], 0)

Backup point
( )

Prvr => retain h4,h5,h6,h10
Backup point
( . D . )

Prvr => sProceeding
Backup point
( . D . 1 . )

Prvr => use next_comp::lemmas
Backup point
( . D . 1 . U . )

Prvr => theorem

H1: comp (p#8, seqno) = next_seqnum (x#12 @ [seq: p#8])
H2: pack#1, seqno = exp
H3: next_seqnum (infom (pkt_rcv, myid)) = exp
H4: next_seqnum (outto (ack_rcv, myid)) = exp
H5: proper_reception (outto (sink, myid), infom (pkt_rcv, myid),
     outto (ack_rcv, myid), 0)

=>
C1: comp (exp)
   = next_seqnum (infom (pkt_rcv, myid) @ [seq: pack#1])

Prvr => put
For what? p#8;
Proofs of the VCs in Gypsy

Put what?* pack#1;

For what?* $done

Backup point
(.* D . 1 . U . PUT . *)

Prvr=> equality substitute h3
H3 yields
  exp := next-seqnum (infrom (pkt_rcv, myid))
Backup point
(.* D . 1 . U . PUT . *=S . *)

Prvr=> theorem

  H1: pack#1.seqno = next-seqnum (infrom (pkt_rcv, myid))
  H2: comp (pack#1.seqno) = next-seqnum (x#125 @ [seq: pack#1])
  H3:  next-seqnum (infrom (pkt_rcv, myid))
        = next-seqnum (outto (ack_rcv, myid))
  H4: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
      outto (ack_rcv, myid), 0)

=>
  C1:  comp (next-seqnum (infrom (pkt_rcv, myid)))
       = next-seqnum (infrom (pkt_rcv, myid)) @ [seq: pack#1])

Prvr=> equality substitute h1
H1 yields
  next-seqnum (infrom (pkt_rcv, myid)) := pack#1.seqno
Backup point

Prvr=> sProceeding
;(* D . 1 . *)
    comp (exp) = next-seqnum (infrom (pkt_rcv, myid)) @ [seq: pack#1])
Proved

Prvr=> sProceeding
Backup point
(.* D . 2 . *)

Prvr=> sProceeding
;:;:;:;:;:;:;:;:;Ran out of tricks

Prvr=> theorem

  H1: pack#1.seqno = exp
  H2: next-seqnum (infrom (pkt_rcv, myid)) = exp
  H3: next-seqnum (outto (ack_rcv, myid)) = exp
  H4: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
      outto (ack_rcv, myid), 0)

=>
Proofs of the VCs in Gypsy

C1: \text{comp (exp)} = \text{next\_seqnum (outto (ack\_rcv, myid)) @ [seq: pack#1]}

Prv\text{r}\rightarrow \text{equality substitute h1}
\text{H1 yields}
\text{exp} := \text{pack#1\_seqno}
Backup point
(., D . 2 . =S . D .)

Prv\text{r}\rightarrow \text{theorem}

\text{H1: pack#1\_seqno = next\_seqnum (infrom (pkt\_rcv, myid))}
\text{H2: pack#1\_seqno = next\_seqnum (outto (ack\_rcv, myid))}
\text{H3: proper\_reception (outto (sink, myid), infrom (pkt\_rcv, myid), outto (ack\_rcv, myid), 0)}

\Rightarrow

\text{C1: comp (pack#1\_seqno) = next\_seqnum (outto (ack\_rcv, myid)) @ [seq: pack#1]}

Prv\text{r}\rightarrow \text{return h2}
Backup point
(., D . 2 . =S . D .)

Prv\text{r}\rightarrow \text{use next\_comp\_lemmas}
Backup point
(., D . 2 . =S . D . U .)

Prv\text{r}\rightarrow \text{theorem}

\text{H1: comp (o#10s\_seqno) = next\_seqnum (x#14s @ [seq: o#10s])}
\text{H2: pack#1\_seqno = next\_seqnum (outto (ack\_rcv, myid))}

\Rightarrow

\text{C1: comp (pack#1\_seqno) = next\_seqnum (outto (ack\_rcv, myid)) @ [seq: pack#1]}

Prv\text{r}\rightarrow \text{proceeding}
(., D . 2 .)

\text{comp (exp) = next\_seqnum (outto (ack\_rcv, myid)) @ [seq: pack#1]}
Proved

Prv\text{r}\rightarrow \text{proceeding}
Backup point
(., D . 3 .)

Prv\text{r}\rightarrow \text{theorem}

\text{H1: pack#1\_seqno = exp}
\text{H2: next\_seqnum (infrom (pkt\_rcv, myid)) = exp}
\text{H3: next\_seqnum (outto (ack\_rcv, myid)) = exp}
\text{H4: proper\_reception (outto (sink, myid), infrom (pkt\_rcv, myid), outto (ack\_rcv, myid), 0)}

\Rightarrow
Proofs of the VCs in Goby

Ci: proper Reception (outto (sink, myid) & [seq: pack1, mssq],
infrom (pkt_rcv, myid) & [seq: pack1],
outto (ack_rcv, myid) & [seq: pack1], 0)

Prvr=> use prop_rec_4::lemmas
backup point
( . D . 3 . U . )

Prvr=> ned
( . D . 3 . U . QED )
( . D . 3 . U . QED BC )
( . D . 3 . U . QED BC 1 )
( . D . 3 . U . QED BC 1 )

pack1, mssq = pack1, mssq
Proved
( . D . 3 . U . QED BC 2 )

pack1, seqno := seq
( . D . 3 . U . QED BC 2 =S )
( . D . 3 . U . QED BC 2 )

pack1, seqno = next_seqnum (infrom (pkt_rcv, myid))
Proved
( . D . 3 . U . QED BC 3 )

pack1, seqno := exp
( . D . 3 . U . QED BC 3 =S )
( . D . 3 . U . QED BC 3 )

pack1, seqno = next_seqnum (outto (ack_rcv, myid))
Proved
( . D . 3 . U . QED BC 4 )
( . D . 3 . U . QED BC 4 )

proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
outto (ack_rcv, myid), 0)
Proved
( . D . 3 . U . QED BC )

pack1, mssq = pack1, mssq
& pack1, seqno = next_seqnum (infrom (pkt_rcv, myid))
& pack1, seqno = next_seqnum (outto (ack_rcv, myid))
& proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
outto (ack_rcv, myid), 0)
Proved
QED

Prvr=> sProceeding
( . D . 3 . )

proper_reception (outto (sink, myid) & [seq: pack1, mssq],
Proofs of the VCs in Cynsys

\[
\begin{align*}
\text{infrom} (\text{pkt_rcv}, \text{myid}) &\triangleq \{\text{seq: pack#1}, 0\} \\
\text{outto} (\text{ack_rcv}, \text{myid}) &\triangleq \{\text{seq: pack#1}, 0\}
\end{align*}
\]

Proved

Provr\textsuperscript{\texttt{-> sProceeding}}
receiver\#12
proved in theorem prover.

Exec\textsuperscript{\texttt{-> prove receiver#19}}
Entering Prover with verification condition receiver\#18

\begin{align*}
H1: \text{infrom} (\text{pkt_rcv}, \text{myid}) &\triangleq \{\text{seq: pack#1}\} = \text{infrom} (\text{pkt_rcv#2}, \text{myid}) \\
H2: \text{outto} (\text{ack_rcv}, \text{myid}) &\triangleq \{\text{seq: pack#1}\} = \text{outto} (\text{ack_rcv#4}, \text{myid}) \\
H3: \text{next_seqnum} (\text{infrom} (\text{pkt_rcv}, \text{myid})) = \text{exp} \\
H4: \text{next_seqnum} (\text{outto} (\text{ack_rcv}, \text{myid})) = \text{exp} \\
H5: \text{infrom} (\text{ack_rcv}, \text{myid}) = \text{infrom} (\text{ack_rcv#4}, \text{myid}) \\
H6: \text{outto} (\text{pkt_rcv}, \text{myid}) = \text{outto} (\text{pkt_rcv#2}, \text{myid}) \\
H7: \text{proper_reception} (\text{outto} (\text{sink}, \text{myid}), \text{infrom} (\text{pkt_rcv}, \text{myid}), \\
\text{outto} (\text{ack_rcv}, \text{myid}), 0) \\
H8: \text{pack#1}, \text{seqno} \neq \text{exp}
\end{align*}

\[\Rightarrow\]

\begin{align*}
C1: \text{next_seqnum} (\text{infrom} (\text{pkt_rcv}, \text{myid}) &\triangleq \{\text{seq: pack#1}\}) = \text{exp} \\
C2: \text{next_seqnum} (\text{outto} (\text{ack_rcv}, \text{myid}) &\triangleq \{\text{seq: pack#1}\}) = \text{exp} \\
C3: \text{proper_reception} (\text{outto} (\text{sink}, \text{myid}), \\
\text{infrom} (\text{pkt_rcv}, \text{myid}) &\triangleq \{\text{seq: pack#1}\}, \\
\text{outto} (\text{ack_rcv}, \text{myid}), 0)
\end{align*}

\textbf{backup point}
\texttt{(.)}

Prvr\textsuperscript{\texttt{-> retain h3,h4,h7,h8}}
Backup point
\texttt{(., D .)}

Prvr\textsuperscript{\texttt{-> sProceeding}}
Backup point
\texttt{(., D . 1 .)}

Prvr\textsuperscript{\texttt{-> theorem}}

\begin{align*}
H1: \text{next_seqnum} (\text{infrom} (\text{pkt_rcv}, \text{myid})) = \text{exp} \\
H2: \text{next_seqnum} (\text{outto} (\text{ack_rcv}, \text{myid})) = \text{exp} \\
H3: \text{proper_reception} (\text{outto} (\text{sink}, \text{myid}), \text{infrom} (\text{pkt_rcv}, \text{myid}), \\
\text{outto} (\text{ack_rcv}, \text{myid}), 0) \\
H4: \text{pack#1}, \text{seqno} \neq \text{exp}
\end{align*}

\[\Rightarrow\]

\begin{align*}
C1: \text{next_seqnum} (\text{infrom} (\text{pkt_rcv}, \text{myid}) &\triangleq \{\text{seq: pack#1}\}) = \text{exp}
\end{align*}

Prvr\textsuperscript{\texttt{-> retain h1,h4}}
Backup point
\texttt{(., D . 1 . D .)}

Prvr\textsuperscript{\texttt{-> use next_conn::lemmas}}
Proofs of the VCs in Gypsy

Backup point
(* D . 1 . D . U .)

Prvr -> use comp::lemmas
Backup point
(* D . 1 . D . U . U .)

Prvr -> theorem

H1: comp (b1#2s) = b2#2s iff b1#2s ne b2#2s
H2: comp (p#14s, seqno) = next_seqnum (x#12s @ [seq: p#14s])
H3: next_seqnum (infrom (pkt_rcv, myid)) = exp
H4: pack#1, seqno ne exp

=>
C1: next_seqnum (infrom (pkt_rcv, myid) @ [seq: pack#1]) = exp

Prvr -> forwardchain h1
which *way? (left or right)
left
::: Forward chaining gives

comp (pack#1, seqno) = exp
Backup point

Prvr -> theorem

H1: comp (pack#1, seqno) = exp
H2: comp (b1#2s) = b2#2s iff b1#2s ne b2#2s
H3: comp (p#14s, seqno) = next_seqnum (x#12s @ [seq: p#14s])
H4: next_seqnum (infrom (pkt_rcv, myid)) = exp
H5: pack#1, seqno ne exp

=>
C1: next_seqnum (infrom (pkt_rcv, myid) @ [seq: pack#1]) = exp

Prvr -> equality substitute h1
H1 yields
exp := comp (pack#1, seqno)
Backup point

Prvr -> theorem

H1: comp (p#14s, seqno) = next_seqnum (x#12s @ [seq: p#14s])
H2: comp (pack#1, seqno) = next_seqnum (infrom (pkt_rcv, myid))
H3: comp (b1#2s) = b2#2s iff b1#2s ne b2#2s
H4: pack#1, seqno ne comp (pack#1, seqno)

=>
C1: comp (pack#1, seqno)
   = next_seqnum (infrom (pkt_rcv, myid) @ [seq: pack#1])

Prvr -> retain h1, h2
Proofs of the VCs in Gypsy

Backup point

Prvr-> sProceeding
(* D . 1 . *)

next_seqnum (infrom (pkt_rcv, myid) @ [seq: pack#1]) = exp

Proved

Prvr-> sProceeding
Backup point
(* D . 2 . *)

Prvr-> theorem

H1: next_seqnum (infrom (pkt_rcv, myid)) = exp
H2: next_seqnum (outto (ack_rcv, myid)) = exp
H3: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
    outto (ack_rcv, myid), @)
H4: pack#1.seqno ne exp

=>
C1: next_seqnum (outto (ack_rcv, myid) & [seq: pack#1]) = exp

Prvr-> retain h2,h4
Backup point
(* D . 2 . D . *)

Prvr-> use next_comp::lemmas
Backup point
(* D . 2 . D . U . *)

Prvr-> use comp::lemmas
Backup point
(* D . 2 . D . U . U . *)

Prvr-> theorem

H1: comp (b1#4s) = h2#4s iff h1#4s ne h2#4s
H2: comp (o#16s, seqno) = next_seqnum (x#20s @ [seq: p#16s])
H3: next_seqnum (outto (ack_rcv, myid)) = exp
H4: pack#1.seqno ne exp

=>
C1: next_seqnum (outto (ack_rcv, myid) & [seq: pack#1]) = exp

Prvr-> forwardchain h1
Which way? (left or right)
left

::Forward chaining gives

comp (pack#1, seqno) = exp
Backup point
(* D . 2 . D . U . U . FC . *)
Proofs of the VCs in Gvsys

Prvr=> hypothesis

H1: comp (pack#1, seqno) = exp
H2: comp (h1#4$) = h2#4$ iff h1#4$ ne h2#4$
H3: comp (o1#6$, seqno) = next_seqnum (x#20$ @ (seq: p#16$))
H4: next_seqnum (outto (ack_rcv, myid)) = exp
H5: pack#1, seqno ne exp

Prvr=> equality substitute h1
H1 yields
exp := comp (pack#1, seqno)
Backup point

Prvr=> theorem

H1: comp (p#16$, seqno) = next_seqnum (x#20$ @ (seq: p#16$))
H2: comp (pack#1, seqno) = next_seqnum (outto (ack_rcv, myid))
H3: comp (o1#4$) = h2#4$ iff h1#4$ ne h2#4$
H4: pack#1, seqno ne comp (pack#1, seqno)

=>
C1: comp (pack#1, seqno)
= next_seqnum (outto (ack_rcv, myid) @ (seq: pack#1))

Prvr=> retain h1, h2
Backup point

Prvr=> sProceeding
(., D . 2 .)

next_seqnum (outto (ack_rcv, myid) @ (seq: pack#1)) = exp
proved

Prvr=> sProceeding
Backup point
(., D . 3 .)

Prvr=> theorem

H1: next_seqnum (infrom (pkt_rcv, myid)) = exp
H2: next_seqnum (outto (ack_rcv, myid)) = exp
H3: proper_reception (outto (sink, myid), infrom (pkt_rcv, myid),
outto (ack_rcv, myid), 0)
H4: pack#1, seqno ne exp

=>
C1: proper_reception (outto (sink, myid),
infrom (pkt_rcv, myid) @ (seq: pack#1),
outto (ack_rcv, myid) @ (seq: pack#1), 0)

Prvr=> use prop_rec_5::lemmas
Proofs of the VCs in Gypsy

Backum point
(* D . 3 . U .)

Prv => qed
(* D . 3 . U . QED)

--------------------
(* D . 3 . U . QED BC 1)
(* D . 3 . U . QED BC 1)

proper_reception (outto (sink, myid), infrom (pckt_rcv, myid),
outto (ack_rcv, myid), 0)

Proved
(* D . 3 . U . QED BC 2)
--------------------
next_seqnum (infrom (pckt_rcv, myid)) := exo
(* D . 3 . U . QED BC 2 =S)
(* D . 3 . U . QED BC 2)

pack#1, seqno ne next_seqnum (infrom (pckt_rcv, myid))

Proved
(* D . 3 . U . QED BC 3)
--------------------
next_seqnum (outto (ack_rcv, myid)) := exo
(* D . 3 . U . QED BC 3 =S)
(* D . 3 . U . QED BC 3)

pack#1, seqno ne next_seqnum (outto (ack_rcv, myid))

Proved
(* D . 3 . U . QED BC)

proper_reception (outto (sink, myid), infrom (pckt_rcv, myid),
outto (ack_rcv, myid), 0)
& pack#1, seqno ne next_seqnum (infrom (pckt_rcv, myid))
& pack#1, seqno ne next_seqnum (outto (ack_rcv, myid))

Proved
QED

Prv => sProceeding
(* D . 3 .)

proper_reception (outto (sink, myid), infrom (pckt_rcv, myid)
a [seq: pack#1],
outto (ack_rcv, myid) a [seq: pack#1], 0)

Proved

Prv => sProceeding
receiver#18
proved in theorem prover.

Exec => show status receiver
Proofs of the VCs in Gypsy

Proved: RECEIVER
receiver#1 . . . receiver#18 proved.

Exec-⇒ prove ab_protocol
ab_protocol#1
proved in VC generator.
Used in proof: TIMER, RECEIVER, MEDIUM, SENDER
Entering Prover with verification condition ab_protocol#2

H1: proper_reception (outto (sink#1, receiver#1), allfrom (pkt_rcv#1),
    allto (ack_rcv#1), 1)
H2: proper_transmission (infrom (source#1, sender#1), allto (pkt_send#1),
    allfrom (ack_send#1), 1)
H3: isblocked (medium#1)
H4: isblocked (medium#2)
H5: isblocked (receiver#1)
H6: isblocked (sender#1)
H7: isblocked (timer#1)
H8: allto (ack_send#1) sub allfrom (ack_rcv#1)
H9: allto (pkt_rcv#1) sub allfrom (pkt_send#1)
⇒
C1: msg_lag (outto (sink#1, receiver#1), infrom (source#1, sender#1), 1)
Backup point
( .)
Prvr-⇒ retain h1, h2, h8, h9
Backup point
( . D . )
Prvr-⇒ use abp-1::lemmas
Backup point
( . D . U . )
Prvr-⇒ theorem

H1: proper_reception (v#2s, x#2s, y#2s, 1)
    & propr_transmission (w#2s, z#2s, y#2s, 1) & x#2s sub w#2s
    & z#2s sub y#2s
⇒ msg_lag (v#2s, u#2s, 1)
H2: proper_reception (outto (sink#1, receiver#1), allfrom (pkt_rcv#1),
    allto (ack_rcv#1), 1)
H3: proper_transmission (infrom (source#1, sender#1), allto (pkt_send#1),
    allfrom (ack_send#1), 1)
H4: allto (ack_send#1) sub allfrom (ack_rcv#1)
H5: allto (pkt_rcv#1) sub allfrom (pkt_send#1)
⇒
C1: msg_lag (outto (sink#1, receiver#1), infrom (source#1, sender#1), 1)
Prvr-⇒ qed
( . D . U . QED)
( . D . U . QED HC)
( . D . U . QED AC 1)
Proofs of the VC's in Gypsy

:(. D . U . QED BC 1)

proper_reception (outto (sink#1, receiver#1), x#2s, y#2s, 1)
Proved
(. D . U . QED BC 2)

::: (. D . U . QED BC 2)

proper_transmission (infrom (source#1, sender#1), w#2s, z#2s, 1)
Proved
(. D . U . QED BC 3)

:..........:........ Ran out of tricks

Prvr⇒ theorem

H1: proper_reception (outto (sink#1, receiver#1), allfrom (pkt_rcv#1),
    allto (ack_rcv#1), 1)
H2: proper_transmission (infrom (source#1, sender#1), allto (pkt_send#1),
    allfrom (ack_send#1), 1)
H3: allto (ack_send#1) sub allfrom (ack_rcv#1)
H4: allto (pkt_rcv#1) sub allfrom (pkt_send#1)
⇒
C1: allfrom (pkt_rcv#1) sub allto (pkt_send#1)

Prvr⇒ janel
(. D . U . QED BC 3)

Prvr⇒ back up 3
Restored
Backup point
(. D . U . QED .)

Prvr⇒ back up 2

### Illegal command = UP2
Back UP or establish a back up POINT to return to? up

How many levels would you like to back up? 2

Restored
Backup point
(. D . U .)

Prvr⇒ theorem

H1: proper_reception (v#2s, x#2s, y#2s, 1) & proper_transmission (u#2s, w#2s, z#2s, 1) & x#2s sub w#2s & z#2s sub y#2s
⇒ msg_lad (v#2s, u#2s, 1)
H2: proper_reception (outto (sink#1, receiver#1), allfrom (pkt_rcv#1),
    allto (ack_rcv#1), 1)
Proofs of the VC5 in GyPSV

H3: proper_transmission (infrm (source#1, sender#1), allto (pkt_send#1),
    allfrom (ack_send#1), 1)

H4: allto (ack_send#1) sub allfrom (ack_rcv#1)
H5: allto (pkt_rcv#1) sub allfrom (pkt_send#1)

⇒
C1: msg_lag (outto (sink#1, receiver#1), infrm (source#1, sender#1), 1)

Prv⇒ claim
Newgoal:
* allfrom(ack_send#1) sub allto(ack_rcv#1)
* & allfrom(pkt_rcv#1) sub allto(pkt_send#1);
Defer proof of claim? y

Backup point
( D . U . . )

Prv⇒ theorem

H1: allfrom (ack_send#1) sub allto (ack_rcv#1)
H2: allfrom (pkt_rcv#1) sub allto (pkt_send#1)
H3: & proper_reception (v#2s, x#2s, y#2s, 1)
    & proper_transmission (u#2s, w#2s, z#2s, 1) & x#2s sub w#2s
    & z#2s sub y#2s
⇒ msg_lag (v#2s, u#2s, 1)
H4: proper_reception (outto (sink#1, receiver#1), allfrom (pkt_rcv#1),
    allto (ack_rcv#1), 1)
H5: proper_transmission (infrm (source#1, sender#1), allto (pkt_send#1),
    allfrom (ack_sender#1), 1)
H6: allto (ack_send#1) sub allfrom (ack_rcv#1)
H7: allto (pkt_rcv#1) sub allfrom (pkt_send#1)

⇒
C1: msg_lag (outto (sink#1, receiver#1), infrm (source#1, sender#1), 1)

Prv⇒ qed
( D . U . . . QED)
(:(:(:(: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (: (QED RC)
( D . U . . . QED RC 1)
( D . U . . . QED RC 1)

proper_reception (outto (sink#1, receiver#1), x#2s, y#2s, 1)
Proved
( D . U . . . QED RC 2)
( D . U . . . QED RC 2)

proper_transmission (infrm (source#1, sender#1), w#2s, z#2s, 1)
Proved
( D . U . . . QED RC 3)
( D . U . . . QED RC 3)

allfrom (pkt_rcv#1) sub allto (pkt_send#1)
Proved
( D . U . . . QED RC 4)
Proofs of the VCs in Gypsy

(* D . U . . QED BC 4)

\text{allfrom (ack\_send#1) sub allto (ack\_rcv#1)}
Proved
(* D . U . . QED BC)

\text{proper\_reception (outto (sink#1, receiver#1), x#2s, y#2s, 1) & proper\_transmission (infrom (source#1, sender#1), w#2s, z#2s, 1)} & x#2s sub w#2s & z#2s sub y#2s
Proved
QED
Prvr=> s Proceeding
Proved using CLAIM, now prove the claim.
Backup point
(CLAIM .)

Prvr=> theorem

H1:  \text{proper\_reception (v#2s, x#2s, y#2s, 1) & proper\_transmission (u#2s, w#2s, z#2s, 1) & x#2s sub w#2s & z#2s sub y#2s}

\Rightarrow \text{msg\_lag (v#2s, u#2s, 1)}

H2: \text{proper\_reception (outto (sink#1, receiver#1), allfrom (pkt\_rcv#1), allto (ack\_rcv#1), 1)}

H3: \text{proper\_transmission (infrom (source#1, sender#1), allto (pkt\_send#1), allfrom (ack\_send#1), 1)}

H4: allto (ack\_send#1) sub allfrom (ack\_rcv#1)

H5: allto (pkt\_rcv#1) sub allfrom (pkt\_send#1)

\Rightarrow

C1: allfrom (ack\_send#1) sub allto (ack\_rcv#1)

C2: allfrom (pkt\_rcv#1) sub allto (pkt\_send#1)

Prvr=> retain h4, h5
Backup point
(CLAIM . D .)

Prvr=> s Proceeding
Backup point
(CLAIM . D . 1 .)

Prvr=> theorem

H1: allto (ack\_send#1) sub allfrom (ack\_rcv#1)

H2: allto (pkt\_rcv#1) sub allfrom (pkt\_send#1)

\Rightarrow

C1: allfrom (ack\_send#1) sub allto (ack\_rcv#1)

Prvr=> delete h2
Backup point
(CLAIM . D . 1 . 0 .)
Proofs of the VCs in Gynsy

Prvr=> sProceeding
Ran out of tricks

Prvr=> simplify theorem

Prvr=> theorem

H1: allto (ack_send#1) sub allfrom (ack_recv#1)
->
C1: allfrom (ack_send#1) sub allto (ack_recv#1)

Prvr=> use hist_sub::lemmas
Backup point
(CLAIM . D . 1 . D . U .)

Prvr=> qed
(CLAIM . D . 1 . D . U . QED)
::(CLAIM . D . 1 . D . U . QED BC)
(CLAIM . D . 1 . D . U . QED BC)
   allfrom (ack_send#1) sub allfrom (ack_recv#1)
Proved
QED

Prvr=> sProceeding
(CLAIM . D . 1 .)
   allfrom (ack_send#1) sub allto (ack_recv#1)
Proved

Prvr=> sProceeding
Backup point
(CLAIM . D . 2 .)

Prvr=> theorem

H1: allto (ack_send#1) sub allfrom (ack_recv#1)
H2: allto (pkt_recv#1) sub allfrom (pkt_send#1)
->
C1: allfrom (pkt_recv#1) sub allto (pkt_send#1)

Prvr=> delete n1
Backup point
(CLAIM . D . 2 . D .)

Prvr=> use hist_sub::lemmas
Backup point
(CLAIM . D . 2 . D . U .)

Prvr=> qed
(CLAIM . D . 2 . D . U . QED)
::(CLAIM . D . 2 . D . U . QED BC)
(CLAIM . D . 2 . D . I . QED HC)

allfrom (pkt_rcv=1) sub allfrom (pkt_send=1)
proved
QED

Prvr-> $procedure

(CLAIM . D . 2 . )

allfrom (pkt_rcv=1) sub allto (pkt_send=1)
proved

Prvr-> $procedure

(CLAIM . )

allfrom (ack_send=1) sub allto (ack_rcv=1)
& allfrom (pkt_rcv=1) sub allto (pkt_send=1)
Established claim
ab_protocol#2
proved in theorem prover.
Used in proof: TIMER, RECEIVER, MEDIUM, SENDER
ab_protocol#3
proved in VC generator.
Used in proof: TIMER, RECEIVER, MEDIUM, SENDER
ab_protocol#4
proved in VC generator.
Used in proof: TIMER, RECEIVER, MEDIUM, SENDER
Exec-> show status scope $

SCOPE ALT_BIT_PROTOCOL

waiting for pending body to be filled in: MEDIUM, TIMER
proved: ab_protocol, comp, receiver, sender
Types, constants: bit, bit_seq, clk_buf, message, msg_buf, msg_seq, packet, pkt_buf, pkt_seq

Exec-> show status scope lemmas

SCOPE LEMMAS

waiting to be proved: abp_1, app_bitnonnull, app_msgnonnull,
app_pktnonnull, bit_cases, comp, eq issuing, eq issuing app, hist_sub,
iss app, iss_trans, interpolate, last next, last unique,
last repeats, main lemma, msg lag eq, ve next, next comp,
nchanges unique, prop_rec_1, prop_rec_2, prop_rec_3, prop_rec_4,
prop_rec_5, prop_trans_1, prop_trans_2, prop_trans_3, prop_trans_4,
prop_trans_5, prop_trans_6, prop_trans_7, sub_app, size null,
sub_seqnum, sub_to lag, sub_vchanges

Exec-> save abp.dmp
Proofs of the VCs in Gyosy

File ABP.DMP already exists. Rewrite it? -> y

Saving.................................................................

Exec-> exit no

[PHOTO: Recording terminated Sat 9-May-91 10:44AM]
It is now time to prove the lemmas. We will start with the higher level ones and move down to the lower level ones.

snow status all

The current design and verification status is:

SCOPE ALT-BIT-PROTOCOL

Waiting for pending body to be filled in: MEDIUM, TIMER
Proved: ABP-PROTOCOL, COMP, RECEIVER, SENDER
Types, constants: BIT, BIT_SEQ, CKI_BUF, MESSAGE, MSG_BUF, MSG_SEQ, PACKET, PKT_BUF, PKT_SEQ

SCOPE ALT-BIT-SPECS

Waiting for pending body to be filled in: INITIAL_SUBSEQ
For specifications only: LAST_BIT, MSG_LAG, NCHANGES, NEXT_SEQNUM, PROPER_RECEPTION, PROPER_TRANSMISSION, REPEATS, SEQUENTIAL, UNIQUE_MSG

SCOPE LEMMAS

Waiting to be proved: APP_1, APP_BIT_NONNULL, APP.MSG_NONNULL, APP_PKT_NONNULL, BIT_CASES, COMP_WE, EQ_ISS, EQ_ISS_APP, HIST_SUB, ISS_APP, ISS_TRANS, INTERPOLATE, LAST_NEXT, LAST_UNIQUE, LAST_REPEATS, MAIN_LEMMA, MSG_LAG_EQ, NE_NEXT, NEXT_Comp, NCHANGES_UNIQE, PROP_REC_1, PROP_REC_2, PROP_REC_3, PROP_REC_4, PROP_REC_5, PROP_TRANS_1, PROP_TRANS_2, PROP_TRANS_3, PROP_TRANS_4, PROP_TRANS_5, PROP_TRANS_6, PROP_TRANS_7, SUB_APP, SIZE_NULL, SUB_SEQNUM, SUB_TO_LAG, SUB_NCHANGES

Exec-> set scope lemmas

Exec-> prove prop_trans_1
Entering Prover with lemma prop_trans_1
all u : msg_seq,
Proofs of Supporting Lemmas in Gympy

```
all x, y : pkt_seq,
all p : packet,
all m : message,
d.mssg = m & p.seqno = next_seqnum (x) & proper_transmission (u, x, y, 0)
=> u & seq: m = unique_msg (x & seq: p)
```

H1: d.mssg = m
H2: p.seqno = next_seqnum (x)
H3: proper_transmission (u, x, y, 0)

=>

C1: u & seq: m = unique_msg (x & seq: p)

Backup point
(*)

Prv-> expand proper_transmission
Backup point
( . , E , . )

Prv-> simplify theorem

Prv-> theorem

```
H1: d.mssg = m
H2: p.seqno = next_seqnum (x)
H3: size (unique_msg (y)) = size (unique_msg (x))
H4: size (unique_msg (y)) = size (u)
H5: msg_lag (unique_msg (x), u, 0)
H6: repeats (x)

=>

C1: u & seq: m = unique_msg (x & seq: p)
```

Prv-> retain h1,h2,h5
Backup point
( . , E , O , . )

Prv-> use msg_lag_eq
Backup point
( . , E , O , . , ! , . )

Prv-> hypothesis

```
H1: u#2s = v#2s iff msg_lag (u#2s, v#2s, 0)
H2: d.mssg = m
H3: p.seqno = next_seqnum (x)
H4: msg_lag (unique_msg (x), u, 0)
```

Prv-> forwardchain h1
Which way? (left or right)
left
::: Forward chaining gives

Proofs of Supporting Lemmas in Gypsy

unique_msg(x) = u
Backup point
(* E . D . U . FC . *)

Prvr-> theorem

H1: unique_msg(x) = u
H2: u#2s = v#2s iff msg_lag(u#2s, v#2s, 0)
H3: p,mssg = m
H4: p,seqno = next_seqnum(x)
H5: msg_lag(unique_msg(x), u, 0)

=>
C1: u @ [seq: m] = unique_msg(x @ [seq: m])

Prvr-> equality substitute h1

THEOREM not in last output
**** Command Aborted

Prvr-> equality substitute h1
H1 yields
u := unique_msg(x)
Backup point
(* E . D . U . FC . =S . *)

Prvr-> theorem

H1: p,mssg = m
H2: p,seqno = next_seqnum(x)
H3: u#2s = v#2s iff msg_lag(u#2s, v#2s, 0)
H4: msg_lag(unique_msg(x), unique_msg(x), 0)

=>
C1: unique_msg(x) @ [seq: m] = unique_msg(x @ [seq: m])

Prvr-> delete h3,h4
Backup point

Prvr-> equality substitute h1
H1 yields
m := p,mssg
Backup point

Prvr-> expand unique_msg
More than one to expand.
Which do you want to expand? show-choices

1: unique_msg(x)
2: unique_msg(x @ [seq: p])

Which do you want to expand? 2
Proofs of Supporting Lemmas in Gypsy

Backup point
(* E 0 U FC =S 0 =S . E *)

Prvr-> conclusion

C1: unique_msg (x) \& [seq: p.mssg]
    = if p.segno = next.segunm (x)
      then unique_msg (x) \& [seq: p.mssg]
      else unique_msg (x)
    fi

Prvr-> simplify theorem

Prvr-> theorem

C1: true

Prvr-> sProceding

prop_trans_1 proved in theorem prover.

Exec-> prove prop_trans_2

Entering Prover with lemma prop_trans_2

all u : msg_seq,
  all x, y : pkt_seq,
  all p : packet,
  all m : message,
  p.mssg = m \& p.segno = next.segunm (x)
  & proper_transmission (u, x, y, 0)
  -> proper_transmission (u \& [seq: m], x \& [seq: p], y, 1)

H1: p.mssg = m
H2: p.segno = next.segunm (x)
H3: proper_transmission (u, x, y, 0)

->

C1: proper_transmission (u \& [seq: m], x \& [seq: p], y, 1)

Backup point
(* . *)

Prvr-> expand proper_transmission

More than one to expand, which one do you want to expand? all

Backup point
(* . E . *)

Prvr-> simplify theorem

Prvr-> theorem

H1: p.mssg = m
H2: p.segno = next.segunm (x)
H3: size (unique_msg (y)) = size (unique_msg (x))
H4: size (unique_msg (y)) = size (u)
Proofs of Supporting Lemmas in Gypsy

H5: msglag(unique_msg(x), u, 0)
H6: repeats(x)

⇒
C1: msglag(unique_msg(x + [seq: n]), u + [seq: n], 1)
C2: repeats(x + [seq: n])
C3: size(unique_msg(x + [seq: n])) = size(unique_msg(y)) in [0..1]
C4: (size(u) = size(unique_msg(y))) + 1 in [0..1]

Prvr⇒ equality substitute H4
H4 can be solved for:
I1: size(unique_msg(y))
T2: size(u)
which term (or label) do you want to substitute for?
t1

size(unique_msg(y)) := size(u)
OK??

Backup point
(*, E, =S, )

Prvr⇒ use msglag_eq
Backup point
(*, E, =S, U, )

Prvr⇒ hypothesis
H1: u#2s = v#2s iff msglag(u#2s, v#2s, 0)
H2: p,mssg = m
H3: p,seqno = next_seqnum(x)
H4: size(u) = size(unique_msg(x))
H5: msglag(unique_msg(x), u, 0)
H6: repeats(x)

Prvr⇒ forwardchain h1
which way? (left or right)
left
:::forwar chaining gives

unique_msg(x) = u
Backup point
(*, E, =S, U, FC, )

Prvr⇒ theorem
H1: unique_msg(x) = u
H2: u#2s = v#2s iff msglag(u#2s, v#2s, 0)
H3: p,mssg = m
H4: p,seqno = next_seqnum(x)
H5: size(u) = size(unique_msg(x))
Proofs of Supporting Lemmas in Gypsy

H6: msg_lag (unique_msg (x), u, 0)
H7: repeats (x)

⇒
C1: msg_lag (unique_msg (x @ [seq: p]), u @ [seq: p], 1)
C2: repeats (x @ [seq: p])
C3: size (unique_msg (x @ [seq: p])) = size (u) in [0..1]

Prr⇒ equality substitute h1,n3
H1 yields
u := unique_msg (x)
H3 yields
m := p.mssq
Backup point
(* E =S , U = FC =S .)

Prr⇒ theorem

H1: p.seqno = next_seqnum (x)
H2: u#2s = v#2s iff msg_lag (u#2s, v#2s, 0)
H3: msg_lag (unique_msg (x), unique_msg (x), 0)
H4: repeats (x)

⇒
C1: msg_lag (unique_msg (x @ [seq: n]), unique_msg (x) @ [seq: p.mssq], 1)
C2: repeats (x @ [seq: p])
C3: size (unique_msg (x @ [seq: p])) = size (unique_msg (x))
in [0..1]

Prr⇒ expand unique_mssq
More than one to expand, which do you want to expand? show=choices

1: unique_msg (x)
2: unique_msg (x @ [seq: p])

which do you want to expand? Z
Backup point
(* E =S , U = FC =S .)

Prr⇒ simplify theorem

Prr⇒ theorem

H1: p.seqno = next_seqnum (x)
H2: u#2s = v#2s iff msg_lag (u#2s, v#2s, 0)
H3: msg_lag (unique_msg (x), unique_msg (x), 0)
H4: repeats (x)

⇒
C1: msg_lag (unique_msg (x) @ [seq: p.mssq],
    unique_msg (x) @ [seq: p.mssq], 1)
C2: repeats (x @ [seq: p])
Proofs of Supporting Lemmas in Gypsy

\textbf{Prv}\textsubscript{r} \rightarrow \text{retain } h1, h1

\textbf{Backup point}
\begin{align*}
( & \cdot \ E . \ = S \ . \ U \ . \ FC . \ = S \ . \ E \ . \ D \ . ) \\
\end{align*}

\textbf{Prv}\textsubscript{r} \rightarrow \text{Proceeding}

\textbf{Backup point}
\begin{align*}
( & \cdot \ E . \ = S \ . \ U \ . \ FC . \ = S \ . \ E \ . \ D \ . \ 1 \ . ) \\
\end{align*}

\textbf{Prv}\textsubscript{r} \rightarrow \text{theorem}

\begin{align*}
H1: \ p,seqno &= \text{next\_seqnum} (x) \\
H2: \ \text{repeats} (x) \\
\rightarrow \ \\
C1: \ \text{msg\_lag} (\text{unique\_msg} (x) \in \{\text{seq: } p, \text{msg}\}) , \\
\text{unique\_msg} (x) \in \{\text{seq: } p, \text{msg}\} , 1) \\
\end{align*}

\textbf{Prv}\textsubscript{r} \rightarrow \text{expand msg\_lag}

\textbf{Backup point}
\begin{align*}
( & \cdot \ E . \ = S \ . \ U \ . \ FC . \ = S \ . \ E \ . \ D \ . \ 1 \ . \ E \ . ) \\
\end{align*}

\textbf{Prv}\textsubscript{r} \rightarrow \text{use eq\_iss}

\textbf{Typelist equalities}

\begin{align*}
\text{size} (u) &= \text{size} (\text{unique\_msg} (x)) \\
\text{Backup point}
( & \cdot \ E . \ = S \ . \ U \ . \ FC . \ = S \ . \ E \ . \ D \ . \ 1 \ . \ E \ . \ U \ . ) \\
\end{align*}

\textbf{Prv}\textsubscript{r} \rightarrow \text{theorem}

\begin{align*}
H1: \ \text{size} (u) &= \text{size} (\text{unique\_msg} (x)) \\
H2: \ u^\#4s &= v^\#4s \rightarrow \text{initial\_subset} (u^\#4s, v^\#4s) \\
H3: \ p,seqno &= \text{next\_seqnum} (x) \\
H4: \ \text{repeats} (x) \\
\rightarrow \ \\
C1: \ \text{initial\_subset} (\text{unique\_msg} (x) \in \{\text{seq: } p, \text{msg}\} , \\
\text{unique\_msg} (x) \in \{\text{seq: } p, \text{msg}\}) \\
\end{align*}

\textbf{Prv}\textsubscript{r} \rightarrow \text{retain } h2

\textbf{Backup point}
\begin{align*}
( & \cdot \ E . \ = S \ . \ U \ . \ FC . \ = S \ . \ E \ . \ D \ . \ 1 \ . \ E \ . \ U \ . \ D \ . ) \\
\end{align*}

\textbf{Prv}\textsubscript{r} \rightarrow \text{ced}

\begin{align*}
( & \cdot \ E . \ = S \ . \ U \ . \ FC . \ = S \ . \ E \ . \ D \ . \ 1 \ . \ E \ . \ U \ . \ D \ . \ \text{QED}) \\
( & \cdot \ E . \ = S \ . \ U \ . \ FC . \ = S \ . \ E \ . \ D \ . \ 1 \ . \ E \ . \ U \ . \ D \ . \ \text{QED BC}) \\
( & \cdot \ E . \ = S \ . \ U \ . \ FC . \ = S \ . \ E \ . \ D \ . \ 1 \ . \ E \ . \ U \ . \ D \ . \ \text{QED BC}) \\
\text{unique\_msg} (x) \in \{\text{seq: } p, \text{msg}\} &= \text{unique\_msg} (x) \in \{\text{seq: } p, \text{msg}\} \\
\text{Proved} \\
\text{QED} \\
\end{align*}

\textbf{Prv}\textsubscript{r} \rightarrow \text{Proceeding}

\begin{align*}
( & \cdot \ E . \ = S \ . \ U \ . \ FC . \ = S \ . \ E \ . \ D \ . \ 1 \ . ) \\
\end{align*}
Proofs of Supporting Lemmas in Gypsy

\[ \text{msg\_lag} \ (\text{unique\_msg} \ (x) \ : \ [\text{seq} : p, \text{msg}], \ \\ \quad \text{unique\_msg} \ (x) \ : \ [\text{seq} : p, \text{msg}], \ 1) \]

Proved

Privr=> $Proceeding$
Backup point
( , E , =S , U , FC , =S , E , D , 2 )

Privr=> theorem

H1: \( p,\text{seqno} = \text{next\_seqnum} \ (x) \)

H2: repeats (x)

\rightarrow

C1: repeats (x \ & \ [\text{seq} : p])

Privr=> expand repeats
More than one to expand.
Which do you want to expand? show=choices

1: repeats (x)
2: repeats (x \ & \ [\text{seq} : p])

Which do you want to expand? 2
Typelist equalities

size (u) = size (unique\_msg (x))

Backup point
( , E , =S , U , FC , =S , E , D , 2 , E , )

Privr=> theorem

H1: size (u) = size (unique\_msg (x))

H2: null (pkt\_seq) ne x

H3: p,\text{seqno} = \text{next\_seqnum} \ (x) 

H4: repeats (x)

\rightarrow

C1: repeats (x)

C2: p,\text{seqno} ne next\_seqnum \ (x) \rightarrow x[\text{size} \ (x)] = \varnothing

Privr=> back up 2
Restored
Backup point
( , E , =S , U , FC , =S , E , D , 2 )

Privr=> theorem

H1: p,\text{seqno} = \text{next\_seqnum} \ (x) 

H2: repeats (x)

\rightarrow

C1: repeats (x \ & \ [\text{seq} : p])
Proofs of Supporting Lemmas in Gypsy

Prvr-> use last_repeats
Typelist equalities

size (u) = size (unique_msg (x))
Backup point
(* E = S · U · FC = S · E · D · 2 · U *)

Prvr-> qed
(* E = S · U · FC = S · E · D · 2 · U · QED *)
:::(E = S · U · FC = S · E · D · 2 · U · QED BC)
:::..:.::: Ran out of tricks

Prvr-> back up 4
Restored
Backup point
(* E = S · U · FC = S · E · D · 2 · U *)

Prvr-> theorem

H1: size (u) = size (unique_msg (x))
H2: x#2S size (x#2S) = p#2S & repeats (x#2S) & null (pkt_seq)
    ne x#2S
    -> repeats (x#2S & (seq: p#2S))
H3: p,seq = next_seqnum (x)
H4: repeats (x)

=>
C1: repeats (x & (seq: p))

Prvr-> back up 2
Restored
Backup point
(* E = S · U · FC = S · E · D · 2 *)

Prvr-> theorem

H1: p,seq = next_seqnum (x)
H2: repeats (x)

=>
C1: repeats (x & (seq: p))

Prvr-> qed
(* E = S · U · FC = S · E · D · 2 · QED *)
:::(E = S · U · FC = S · E · D · 2 · QED E)
Typelist equalities

size (u) = size (unique_msg (x))
(* E = S · U · FC = S · E · D · 2 · QED E P=>)
(* E = S · U · FC = S · E · D · 2 · QED E P=> 1)
(* E = S · U · FC = S · E · D · 2 · QED E P=> 1)

repeats (x)
Proofs of Supporting Lemmas in Gopsy

Proved

(* E = S . U . FC = S . E . D . 2 . QED & p => 2)
(* E = S . U . FC = S . E . D . 2 . QED & p => 2)

p. seqno ne next-seqnum (x) => x.size (x) = p

Proved

QED

Prv => sProceeding

(* E = S . U . FC = S . E . D . 2 .)

repeats (x a [seq: p])

Proved

Prv => sProceeding

prop-trans-2 proved in theorem prover.

Exec => prove prop-trans-3

Entering Prover with lemma prop-trans-3

all u : msg-seq,
   all x, y : pkt-seq,
   all p : packet,
   x.size (x) = p & proper-transmission (u, x, y, 1)
   & null (pkt-seq) ne x
     => proper-transmission (u, x a [seq: p], y, 1)

H1: x.size (x) = p
H2: proper-transmission (u, x, y, 1)
H3: null (pkt-seq) ne x

=>

Cl: proper-transmission (u, x a [seq: p], y, 1)

Backup point

().

Prv => expand proper-transmission

more than one to expand.

which do you want to expand? all

Backup point

().

Prv => simplify theorem

Prv => theorem

H1: x.size (x) = p
H2: msg-1ag (unique-msg (x), u, 1)
H3: repeats (x)
H4: size (unique-msg (x)) = size (unique-msg (y)) in {0,.1}
H5: size (u) = size (unique-msg (y)) in {0,.1}
H6: null (pkt-seq) ne x

=>
Proofs of Supporting Lemmas in Gypsy

C1: msg-lag (unique-msg (x a [seq: p]), u, 1)
C2: repeats (x a [seq: p])
C3: size (unique-msg (x a [seq: p])) = size (unique-msg (y)) in [0..1]

Prvr-> use last_unique
Backup point
(* E . U .)

Prvr-> hypothesis

H1: x#2s[size (x#2s)] = p#2s & null (pkt-seq) ne x#2s
    => unique-msg (x#2s a [seq: p#2s]) = unique-msg (x#2s)
H2: x[size (x)] = p
H3: msg-lag (unique-msg (x), u, 1)
H4: repeats (x)
H5: size (unique-msg (x)) = size (unique-msg (y)) in [0..1]
H6: size (u) = size (unique-msg (y)) in [0..1]
H7: null (pkt-seq) ne x

Prvr-> forward-chain h1
:Forward chaining gives

unique-msg (x a [seq: p]) = unique-msg (x)
Backup point
(* E . U . FC .)

Prvr-> theorem

H1: unique-msg (x a [seq: p]) = unique-msg (x)
H2: x#2s[size (x#2s)] = p#2s & null (pkt-seq) ne x#2s
    => unique-msg (x#2s a [seq: p#2s]) = unique-msg (x#2s)
H3: x[size (x)] = p
H4: msg-lag (unique-msg (x), u, 1)
H5: repeats (x)
H6: size (unique-msg (x)) = size (unique-msg (y)) in [0..1]
H7: size (u) = size (unique-msg (y)) in [0..1]
H8: null (pkt-seq) ne x

=>

C1: msg-lag (unique-msg (x a [seq: p]), u, 1)
C2: repeats (x a [seq: p])
C3: size (unique-msg (x a [seq: p])) = size (unique-msg (y)) in [0..1]

Prvr-> equality substitute h1
H1 yields

unique-msg (x a [seq: p]) := unique-msg (x)
Backup point
(* E . U . FC . =S .)

Prvr-> simplify theorem
Proofs of Supporting Lemmas in Gypsy

Prvr-> theorem

H1: \( x[\text{size}(x)] = d \)
H2: \( \text{msg}_{-}\text{laq}(\text{unique}_{-}\text{msg}(x), u, l) \)
H3: \( \text{repeats}(x) \)
H4: \( x[\text{size}(x)_2s] = p#s \& \text{null}(\text{pkt}_{-}\text{seq}) \neq x\#s \)
\[ \implies \text{unique}_{-}\text{msg}(x[\text{size}(x)_2s] \& \text{seq}: p#s)) = \text{unique}_{-}\text{msg}(x\#s) \]
H5: \( \text{size}(\text{unique}_{-}\text{msg}(x)) = \text{size}(\text{unique}_{-}\text{msg}(y)) \text{ in } [0..1] \)
H6: \( \text{size}(u) = \text{size}(\text{unique}_{-}\text{msg}(y)) \text{ in } [0..1] \)
H7: \( \text{null}(\text{pkt}_{-}\text{seq}) \neq x \)
\[ \implies \]
C1: \( \text{repeats}(x \& \text{seq}: n) \)

Prvr-> retain h1, h3, h7

Backup point
(* E U FC =S O .)

Prvr-> use last_repeats

Backup point
(* E U FC =S O U .)

Prvr-> qed

(* E U FC =S D U QED)
(* E U FC =S D U QED BC)
(* E U FC =S D U QED BC 1)
(* E U FC =S D U QED BC 1)

x[\text{size}(x)] = d

Proved
(* E U FC =S D U QED BC 2)
(* E U FC =S D U QED BC 2)

repeats(x)

Proved
(* E U FC =S D U QED BC 3)
(* E U FC =S D U QED BC 3)

null(\text{pkt}_{-}\text{seq}) \neq x

Proved
(* E U FC =S D U QED BC)

x[\text{size}(x)] = d \& \text{repeats}(x) \& \text{null}(\text{pkt}_{-}\text{seq}) \neq x

Proved
QED

Prvr-> sProceeding
prop_trans_3 proved in theorem prover.

Exec-> prove prop_trans_4
Entering Prover with lemma prop_trans_4
all u : msg_{-}seq,
all x, y : pkt_{-}seq,
Proofs of Supporting Lemmas in Gymsy

all p : packet,
    proper_transmission (u, x, y, l) & p.seqno ne next_seqnum (y)
    => proper_transmission (u, x, y @ [seq: p], 1)

H1: proper_transmission (u, x, y, l)
H2: p.seqno ne next_seqnum (y)
=> C1: proper_transmission (u, x, y @ [seq: p], 1)

Backup point
(.*

Prvra> expand proper_transmission
More than one to expand.
Which do you want to expand? all
Backup point
(.*

Prvra> simplify theorem

Prvra> theorem

H1: msg-1aq (unique_msg (x), u, l)
H2: repeats (x)
H3: size (unique_msg (x)) = size (unique_msg (y)) in [0..1]
H4: size (u) = size (unique_msg (y)) in [0..1]
H5: p.seqno ne next_seqnum (y)
=> C1: size (unique_msg (x)) = size (unique_msg (y @ [seq: p]))
      in [0..1]
C2: size (u) = size (unique_msg (y @ [seq: p])) in [0..1]

Prvra> expand unique_msg
More than one to expand.
Which do you want to expand? show-choices

1: unique_msg (x)
2: unique_msg (y)
3: unique_msg (y @ [seq: p])

Which do you want to expand? 3
Backup point
(.*

Prvra> simplify theorem

Prvra> theorem

C1: true

Prvra> sProceeding
prop_trans-1 proved in theorem prover.
Proofs of Supporting Lemmas in Gypsy

[Your LOG file is jo011.LOG; Fri May 27 17:17:40]

[Recording initiated at Sat May 1 12:17:42]

prove prop_trans_5
entering Prover with lemma prop_trans_5
all u : msg_seq,
   all x, y : pkt_seq,
   all m : message,
   proper_transmission (u, x, y, 0)
   => proper_transmission (u @ [seq: m], x, y, 1)

H1: proper_transmission (u, x, y, 0)
=>
   C1: proper_transmission (u @ [seq: m], x, y, 1)
Backup point
(.)

prv-> expand proper_transmission
   more than one to expand,
   which do you want to expand? all
   backup point
   (. E .)

prv-> simplify theorem

prv-> theorem

H1: size (unique_msg (y)) = size (unique_msg (x))
H2: size (unique_msg (y)) = size (u)
H3: msg_seq (unique_msg (x), u, 0)
H4: repeats (x)

=>
   C1: msg_seq (unique_msg (x), u @ [seq: m], 1)
   C2: size (unique_msg (x)) = size (unique_msg (y)) in [0..1]
   C3: (size (u) = size (unique_msg (y))) + 1 in [0..1]

prv-> equality substitute h1,h2
H1 can be solved for:
   T1: size (unique_msg (y))
   T2: size (unique_msg (x))
which term (by label) do you want to substitute for?
  t2

   size (unique_msg (x)) := size (unique_msg (y))
OK??
Y
H2 can be solved for:
Proofs of Supporting Lemmas in Gypsy

T1: size (unique\_msg (y))
T2: size (u)
Which term (by label) do you want to substitute for?
t2

size (u) := size (unique\_msg (y))
OK??
y
Backup point
(\* E = S \* )

Prvr-> theorem

H1: msg\_lag (unique\_msg (x), u, 0)
H2: repeats (x)

=>
C1: msg\_lag (unique\_msg (x), u \in \{seq\_m\}, 1)

Prvr-> retain h1
Backup point
(\* E = S , D \* )

Prvr-> use eq\_iss
Backup point
(\* E = S , D , U \* )

Prvr-> hypothesis

H1: u\#8s = v\#8s => initial\_subseq (u\#8s, v\#8s)
H2: msg\_lag (unique\_msg (x), u, 0)

Prvr-> forwardchain h1
::forward chaining gives

initial\_subseq (v\#8s, v\#8s)
Backup point
(\* E = S , D , U , FC \* )

Prvr-> back up 3
Restored
Backup point
(\* E = S , D \* )

Prvr-> use msg\_lag\_eq
Backup point
(\* E = S , D , U \* )

Prvr-> theorem

H1: u\#10s = v\#10s iff msg\_lag (u\#10s, v\#10s, 0)
H2: msg\_lag (unique\_msg (x), u, 0)
Proofs of Supporting Lemmas in Gypsy

\[ C1: \text{msg-lag} (\text{unique-msg} (x), u \in [\text{seq}: m], 1) \]

PrvR-> forward-chain h1
which way? (left or right)
left
::forward chaining gives

\[ \text{unique-msg} (x) = u \]
Backup point
(* E * =S * D * U * FC *)

PrvR-> equality substitute h1
H1 can be solved for:
\[ T1: u \neq 10s \]
\[ T2: \text{msg-lag} (u \neq 10s, v \neq 10s, 0) \]
which term (by label) do you want to substitute for?
t1

\[ u \neq 10s = v \neq 10s := \text{msg-lag} (u \neq 10s, v \neq 10s, 0) \]
OK??
Y
Backup point
(* E * =S * D * U * FC * =S *)

PrvR-> theorem

\[ H1: \text{unique-msg} (x) = u \]
\[ H2: \text{msg-lag} (\text{unique-msg} (x), u, 0) \]
=>
\[ C1: \text{msg-lag} (\text{unique-msg} (x), u \in [\text{seq}: m], 1) \]

PrvR-> equality substitute h1
H1 can be solved for:
\[ T1: \text{unique-msg} (x) \]
\[ T2: u \]
which term (by label) do you want to substitute for?
t1

\[ \text{unique-msg} (x) := u \]
OK??
Y
Backup point
(* E * =S * D * U * FC * =S * =S *)

PrvR-> expand msg-lag
more than one to expand,
which do you want to expand? 2
Backup point
(* E * =S * D * U * FC * =S * =S * E *)
Proofs of Supporting Lemmas in Gypsy

Proofs of supporting lemmas in Gypsy

Prvr=> simplify theorem
Prvr=> theorem

H1: msg_len (u, u, 0)
=>
C1: initial_subseq (u, u & (seq: m))

Prvr=> use eq_iss
Backup point
(* E, =S, D, U, FC, =S, =S, E, U, *)

Prvr=> use iss_app
Backup point
(* E, =S, D, U, FC, =S, =S, F, U, U, *)

Prvr=> qed
(* E, =S, D, U, FC, =S, =S, F, U, U, U, QED *)
::: (* E, =S, D, U, FC, =S, =S, E, U, U, QED BC *)
::: (* E, =S, D, U, FC, =S, =S, E, U, U, QED BC BC *)
(* E, =S, D, U, FC, =S, =S, E, U, U, QED BC BC *)

u = u
Proved
(* E, =S, D, U, FC, =S, =S, E, U, U, QED BC *)

initial_subseq (u, u)
Proved
QED

Prvr=> sProceeding
prop_trans_5 proved in theorem prover.

Exec=> prove prop_trans_6
Entering Prover with lemma prop_trans_6
all u : msg_seq,
    all x, y : pkt_seq,
    proper_transmission (u, x, y, 0) => proper_transmission (u, x, y, 1)

H1: proper_transmission (u, x, y, 0)
=>
C1: proper_transmission (u, x, y, 1)
Backup point
(*, *)

Prvr=> expand proper_transmission
More than one to expand.
Which do you want to expand? all
Backup point
(*, E, *)
Proofs of Supporting Lemmas in Gypsy

Prvr=> simplify theorem

Prvr=> theorem

H1: size (unique_msg (y)) = size (unique_msg (x))
H2: size (unique_msg (y)) = size (u)
H3: msg_lag (unique_msg (x), u, 0)
H4: repeats (x)

=>
C1: msg_lag (unique_msg (x), u, 1)
C2: size (unique_msg (x)) = size (unique_msg (y)) in [u..1]
C3: size (u) = size (unique_msg (y)) in [0..1]

Prvr=> equality substitute h1,h2
H1 can be solved for:
T1: size (unique_msg (y))
T2: size (unique_msg (x))

Which term (by label) do you want to substitute for?
t2

size (unique_msg (x)) := size (unique_msg (y))

OK??

Y

H2 can be solved for:
T1: size (unique_msg (y))
T2: size (u)

Which term (by label) do you want to substitute for?
t2

size (u) := size (unique_msg (y))

OK??

Y

Backup point
( . E . =S . )

Prvr=> theorem

H1: msg_lag (unique_msg (x), u, 0)
H2: repeats (x)

=>
C1: msg_lag (unique_msg (x), u, 1)

Prvr=> retain h1
Backup point
( . E . =S . D . )

Prvr=> expand msg_lag
More than one to expand,
Which do you want to expand? all
Backup point
Proofs of Supporting Lemmas in Gypsy

( E , =S . D , E . )

Prvr=>simplify theorem

Prvr=>theorem

H1: size (unique_msg (x)) = size (u)
H2: initial_subset (unique_msg (x), u)

=>
C1: size (u) = size (unique_msg (x)) in [0, 1]

Prvr=>equality substitute h1

H1 can be solved for:
T1: size (unique_msg (x))
T2: size (u)

Which term (by label) do you want to substitute for?
T1

size (unique_msg (x)) := size (u)

OK??
Y
Backup point
( E , =S . D , E , =S . )

Prvr=>proceeding

prop_trans_6 proved in theorem prover.

Exec=> prove prop_trans_7
Entering prover with lemma prop_trans_7
all u : msg_seq,
   all x, y : pkt_seq,
   all p : packet,
   p Seqno = next_seqnum (y) & unique_msg (x) = u
   & size (unique_msg (x)) = size (unique_msg (y)) + 1
   & proper transmission (u, x, y, 1)

=> proper transmission (u, x, y @ [seq: p], 0)

H1: p Seqno = next seqnum (y)
H2: unique_msg (x) = u
H3: size (unique_msg (x)) = size (unique_msg (y)) + 1
H4: proper transmission (u, x, y, 1)

=>
C1: proper transmission (u, x, y @ [seq: p], 0)

Typelist equalities

size (unique_msg (y)) + 1 = size (unique_msg (x))

Backup point
( )

Prvr=> expand proper transmission

more than one to expand.
Proofs of Supporting Lemmas in Gypsy

which do you want to expand? all
Backup point
(* E *)

Prvr=>simplify theorem

Prvr=>theorem

H1: p,seqno = next_sequnum (y)
H2: unique_msg (x) = u
H3: size (unique_msg (x)) = size (unique_msg (y)) + 1
H4: size (unique_msg (y)) + 1 = size (unique_msg (x))
H5: msg_lag (unique_msg (x), u, 1)
H6: repeats (x)
H7: size (unique_msg (x)) = size (unique_msg (y)) in [0..1]
H8: size (u) = size (unique_msg (y)) in [0..1]

=>

C1: size (unique_msg (y @ [seq: p])) = size (unique_msg (x))
C2: size (unique_msg (y @ [seq: p])) = size (u)
C3: msg_lag (unique_msg (x), u, 0)

Prvr=> expand unique_msg
more than one to expand,
which do you want to expand? show=choices

1: unique_msg (x)
2: unique_msg (y)
3: unique_msg (y @ [seq: p])

which do you want to expand? 3
Backup point
(* E * E *)

Prvr=>simplify theorem

Prvr=>theorem

H1: p,seqno = next_sequnum (y)
H2: unique_msg (x) = u
H3: size (unique_msg (x)) = size (unique_msg (y)) + 1
H4: size (unique_msg (y)) + 1 = size (unique_msg (x))
H5: msg_lag (unique_msg (x), u, 1)
H6: repeats (x)
H7: size (unique_msg (x)) = size (unique_msg (y)) in [0..1]
H8: size (u) = size (unique_msg (y)) in [0..1]

=>

C1: size (unique_msg (y)) + 1 = size (u)
C2: msg_lag (unique_msg (x), u, 0)

Prvr=> equality substitute h2
H2 can be solved for:
F1: unique_msg (x)
which term (my label) do you want to substitute for?

unique_msg (x) := u

OK??
y
Backup point
( . E . E . = S . )

Prvr=> theorem

H1: p.segno = next.segnum (y)
H2: size (unique_msg (y)) + 1 = size (u)
H3: size (u) = size (unique_msg (y)) + 1
H4: msg_lag (u, u, 1)
H5: repeats (x)
H6: size (u) = size (unique_msg (y)) in [0..1]

=>
C1: size (unique_msg (y)) + 1 = size (u)
C2: msg_lag (u, u, 0)

Prvr=> retain

what hypotheses would you like to retain? n4

Backup point

Prvr=> simplify theorem

Prvr=>theorem,

H1: msg_lag (u, u, 1)

=>
C1: size (unique_msg (y)) + 1 = size (u)
C2: msg_lag (u, u, 0)

Prvr=> back up2

**** Illegal command = UP2
Back UP or establish a back up POINT to return to? up

How many levels would you like to back up? 2

Restored
Backup point
( . E . E . = S . )
Proofs of Supporting Lemmas in Gypsy

Prvr→ theorem

H1: p seqno = next seqnum (y)
H2: size (unique msg (y)) + 1 = size (u)
H3: size (u) = size (unique msg (y)) + 1
H4: msg lag (u, u, 1)
H5: repeats (x)
H6: size (u) = size (unique msg (y)) in [0..1]

→
C1: size (unique msg (y)) + 1 = size (u)
C2: msg lag (u, u, 0)

Prvr→ retain h2
Typelist equalities

size (u) = size (unique msg (y)) + 1
Backup point
( E , E = S , D , )

Prvr→ simplify theorem

Prvr→ theorem

H1: size (unique msg (y)) + 1 = size (u)
H2: size (u) = size (unique msg (y)) + 1

→
C1: msg lag (u, u, 0)

Prvr→ use msg lag eq
Backup point
( E , E = S , D , u , )

Prvr→ ded
( E , E = S , D , u , QEVI)

:::
size (unique msg (y)) := size (u) = 1
( E , E = S , D , u , QEVD = S )
( E , E = S , D , u , QEVD = S E )
Ran out of tricks

Prvr→ theorem

H1: u#4s = v#4s iff msg lag (u#4s, v#4s, 0)

→
C1: initial subsec (u, u)

Prvr→ back up 4
Restored
Backup point
( E , E = S , D , u , )

Prvr→ theorem
Proofs of Supporting Lemmas in Gyosa

H1: u#4s = v#4s iff msg_lag (u#4s, v#4s, 0)
H2: size (unique_msg (y)) + 1 = size (u)
H3: size (u) = size (unique_msg (y)) + 1

\implies
C1: msg_lag (u, u, 0)

Provr\to\ use eq_iiss
Backup point
( E, E = S, O, U, U, )

Provr\to\ hypothesis
H1: u#6s = v#6s \implies initial_subseq (u#6s, v#6s)
H2: u#4s = v#4s iff msg_lag (u#4s, v#4s, 0)
H3: size (unique_msg (y)) + 1 = size (u)
H4: size (u) = size (unique_msg (y)) + 1

Provr\to\ retain h1
Backup point
( E, E = S, O, U, U, D, )

Provr\to\ expand msg_lag
Backup point
( E, E = S, O, U, U, D, E, )

Provr\to\ qed
( E, E = S, O, U, U, D, E, Qed)
( E, E = S, O, U, U, D, E, Qed bc)
( E, E = S, O, U, U, D, E, Qed bc)

u = u
Proved
QED

Provr\to\ $Proceeding$
prop_trans_7 proved in theorem prover.

Exec\to\ show status scope$

SCOPE LEMMAS

Waiting to be proved: ABP_1, APP_BIT_NONNULL, APP_MSG_NONNULL,
APP_PKT_NONNULL, BIT_CASES, COMP_EQ, EQ_ISS, EQ_ISS_APP, HIST_SUB,
ISS_APP, ISS_TRANS, INTERPOLATE, LAST_NEXT, LAST_UNIQUE,
LAST_REPEATS, MAIN_LEMMA, MSG_LAG_EQ, NE_NEXT, NEXT_COMP,
NEXT_UNCHANGEAT_EQ, PROP_REC_1, PROP_REC_2, PROP_REC_3, PROP_REC_4,
PROP_REC_5, SUB_APP, SIZE_NONNULL, SUB_SEQUENTIAL, SUB_TO_LAG, SUB_UNCHANGE
Proved: PROP_TRANS_1, PROP_TRANS_2, PROP_TRANS_3, PROP_TRANS_4,
PROP_TRANS_5, PROP_TRANS_5, PROP_TRANS_7

Exec\to\ prove PROP_REC_1
Proofs of Supporting Lemmas in Gypsy

Entering Prover with lemma prop_rec_1
all u : msg_seq,
   all x, y : pkt_seq,
   all p : packet,
   all m : message,
   p.mssa = m & p.segno = next.sequNUM (x)
   & proper.reception (u, x, y, 0)
   => proper.reception (u @ [seq: m], x @ [seq: p], y, 1)

H1: p.mssa = m
H2: p.segno = next.sequNUM (x)
H3: proper.reception (u, x, y, 0)

=>
   C1: proper.reception (u @ [seq: m], x @ [seq: p], y, 1)
Backup point
( .)

Prvr=> expand proper.reception
More than one to expand.
Which do you want to expand? all
Backup point
( . E ,)

Prvr=> simplify theorem

Prvr=> theorem

H1: p.mssa = m
H2: p.segno = next.sequNUM (x)
H3: size (unique_msg (y)) = size (u)
H4: msg.lag (u, unique_msg (x), 0)
H5: y sub x

=>
   C1: msg.lag (u @ [seq: m], unique_msg (x @ [seq: p]), 1)
   C2: (size (u) = size (unique_msg (y))) + 1 in [0,.1]

Prvr=> equality substitute H3
H3 can be solved for:
   T1: size (unique_msg (y))
   T2: size (u)
Which term (by label) do you want to substitute for?
   t1

   size (unique_msg (y)) := size (u)
OK??
Y
Backup point
( . E , =S ,)

Prvr=> theorem
Proofs of Supporting Lemmas in Gypsy

H1: p.mssa = m
H2: p.seqno = next_seqnum (x)
H3: msg_lag (u, unique_msg (x), 0)
H4: y sub x

=> C1: msg_lag (u @ [seq: m], unique_msg (x @ [seq: c]), 1)

prvr=> expand unique_msg
more than one to expand,
which do you want to expand? ?
Backup point
(. E =S E .)

prvr=> use msg_lag_eq
Backup point
(. E =S E U .)

prvr=> theorem
H1: u#2s = v#2s iff msg_lag (u#2s, v#2s, 0)
H2: p.mssa = m
H3: p.seqno = next_seqnum (x)
H4: msg_lag (u, unique_msg (x), 0)
H5: y sub x

=> C1: msg_lag (u @ [seq: m],
    if p.seqno = next_seqnum (x)
    then unique_msg (x) @ [seq: p.mssa]
    else unique_msg (x)
    fi, 1)

prvr=> forwardchain h1
which way? (left or right)
left
:::Forward chaining gives

u = unique_msg (x)
Backup point
(. E =S E U FC .)

prvr=> simplify theorem

prvr=> theorem
H1: p.mssa = m
H2: p.seqno = next_seqnum (x)
H3: unique_msg (x) = u
H4: u#2s = v#2s iff msg_lag (u#2s, v#2s, 0)
H5: msg_lag (u, unique_msg (x), 0)
H6: y sub x

=> C1: msg_lag (u @ [seq: m], unique_msg (x) @ [seq: p.mssa], 1)
Proves of Supporting Lemmas in Gypsy

\texttt{PrvR \rightarrow retain \ h1, h3}
Backup point
( \ast \ E \ = S \ . \ E \ . \ U \ . \ FC \ . \ D \ . )

\texttt{PrvR \rightarrow hypothesis}

\hspace{1cm} H1: \ p.mssg = m
\hspace{1cm} H2: \ unique.mssg \ (x) = u

\texttt{PrvR \rightarrow equality substitute h1,h2}
H1 can be solved for:
\hspace{1cm} T1: \ p.mssg
\hspace{1cm} T2: \ m

which term (by label) do you want to substitute for? t1

\hspace{1cm} p.mssg := m

\texttt{OK??}

\hspace{1cm} y

H2 can be solved for:
\hspace{1cm} T1: \ unique.mssg \ (x)
\hspace{1cm} T2: \ u

which term (by label) do you want to substitute for? t1

\hspace{1cm} unique.mssg \ (x) := u

\texttt{OK??}

\hspace{1cm} y

Backup point
( \ast \ E \ = S \ . \ E \ . \ U \ . \ FC \ . \ D \ . = S \ . )

\texttt{PrvR \rightarrow theorem}

\hspace{1cm} H1: \ true
=>
\hspace{1cm} Cl: \ msg.lag \ (u \ @ [seq: \ m], u \ @ [seq: \ m], 1)

\texttt{PrvR \rightarrow expand msg.lag}
Backup point
( \ast \ E \ = S \ . \ E \ . \ U \ . \ FC \ . \ D \ . = S \ . \ E \ . )

\texttt{PrvR \rightarrow use eq.iss}
Backup point
( \ast \ E \ = S \ . \ E \ . \ U \ . \ FC \ . \ D \ . = S \ . \ E \ . \ U \ . )

\texttt{PrvR \rightarrow qed}
( \ast \ E \ = S \ . \ E \ . \ U \ . \ FC \ . \ D \ . = S \ . \ E \ . \ U \ . \ RED )
( \ast \ E \ = S \ . \ E \ . \ U \ . \ FC \ . \ D \ . = S \ . \ E \ . \ U \ . \ RED \ BC )
( \ast \ FC \ = S \ . \ E \ . \ U \ . \ FC \ . \ D \ . = S \ . \ E \ . \ U \ . \ RED \ BC )
\[ u \oplus [seq: m] = u \oplus [seq: m] \]

Proved
QED

Prvr-> $\$Proceeding
prop_rec_1 proved in theorem prover.

Exec-> $\$prove prop_rec_2
Entering Prover with lemma prop_rec_2
all u : msg_seq,
    all x, y : pkt_seq,
    all p : packet,
    proper_reception (u, x, y, 0)
    $\Rightarrow$ proper_reception (u, x $\oplus$ [seq: p], y, 1)

H1: proper_reception (u, x, y, 0)
    $\Rightarrow$
C1: proper_reception (u, x $\oplus$ [seq: p], y, 1)
Backup point
( . )

Prvr-> $\$expand proper_reception
More than one to expand.
Which do you want to expand?all
Backup point
( . F . )

Prvr-> simolify theorem

Prvr-> theorem

H1: size (unique_msg (y)) = size (u)
H2: msg_lag (u, unique_msg (x), 0)
H3: y sub x
    $\Rightarrow$
C1: msg_lag (u, unique_msg (x $\neq$ [seq: p]), 1)
C2: size (u) = size (unique_msg (y)) in [0..1]

Prvr-> equality substitute h1
H1 can be solved for:
    T1: size (unique_msg (y))
    T2: size (u)
Which term (by label) do you want to substitute for?
    t1

    size (unique_msg (y)) := size (u)
OK??
y
Backup point
( . E . =8 . )
Proofs of Supporting Lemmas in Gypsy

Prvr→ theorem

H1: msg_lag (u, unique_msg (x), 0)
H2: y sub x
⇒
C1: msg_lag (u, unique_msg (x & [seq: p]), 1)

Prvr→ retain h1
Backup point
(. E, =S, D, )

Prvr→ use msg_lag_eq
Backup point
(. E, =S, D, U, )

Prvr→ hypothesis

H1: u#2s = v#2s iff msg_lag (u#2s, v#2s, 0)
H2: msg_lag (u, unique_msg (x), 0)

Prvr→ forwardchain h1
Which way? (left or right)
left
:: Forward chaining gives

u = unique_msg (x)
Backup point
(. E, =S, D, U, FC, )

Prvr→ hypothesis

H1: u = unique_msg (x)
H2: u#2s = v#2s iff msg_lag (u#2s, v#2s, 0)
H3: msg_lag (u, unique_msg (x), 0)

Prvr→ retain h1
Backup point
(. E, =S, D, U, FC, D, )

Prvr→ equality substitute h1
H1 yields
u := unique_msg (x)
Backup point
(. E, =S, D, U, FC, D, =S, )

Prvr→ theorem

H1: true
⇒
C1: msg_lag (unique_msg (x), unique_msg (x & [seq: p]), 1)
Proofs of Supporting Lemmas in Gypsy

Prvr⇒ expand unique_msg

More than one to expand,
Which do you want to expand? show=choices

1: unique_msg (x)
2: unique_msg (x ≠ [seq: p])

Which do you want to expand? 2
Backup point

Prvr⇒ theorem

H1: true
⇒
C1: msg_lag (unique_msg (x),
    if p_seqno = next_seqnum (x)
    then unique_msg (x) ≠ [seq: p.mssg]
    else unique_msg (x)
    fi, 1)

Prvr⇒ cases

Case:
  * p_seqno=next_seqnum(x);
Case:
  * S_done

Attempting CASE1

p_seqno = next_seqnum (x)
Backup point

Prvr⇒ theorem

H1: p_seqno = next_seqnum (x)
⇒
C1: msg_lag (unique_msg (x),
    if p_seqno = next_seqnum (x)
    then unique_msg (x) ≠ [seq: p.mssg]
    else unique_msg (x)
    fi, 1)

Prvr⇒ simplify theorem

Prvr⇒ theorem

H1: p_seqno = next_seqnum (x)
⇒
C1: msg_lag (unique_msg (x), unique_msg (x) ≠ [seq: p.mssg], 1)

Prvr⇒ use eq_lss
Backup point
Proofs of Supporting Lemmas in Gypsy


Prv⇒use iss_app
Backup point

Prv⇒qed

unique_msg (x) = unique_msg (x)
Proved

initial_subseq (unique_msg (x), unique_msg (x))
Proved
QED

Prv⇒sProceeding

p.segno = next_segnum (x)
⇒ msg_lag (unique_msg (x),
   if p.segno = next_segnum (x)
       then unique_msg (x) ∪ {seq: p.mssag}
       else unique_msg (x)
   fi, 1)
Proved
Attempting CASE2

p.segno ne next_segnum (x)
Backup point

Prv⇒ theorem

H1: p.segno ne next_segnum (x)
⇒
C1: msg_lag (unique_msg (x),
   if p.segno = next_segnum (x)
       then unique_msg (x) ∪ {seq: p.mssag}
       else unique_msg (x)
   fi, 1)
Prv⇒ simplify theorem
Prv⇒theorem

H1: p.segno ne next_segnum (x)
Proofs of Supporting Lemmas in Gympy

\[ C1: \text{msg}_\text{lag} (\text{unique}_\text{msg} (x), \text{unique}_\text{msg} (x), 1) \]

**Prv**r→ use eq.1ss
Backup point
(.* E.* =S.* D.* U.* FC.* D.* =S.* E.* CASE2.* U.* )

**Prv**r→ qed
(.* E.* =S.* D.* U.* FC.* D.* =S.* E.* CASE2.* U.* QED)

(.* E.* =S.* D.* U.* FC.* D.* =S.* E.* CASE2.* U.* QED E BC)

(.* E.* =S.* D.* U.* FC.* D.* =S.* E.* CASE2.* U.* QED E BC)

\[ \text{unique}_\text{msg} (x) = \text{unique}_\text{msg} (x) \]

Proved
QED

**Prv**r→ sProceeding
(.* E.* =S.* D.* U.* FC.* D.* =S.* E.* CASE2.* )

\[ \text{p}_\text{seanno} \neq \text{next}_\text{seqnum} (x) \]

\[ \Rightarrow \text{msg}_\text{lag} (\text{unique}_\text{msg} (x), \]

\[ \quad \text{if} \ p\text{.seanno = next}_\text{seqnum} (x) \]

\[ \quad \text{then unique}_\text{msg} (x) \oplus \{ \text{seq: p.mssg} \} \]

\[ \quad \text{else unique}_\text{msg} (x) \]

\[ \text{fi, 1} \]

Proved
Done with cases
prop-rec_2 proved in theorem prover.

**Exec**r→

**** Illegal command - <ESCAPE>

**Exec**r→ prove prop-rec_3
Entering Prover with lemma prop-rec_3
all u : msg_seq,
   all x, y : pkt_seq,
   proper_reception (u, x, y, 0) → proper_reception (u, x, y, 1)

H1: proper_reception (u, x, y, 0)

\[ \Rightarrow \]

C1: proper_reception (u, x, y, 1)
Backup point
(.* )

**Prv**r→ expand proper_reception
More than one to expand,
which do you want to expand? all Backup point
(.* E.* )
Proofs of Supporting Lemmas in Gopsy

Prvr->Simplify theorem

Prvr->Theorem

H1: size (unique_msg (y)) = size (u)
H2: msg_lag (u, unique_msg (x), 0)
H3: y sub x

=>
C1: msg_lag (u, unique_msg (x), 1)
C2: size (u) = size (unique_msg (y)) in [0..1]

Prvr-> Equality substitute h1

H1 can be solved for:
T1: size (unique_msg (y))
T2: size (u)

Which term (by label) do you want to substitute for?
T1

size (unique_msg (y)) := size (u)

OK??

Backup point
(, E , =S , )

Prvr-> Theorem

H1: msg_lag (u, unique_msg (x), 0)
H2: y sub x

=>
C1: msg_lag (u, unique_msg (x), 1)

Prvr-> Expand msg_lag

More than one to expand.
Which do you want to expand? all
Backup point
(, E , =S , E , )

Prvr-> Simplify theorem

Prvr-> Theorem

H1: size (u) = size (unique_msg (x))
H2: initial_subseq (u, unique_msg (x))
H3: y sub x

=>
C1: size (unique_msg (x)) = size (u) in [0..1]

Prvr-> Equality substitute h1

H1 can be solved for:
T1: size (u)
T2: size (unique_msg (x))
Proofs of Supporting Lemmas in Gypsy

Which term (by label) do you want to substitute for?

size (u) := size (unique_msg (x))

ok?

Backup point

E = S, E = S.

Prvr-> $proceeding

prop_rec_3 proved in theorem prover.

Exec->$

**** Illegal command = <ESCAPE>

Exec-> prove prop_rec_4

Entering Prover with lemma prop_rec_4

all u : msg_seq,

all x, y : pkt_seq,

all p : packet,

all m : message,

p.mssa = m \& p.segno = next.segnum (x)

\& p.segno = next.segnum (y) \& proper.reception (u, x, y, 0)

-> proper.reception (u \& [seq: n], x \& [seq: p], y \& [seq: p], 0)

H1: p.mssa = m
H2: p.segno = next.segnum (x)
H3: p.segno = next.segnum (y)
H4: proper.reception (u, x, y, 0)

->

C1: proper.reception (u \& [seq: m], x \& [seq: p], y \& [seq: p], 0)

Backup point

(.)

Prvr-> expand proper.reception

More than one to expand,

which do you want to expand?all

Backup point

(., .)

Prvr-> simplify theorem

Prvr-> theorem

H1: p.mssa = m
H2: p.segno = next.segnum (x)
H3: p.segno = next.segnum (y)
H4: size (unique_msg (y)) = size (u)
H5: msg_seg (u, unique_msg (x), 0)
H6: y sub x
Proofs of Supporting Lemmas in Gypsy

\[ \Rightarrow \]
\( C1: \text{size } (\text{unique}_\text{msg} (y \in [\text{seq}: p])) = \text{size } (u) + 1 \)
\( C2: \text{msg}_\text{lag} (u \in [\text{seq}: p], \text{unique}_\text{msg} (x \in [\text{seq}: p]), 0) \)
\( C3: y \in [\text{seq}: p] \text{ sub } x \in [\text{seq}: p] \)

Prvrv expand unique_msg

More than one to expand.
Which do you want to expand? show=choices

1: unique_msg (y)
2: unique_msg (x)
3: unique_msg (y \in [\text{seq}: p])
4: unique_msg (x \in [\text{seq}: p])

Which do you want to expand? 3,4
Backup point
(*, E, E, *)

Prvrv simplify theorem

Prvrv theorem

H1: p.mssg = m
H2: p.seqno = next_seqnum (x)
H3: p.seqno = next_seqnum (y)
H4: size (unique_msg (y)) = size (u)
H5: msg_lag (u, unique_msg (x), 0)
H6: y sub x

\[ \Rightarrow \]
\( C1: \text{msg}_\text{lag} (u \in [\text{seq}: m], \text{unique}_\text{msg} (x \in [\text{seq}: m.mssg]), 0) \)
\( C2: y \in [\text{seq}: p] \text{ sub } x \in [\text{seq}: p] \)

Prvrv use eq-ass
Backup point
(*, E, E, U, *)

Prvrv hypothesis

H1: u#2s = v#2s \Rightarrow \text{initial_subseq } (u#2s, v#2s)
H2: p.mssg = m
H3: p.seqno = next_seqnum (x)
H4: p.seqno = next_seqnum (y)
H5: size (unique_msg (y)) = size (u)
H6: msg_lag (u, unique_msg (x), 0)
H7: y sub x

Prvrv forwardchain h1
::: Forward chaining gives

\text{initial_subseq } (p, \text{seqno}, \text{next_seqnum } (x))
Backup point
(*, E, E, U, FC, *)
Proofs of Supporting Lemmas in Gopsy

Prvr=> back up 2
Restored
Backup point
(* E, E, U, )

Prvr=> back up 2
Restored
Backup point
(* E, E, )

Prvr=> use msg_lag_eq
Backup point
(* E, E, U, )

Prvr=> hypothesis

H1: u#4s = v#4s iff msg_lag (u#4s, v#4s, 0)
H2: p.mssg = m
H3: p.segno = next_segnum (x)
H4: p.segno = next_segnum (y)
H5: size (unique_msg (y)) = size (u)
H6: msg_lag (u, unique_msg (x), 0)
H7: y sub x

Prvr=> forwardchain h1
Which way? (left or right)
left
- - - - -: Forward chaining gives

u = unique_msg (x)
Backup point
(* E, E, U, FC, )

Prvr=> hypothesis

H1: u = unique_msg (x)
H2: u#4s = v#4s iff msg_lag (u#4s, v#4s, 0)
H3: p.mssg = m
H4: p.segno = next_segnum (x)
H5: p.segno = next_segnum (y)
H6: size (unique_msg (y)) = size (u)
H7: msg_lag (u, unique_msg (x), 0)
H8: y sub x

Prvr=> equality substitute h1
H1 can be solved for:
T1: u
T2: unique_msg (x)
Which term (by label) do you want to substitute for?
t2
Proofs of Supporting Lemmas in Gymy

unique_msg (x) := u
OK??

 Tipoist equalities

size (u) = size (unique_msg (y))
Backup point
(. E . E . U . FC . = S .)

Prvr->theorem

H1: size (u) = size (unique_msg (y))
H2: p.mssg = m
H3: p.seqno = next_seqnum (x)
H4: p.seqno = next_seqnum (y)
H5: size (unique_msg (y)) = size (u)
H6: u # 4S = v # 4S iff msg_lag (u # 4S, v # 4S, 0)
H7: msg_lag (u, u, 0)
H8: y sub x

=>

C1: size (if p.seqno = next_seqnum (y)
    then unique_msg (y) @ [seq: p.mssg]
    else unique_msg (y)
    fi)
    = size (u) + 1
C2: msg_lag (u @ [seq: m],
    if p.seqno = next_seqnum (x)
    then u @ [seq: p.mssg]
    else u
    fi, @)
C3: y @ [seq: p] sub x @ [seq: p]

Prvr-> simplify theorem

Prvr->theorem

H1: p.mssg = m
H2: p.seqno = next_seqnum (x)
H3: p.seqno = next_seqnum (y)
H4: size (unique_msg (y)) = size (u)
H5: size (u) = size (unique_msg (y))
H6: u # 4S = v # 4S iff msg_lag (u # 4S, v # 4S, 0)
H7: msg_lag (u, u, 0)
H8: y sub x

=>

C1: msg_lag (u @ [seq: m], u @ [seq: p.mssg], 0)
C2: v @ [seq: p] sub x @ [seq: p]

Prvr-> equality substitute h1
H1 can be solved for:
T1: p.mssg
Proofs of Supporting Lemmas in Gypsy

T2: m
Which term (by label) do you want to substitute for?

p, mssq := m

OK??

y
Backup point

Prvr => hypothesis

H1: p, seqno = next_seqnum (x)
H2: p, seqno = next_seqnum (y)
H3: size (unique_msg (y)) = size (u)
H4: size (u) = size (unique_msg (y))
H5: u # s = v # s iff msg_lag (u # s, v # s, 0)
H6: msg_lag (u, u, 0)
H7: y sub x

Prvr => retain h6, h7
Backup point

Prvr => sProceeding
Backup point

Prvr => theorem

H1: msg_lag (u, u, 0)
H2: y sub x

=>
C1: msg_lag (u a [seq: m], u a [seq: m], 0)

Prvr => use eq_msg_app
Typelist equalities

size (u) = size (unique_msg (y))
Backup point

Prvr => expand msg_lag
More than one to expand,
Which do you want to expand? all
Backup point

Prvr => qed
Proofs of Supporting Lemmas in GYSY

\((\cdot E \cdot E \cdot U \cdot FC \cdot = S \cdot = S \cdot D \cdot 1 \cdot U \cdot \ell \cdot \text{QED SC})\)

\(u = u\)
Proved
QED

Prvr→ sProceeding

\((\cdot E \cdot E \cdot U \cdot FC \cdot = S \cdot = S \cdot D \cdot 1 \cdot )\)

msg_seq (u @ [seq: m], u @ [seq: m], 0)
Proved

Prvr→ sProceeding
Backup point

\((\cdot E \cdot U \cdot FC \cdot = S \cdot = S \cdot D \cdot 2 \cdot )\)

Prvr→ theorem

\(H1: \text{msg_seq} (u, u, 0)\)
\(H2: y \supset x\)

⇒
\(C1: y @ [seq: p] \supset x @ [seq: p]\)

Prvr→ retain H2
Backup point

\((\cdot E \cdot U \cdot FC \cdot = S \cdot = S \cdot D \cdot 2 \cdot D \cdot )\)

Prvr→ simplify theorem

Prvr→ theorem

\(H1: y \supset x\)

⇒
\(C1: y @ [seq: p] \supset x @ [seq: p]\)

Prvr→ use sub_app
Typelist equalities

\(\text{size (u)} = \text{size (unique_msg (y))}\)
Backup point

\((\cdot E \cdot E \cdot U \cdot FC \cdot = S \cdot = S \cdot D \cdot 2 \cdot 0 \cdot J \cdot )\)

Prvr→ qed

\((\cdot E \cdot E \cdot U \cdot FC \cdot = S \cdot = S \cdot D \cdot 2 \cdot D \cdot J \cdot \text{QED})\)

\(\text{size (u)} \supset x\)
Proved
QED

Prvr→ sProceeding
Proofs of Supporting Lemmas in Gypsy

(* E E U FC S S U 2 *)

\[ y \in \text{seq} \; \alpha \; \text{sub} \; x \in \text{seq} \; \alpha \]
Proved

\textbf{Prvr-> $\text{proceeding}$}
\textbf{prop-rec-4 proved in theorem prover.}

\textbf{Exec-> $\text{prove prop-rec-5}$}
\textbf{Entering Prover with lemma prop-rec-5}
\textbf{all u : msg-seq,}
\textbf{all x, y : pkt-seq,}
\textbf{all p : packet,}
\textbf{\textcolor{red}{proper-reception}}(u, x, y, 0) \& p.seqno ne next_seqnum(x) \& p.seqno ne next_seqnum(y)
\Rightarrow \text{proper-reception}(u, x \in \text{seq} \; \alpha, y \in \text{seq} \; \alpha, 0)

\textbf{H1: proper-reception}(u, x, y, 0)
\textbf{H2: p.seqno ne next_seqnum}(x)
\textbf{H3: p.seqno ne next_seqnum}(y)
\Rightarrow \textbf{C1: proper-reception}(u, x \in \text{seq} \; \alpha, y \in \text{seq} \; \alpha, 0)
\textbf{Backup point}
(* *)

\textbf{Prvr-> $\text{expand proper-reception}$}
\textbf{more than one to expand.}
\textbf{which do you want to expand? all}
\textbf{Backup point}
(* *)

\textbf{Prvr-> $\text{simplify theorem}$}

\textbf{Prvr-> theorem}

\textbf{H1: size(unique_msg(y)) = size(u)}
\textbf{H2: msg_tag(u, unique_msg(x), 0)}
\textbf{H3: p.seqno ne next_seqnum}(x)
\textbf{H4: p.seqno ne next_seqnum}(y)
\textbf{H5: y sub x}
\Rightarrow
\textbf{C1: size(unique_msg(y \in \text{seq} \; \alpha)) = size(u)}
\textbf{C2: msg_tag(u, unique_msg(x \in \text{seq} \; \alpha), 0)}
\textbf{C3: y \in \text{seq} \; \alpha sub x \in \text{seq} \; \alpha}

\textbf{Prvr-> $\text{expand unique_msg}$}
\textbf{more than one to expand.}
\textbf{which do you want to expand? show-choices}

1: unique_msg(y)
2: unique_msg(x)
3: unique_msg(y \in \text{seq} \; \alpha)
Proofs of Supporting Lemmas in Gypsy

4: unique_msg (x * [seq: p])

which do you want to expand? 3,4
Backup point
(* E . E . *)

prvr-> simplify theorem

prvr-> theorem

H1: size (unique_msg (y)) = size (u)
H2: msg_lag (u, unique_msg (x), 0)
H3: p, seqno ne next_seqnum (x)
H4: p, seqno ne next_seqnum (y)
H5: y sub x

=>

Cl: y @ [seq: p] sub x @ [seq: p]

prvr-> retain h5
Backup point
(* E . E . D . *)

prvr-> use sub-app
Backup point
(* E . E . D . U . *)

prvr-> qed

y sub x
Proved
QED

prvr-> $Proceeding
prop-rec-5 proved in theorem prover.

exec-> show status scope $
Proofs of Supporting Lemmas in Gypsy

Exec-> save abp.dmp

File ABP.DMP already exists. Rewrite it? -> y

Executing Prover with lemma abp-1
all u, v : msg-seq,
  all w, x, y, z : pkt-seq,
  proper-reception (v, x, y, l) & proper-transmission (u, w, z, l)
  & x sub w & z sub y
  => msg-lag (v, u, l)

H1: proper-reception (v, x, y, l)
H2: proper-transmission (u, w, z, l)
H3: x sub w
H4: z sub y

=>
C1: msg-lag (v, u, l)
Backup point
(*)

Prvr-> expand proper-reception
Backup point
(* E *)

Prvr-> expand proper-transmission
Backup point
(* E * E *)

Prvr-> simplify theorem

Prvr-> theorem

H1: msg-lag (unique-msg (v), u, l)
H2: msg-lag (v, unique-msg (x), l)
H3: repeats (v)
H4: size (v) = size (unique-msg (y)) in [0..1]
H5: size (unique-msg (w)) = size (unique-msg (z)) in [0..1]
H6: size (u) = size (unique-msg (z)) in [0..1]
H7: x sub w
H8: y sub x
H9: z sub y
=>
C1: msg-lag (v, u, l)
Proofs of Supporting Lemmas in Gymsy

Prvr-> use nchanges_unique
Backup point
(* E E u *)

Prvr-> use nchanges_unique
Backup point
(* E E u u u *)

Prvr-> hypothesis

H1: size (unique_msg (x#4S)) = nchanges (senums (x#4S))
H2: size (unique_msg (x#2S)) = nchanges (senums (x#2S))
H3: msg_lag (unique_msg (#), u, 1)
H4: msg_lag (v, unique_msg (x), 1)
H5: repeats (#)
H6: size (v) = size (unique_msg (y)) in [0..1]
H7: size (unique_msg (w)) = size (unique_msg (z)) in [0..1]
H8: size (x) = size (unique_msg (z)) in [0..1]
H9: x sub w
H10: y sub x
H11: z sub w

Prvr-> put

For what?* x#4S;
Put what?* #;

For what?* x#2S;
Put what?* z;

For what?* $done

Backup point
(* E E u u u PUT *)

Prvr-> hypothesis

H1: size (unique_msg (z)) = nchanges (senums (z))
H2: size (unique_msg (w)) = nchanges (senums (w))
H3: z sub y
H4: y sub x
H5: x sub w
H6: size (u) = size (unique_msg (z)) in [0..1]
H7: size (unique_msg (w)) = size (unique_msg (z)) in [0..1]
H8: size (y) = size (unique_msg (y)) in [0..1]
H9: repeats (#)
H10: msg_lag (v, unique_msg (x), 1)
H11: msg_lag (unique_msg (#), u, 1)

Prvr-> equality substitute h1,h2
H1 yields
size (unique_msg (z)) := nchanges (senums (z))
Proofs of Supporting Lemmas in Gypsy

H2 yields
  size (unique_msg (w)) := nchanges (seqnums (w))
Backup point
(* E E U U PUT =S *)

Prv => theorem

H1: msg-lag (unique_msg (w), u, 1)
H2: msg-lag (v, unique_msg (x), 1)
H3: repeats (w)
H4: nchanges (seqnums (v)) = nchanges (seqnums (z))
    in [0,1]
H5: size (u) = nchanges (seqnums (z)) in [0,1]
H6: size (v) = size (unique_msg (v)) in [0,1]
H7: x sub w
H8: y sub x
H9: z sub y

=>
C1: msg-lag (v, u, 1)

Prv => claim
New goal:
* z sub x;
Defer proof of claim? y

Backup point
(* E E U PUT =S *)

Prv => use sub-seqnum
Backup point
(* E E U PUT =S U *)

Prv => hypothesis

H1: x#s sub y#7s => seqnums (x#6s) sub seqnums (y#7s)
H2: z sub x
H3: msg-lag (unique_msg (w), u, 1)
H4: msg-lag (v, unique_msg (x), 1)
H5: repeats (w)
H6: nchanges (seqnums (w)) = nchanges (seqnums (z))
    in [0,1]
H7: size (u) = nchanges (seqnums (z)) in [0,1]
H8: size (v) = size (unique_msg (y)) in [0,1]
H9: x sub w
H10: y sub x
H11: z sub y

Prv => forwardchain H1
::Forward chaining gives

  seqnums (z) sub seqnums (x)
Backup point
Proofs of Supporting Lemmas in Gypsy


Prvr→ use main_lemma
Backup point

Prvr→ theorem

H1:  
repeat (y#48) 
& nchanges (senums (y#48)) = nchanges (s#28) 
in [0..1] & s#28 sub senums (x#38) & x#38 sub y#48s 
→ msg_lag (unique_msg (x#38), unique_msg (y#48), 1)
H2:  senums (z) sub senums (x)
H3:  x#38s sub y#28 → senums (x#38s) sub senums (y#28)
H4:  z sub x
H5:  msg_lag (unique_msg (w), u, l)
H6:  msg_lag (v, unique_msg (x), l)
H7:  repeat (w)
H8:  nchanges (senums (w)) = nchanges (senums (z)) 
in [0..1]
H9:  size (u) = nchanges (senums (z)) in [0..1]
H10: size (v) = size (unique_msg (y)) in [0..1]
H11: x sub w
H12: y sub x
H13: z sub v

→ C1: msg_lag (v, u, l)

Prvr→ forwardchain h1

msg_lag (unique_msg (x), unique_msg (w), 1)
Backup point

Prvr→ expand msg_lag
More than one to expand, 
Which do you want to expand? all
Backup point

Prvr→ theorem

H1:  initial_subseq (unique_msg (x), unique_msg (w)) 
& size (unique_msg (w)) = size (unique_msg (x)) in [0..1]
H2:  repeats (y#48) 
& nchanges (senums (y#13)) = nchanges (s#28) 
in [0..1] & s#28 sub senums (x#38) & x#38 sub y#48s 
→ initial_subseq (unique_msg (x#38), unique_msg (y#48)) 
& size (unique_msg (y#48)) = size (unique_msg (x#38)) 
in [0..1]
H3:  senums (z) sub senums (x)
Proofs of Supporting Lemmas in Gypsy

\[ H_4: \text{sub } y \#2s \implies \text{sequences (}x \#6s\text{) sub sequences (}y \#2s\text{)} \]
\[ H_5: z \text{ sub x} \]

\[ H_6: \text{initial-subseq (unique-seq (}w\text{), u)} \]
& size (u) = size (unique-seq (}w\text{)) in \{0,1\} \]

\[ H_7: \text{initial-subseq (v, unique-seq (}x\text{))} \]
& size (unique-seq (}x\text{)) = size (v) in \{0,1\} \]

\[ H_8: \text{repeats (}w\text{)} \]

\[ H_9: \text{ncchanges (sequences (}w\text{)) = ncchanges (sequences (}z\text{))} \]
in \{0,1\} \]

\[ H_{10}: \text{size (u) = ncchanges (sequences (}z\text{)) in \{0,1\}} \]

\[ H_{11}: \text{size (v) = size (unique-seq (}y\text{)) in \{0,1\}} \]

\[ H_{12}: x \text{ sub w} \]

\[ H_{13}: y \text{ sub } x^Bz^B \]

\[ H_{14}: z \text{ sub y} \]

\[ \implies \]

\[ C_1: \text{initial-subseq (}v, u\text{)} \]

\[ C_2: \text{size (}u\text{) = size (}v\text{) in \{0,1\}} \]

Prvr\[\rightarrow\] delete \(n_2, n_4\)

Backup point

(., E, E, U, U, PUT, =S, U, FC, U, FC, E, D, .)

Prvr\[\rightarrow\] simplify theorem

Prvr\[\rightarrow\] theorem

\[ H_1: \text{initial-subseq (unique-seq (}v\text{), u)} \]

\[ H_2: \text{initial-subseq (unique-seq (}x\text{), unique-seq (}w\text{))} \]

\[ H_3: \text{initial-subseq (v, unique-seq (}x\text{))} \]

\[ H_4: x \#6s \text{ sub } y \#2s \implies \text{sequences (}x \#6s\text{) sub sequences (}y \#2s\text{)} \]

\[ H_5: \text{ncchanges (sequences (}w\text{)) = ncchanges (sequences (}z\text{))} \]
in \{0,1\} \]

\[ H_6: \text{size (u) = ncchanges (sequences (}z\text{)) in \{0,1\}} \]

\[ H_7: \text{size (u) = size (unique-seq (}w\text{)) in \{0,1\}} \]

\[ H_8: \text{size (unique-seq (}w\text{)) = size (unique-seq (}x\text{)) in \{0,1\}} \]

\[ H_9: \text{size (}v\text{) = size (unique-seq (}y\text{)) in \{0,1\}} \]

\[ H_{10}: \text{size (unique-seq (}x\text{)) = size (}v\text{) in \{0,1\}} \]

\[ H_{11}: \text{sequences (}z\text{) sub sequences (}x\text{)} \]

\[ H_{12}: x \text{ sub w} \]

\[ H_{13}: y \text{ sub } x \]

\[ H_{14}: z \text{ sub } x \]

\[ H_{15}: z \text{ sub y} \]

\[ \implies \]

\[ C_1: \text{initial-subseq (}v, u\text{)} \]

\[ C_2: \text{size (}u\text{) = size (}v\text{) in \{0,1\}} \]

Prvr\[\rightarrow\] sProceeding

Backup point

(., E, E, U, U, PUT, =S, U, FC, J, FC, E, D, 1.)

Prvr\[\rightarrow\] retain \(n_1,n_2,n_3\)

Backup point
Proofs of Supporting Lemmas in Gypsy

(• E • E • U • U • PUT = S • U • FC • U • FC • E • D • 1 • D • U • U •)

Prv⇒ use iss_trans
Backup point
(• E • E • U • U • PUT = S • U • FC • U • FC • E • D • 1 • D • U • U •)

Prv⇒ use iss_trans
Backup point
(• E • E • U • U • PUT = S • U • FC • U • FC • E • D • 1 • D • U • U •)

Prv⇒ theorem

H1: initial_subseq (u#48, v#48) & initial_subseq (v#48, w#48)
⇒ initial_subseq (u#48, w#48)

H2: initial_subseq (u#28, v#28) & initial_subseq (v#28, w#28)
⇒ initial_subseq (v#28, w#28)

H3: initial_subseq (unique_msg (v), u)
H4: initial_subseq (unique_msg (x), unique_msg (w))

⇒ C1: initial_subseq (v, u)

Prv⇒ qed
(• E • E • U • U • PUT = S • U • FC • U • FC • E • D • 1 • D • U • U •)
QED

H1: initial_subseq (v, v#48)
Proved
(• E • E • U • U • PUT = S • U • FC • U • FC • E • D • 1 • D • U • U •)
QED B
C 2

H2: initial_subseq (u#28, v#28)
Proved
(• E • E • U • U • PUT = S • U • FC • U • FC • E • D • 1 • D • U • U •)
QED B
C 2

H3: initial_subseq (unique_msg (x), v#28)
Proved
(• E • E • U • U • PUT = S • U • FC • U • FC • E • D • 1 • D • U • U •)
QED B
C 2 dC 2

H4: initial_subseq (unique_msg (x), u)
Proved
Proofs of Supporting Lemmas in Gypsy

13

\[ \text{PUT } = S \cdot U \cdot FC \cdot U \cdot FC \cdot E \cdot D \cdot 1 \cdot D \cdot U \cdot U \cdot QED \]

(C 2 8 C)

\[ \text{initial-subseq (unique-msg (x), v \# 2 S) } \& \text{ initial-subseq (v \# 2 S, u)} \]

Proved

\[ \text{PUT } = S \cdot U \cdot FC \cdot U \cdot FC \cdot E \cdot D \cdot 1 \cdot D \cdot U \cdot U \cdot QED \]

(C 2 8 C)

\[ \text{initial-subseq (unique-msg (x), u)} \]

Proved

\[ \text{PUT } = S \cdot U \cdot FC \cdot U \cdot FC \cdot E \cdot D \cdot 1 \cdot D \cdot U \cdot U \cdot QED \]

(C 2 8 C)

\[ \text{initial-subseq (v, v \# 4 S) } \& \text{ initial-subseq (v \# 4 S, u)} \]

Proved

QED

Prv\(r \rightarrow\) sProceeding

\[ \text{PUT } = S \cdot U \cdot FC \cdot U \cdot FC \cdot E \cdot D \cdot 1 \cdot D \cdot U \cdot U \]

initial-subseq (v, u)

Proved

Prv\(r \rightarrow\) sProceeding

Backup point

\[ \text{PUT } = S \cdot U \cdot FC \cdot U \cdot FC \cdot E \cdot D \cdot 2 \cdot D \]

Prv\(r \rightarrow\) theorem

H1: initial-subseq (unique-msg (x), u)
H2: initial-subseq (unique-msg (x), unique-msg (\(w\)))
H3: initial-subseq (v, unique-msg (x))
H4: x \# 6 S sub y \# 2 S \rightarrow \text{sequums (x \# 6 S) sub sequums (y \# 2 S)}

H5: \text{ nchanges (sequums (w)) } = \text{ nchanges (sequums (z))}

in \([0,1]\)
H6: size (u) = nchanges (sequums (z)) in \([0,1]\)
H7: size (u) = size (unique-msg (w)) in \([0,1]\)
H8: size (unique-msg (x)) = size (unique-msg (x)) in \([0,1]\)
H9: size (v) = size (unique-msg (y)) in \([0,1]\)
H10: size (unique-msg (x)) = size (v) in \([0,1]\)
H11: sequums (z) sub sequums (x)
H12: x sub w
H13: y sub x
H14: z sub x
H15: z sub y

\[ \Rightarrow \]
C1: size (u) = size (v) in \([0,1]\)

Prv\(r \rightarrow\) delete h1, h2, h3, h4, h11, h12, h13, h14

Backup point

\[ \text{PUT } = S \cdot U \cdot FC \cdot U \cdot FC \cdot E \cdot D \cdot 2 \cdot D \]
Proofs of Supporting Lemmas in Gypsy

Prv=> theorem

H1: nchanges (seqnums (w)) = nchanges (seqnums (z)) in [0..1]
H2: size (u) = nchanges (seqnums (z)) in [0..1]
H3: size (u) = size (unique_msq (w)) in [0..1]
H4: size (unique_msq (w)) = size (unique_msq (x)) in [0..1]
H5: size (y) = size (unique_msq (y)) in [0..1]
H6: size (unique_msq (x)) = size (v) in [0..1]
H7: z sub y

=> C1: size (u) = size (v) in [0..1]

Prv=> use sub_seqnums
Backup point

Prv=> use sub_nchanges
Backup point

Prv=> hypothesis

H1: s#4s sub t#2s => nchanges (s#4s) le nchanges (t#2s)
H2: x#10s sub y#6s => seqnums (x#10s) sub seqnums (y#6s)
H3: nchanges (seqnums (w)) = nchanges (seqnums (z)) in [0..1]
H4: size (u) = nchanges (seqnums (z)) in [0..1]
H5: size (u) = size (unique_msq (w)) in [0..1]
H6: size (unique_msq (w)) = size (unique_msq (x)) in [0..1]
H7: size (y) = size (unique_msq (y)) in [0..1]
H8: size (unique_msq (x)) = size (v) in [0..1]
H9: z sub y

Prv=> forwardchain n2

Forward chaining gives

seqnums (z) sub seqnums (y)

Backup point

Prv=>nynypothesis

H1: seqnums (z) sub seqnums (y)
H2: s#4s sub t#2s => nchanges (s#4s) le nchanges (t#2s)
H3: x#10s sub y#6s => seqnums (x#10s) sub seqnums (y#6s)
H4: nchanges (seqnums (x)) = nchanges (seqnums (z)) in [0..1]
H5: size (u) = nchanges (seqnums (z)) in [0..1]
H6: size (u) = size (unique_msq (w)) in [0..1]
H7: size (unique_msq (w)) = size (unique_msq (x)) in [0..1]
H8: size (v) = size (unique_msq (y)) in [0..1]
Proofs of Supporting Lemmas in Gypsy

H9: size (unique_msg (x)) = size (v) in [0..1]
H10: z sub y

Prvr-> hypothesis

H1: sequums (z) sub sequums (y)
H2: s#4s sub t#2s -> nchanges (s#4s) le nchanges (t#2s)
H3: x#10s sub y#6s -> sequums (x#10s) sub sequums (y#6s)
H4: nchanges (sequums (x)) = nchanges (sequums (z)) in [0..1]
H5: size (u) = nchanges (sequums (z)) in [0..1]
H6: size (u) = size (unique_msg (x)) in [0..1]
H7: size (unique_msg (x)) = size (unique_msg (x)) in [0..1]
H8: size (v) = size (unique_msg (v)) in [0..1]
H9: size (unique_msg (x)) = size (v) in [0..1]
H10: z sub y

Prvr-> forwardchain h2

Forward chaining gives

nchanges (sequums (z)) le nchanges (sequums (y))

Backup point

Prvr-> hypothesis

H1: nchanges (sequums (z)) le nchanges (sequums (y))
H2: sequums (z) sub sequums (y)
H3: s#4s sub t#2s -> nchanges (s#4s) le nchanges (t#2s)
H4: x#10s sub y#6s -> sequums (x#10s) sub sequums (y#6s)
H5: nchanges (sequums (x)) = nchanges (sequums (z)) in [0..1]
H6: size (u) = nchanges (sequums (z)) in [0..1]
H7: size (u) = size (unique_msg (x)) in [0..1]
H8: size (unique_msg (x)) = size (unique_msg (x)) in [0..1]
H9: size (v) = size (unique_msg (v)) in [0..1]
H10: size (unique_msg (x)) = size (v) in [0..1]
H11: z sub y

Prvr-> delete h2,h3,h4,h11

Backup point

Prvr-> simplify theorem

Prvr-> theorem

H1: nchanges (sequums (x)) = nchanges (sequums (z)) in [0..1]
H2: size (u) = nchanges (sequums (z)) in [0..1]
Proofs of Supporting Lemmas in Gyosy

H3: size (u) = size (unique_msg (w)) in [0, 1]
H4: size (unique_msg (w)) = size (unique_msg (x)) in [0, 1]
H5: size (v) = size (unique_msg (v)) in [0, 1]
H6: size (unique_msg (x)) = size (v) in [0, 1]
H7: nchanges (seqnums (z)) ≠ nchanges (seqnums (y))

=>
C1: size (u) = size (v) in [0, 1]

Prvr=> use nchanges_unique
Backup point
(* E * E * U * U * PUT = $ * U * FC * U * FC * E * D * 2 * D * U * U * FC * FC * D * U * PUT *)

Prvr=>out

For what? * sdone

Prvr=> hypothesis

H1: size (unique_msg (x # 128)) = nchanges (seqnums (x # 128))
H2: nchanges (seqnums (w)) = nchanges (seqnums (z))
in [0, 1]
H3: size (u) = nchanges (seqnums (z)) in [0, 1]
H4: size (u) = size (unique_msg (w)) in [0, 1]
H5: size (unique_msg (w)) = size (unique_msg (x)) in [0, 1]
H6: size (v) = size (unique_msg (y)) in [0, 1]
H7: size (unique_msg (x)) = size (v) in [0, 1]
H8: nchanges (seqnums (z)) ≠ nchanges (seqnums (y))

Prvr=> out

For what? * x # 128;
Put what? * y;

For what? * sdone

Backup point
(* E * E * U * U * PUT = $ * U * FC * U * FC * E * D * 2 * D * U * U * FC * FC * D * U * PUT *)

Prvr=> equality substitute n1
H1 no longer part of current HYPOTHESES.

### Command Aborted

Prvr=> hypothesis

H1: size (unique_msg (y)) = nchanges (seqnums (y))
H2: nchanges (seqnums (z)) ≠ nchanges (seqnums (y))
H3: size (unique_msg (x)) = size (v) in [0, 1]
H4: size (v) = size (unique_msg (y)) in [0, 1]
Proofs of Supporting Lemmas in Gypsy

H5: size (unique\_msg (w)) = size (unique\_msg (x)) in [0..1]
H6: size (u) = size (unique\_msg (w)) in [0..1]
H7: size (v) = nhcanges (seqnums (z)) in [0..1]
H8: nhcanges (seqnums (w)) = nhcanges (seqnums (z))
in [0..1]

Prvr\(\rightarrow\) equality substitute h1
H1 can be solved for:
T1: nhcanges (seqnums (y))
T2: size (unique\_msg (y))

which term (by label) do you want to substitute for? t1

nhcanges (seqnums (y)) := size (unique\_msg (y))

OK??

Backup point

Prvr\(\rightarrow\) theorem

H1: nhcanges (seqnums (w)) = nhcanges (seqnums (z))
in [0..1]
H2: size (u) = nhcanges (seqnums (z)) in [0..1]
H3: size (u) = size (unique\_msg (w)) in [0..1]
H4: size (unique\_msg (w)) = size (unique\_msg (x)) in [0..1]
H5: size (v) = size (unique\_msg (y)) in [0..1]
H6: size (unique\_msg (x)) = size (v) in [0..1]
H7: nhcanges (seqnums (z)) le size (unique\_msg (y))

\(\rightarrow\)

C1: size (u) = size (v) in [0..1]

Prvr\(\rightarrow\) sProceeding

Prvr\(\rightarrow\) sProceeding

size (u) = size (v) in [0..1]

Proved

Prvr\(\rightarrow\) theorem

H1: msg\_lag (unique\_msg (x), u, 1)
H2: msg\_lag (v, unique\_msg (x), 1)
H3: repeats (w)
H4: nhcanges (seqnums (w)) = nhcanges (seqnums (z))
in [0..1]
Proofs of Supporting Lemmas in Gypsy

H5: size (u) = nchanges (seqnums (z)) in [0..1]
H6: size (v) = size (unique_msg (v)) in [0..1]
H7: x sub w
H8: y sub x
H9: z sub y

=>
C1: z sub x

Prvr-> retain h8, h9
Backup point
(CLAIM . 9 .)

Prvr-> sProceeding
(CLAIM .)

z sub x
Established claim
abP_1 proved in theorem prover.

Exec-> prove main_lemma
Entering Prover with lemma main_lemma
all s : bit_seq,
  all x, y : pkt_seq,
    repeats (y)
      & nchanges (seqnums (y)) = nchanges (s) in [0..1]
      & s sub seqnums (x) & x sub y
    => msg_lag (unique_msg (x), unique_msg (y), 1)

H1: repeats (y)
H2: nchanges (seqnums (y)) = nchanges (s) in [0..1]
H3: s sub seqnums (x)
H4: x sub y

=>
C1: msg_lag (unique_msg (x), unique_msg (y), 1)
Backup point
(.

Prvr-> use interpolate
Backup point
( . 0 .

Prvr-> hypothesis

H1: repeats (y#2s)
  & nchanges (seqnums (y#2s)) = nchanges (s#2s)
  in [0..1] & s#2s sub seqnums (x#2s) & x#2s sub y#2s
  => nchanges (seqnums (y#2s))
  = nchanges (seqnums (x#2s))
  in [0..1]
H2: repeats (y)
H3: nchanges (seqnums (y)) = nchanges (s) in [0..1]
H4: s sub seqnums (x)
Proofs of Supporting Lemmas in Gypsy

H5: x sub y

Prv => forwardchain h1

::: :Forward chaining gives

\[ \text{nchanges}(\text{senums}(y)) = \text{nchanges}(\text{senums}(x)) \]

in \([0,1]\)

Backup point

(. U . FC .)

Prv => examp msg-lg

Backup point

(. U . FC . E .)

Prv => theorem

H1: \[ \text{nchanges}(\text{senums}(y)) = \text{nchanges}(\text{senums}(x)) \]

in \([0,1]\)

H2: \[ \text{repeats}(y#2s) \]

& \[ \text{nchanges}(\text{senums}(y#2s)) = \text{nchanges}(s#2s) \]

in \([0,1]\) & s#2s sub семумs(x#2s) & x#2s sub y#2s

=> \[ \text{nchanges}(\text{senums}(y#2s)) \]

= \[ \text{nchanges}(\text{senums}(x#2s)) \]

in \([0,1]\)

H3: \text{repeats}(y)

H4: \text{nchanges}(\text{senums}(y)) = \text{nchanges}(s) in \([0,1]\)

H5: s sub семумs(x)

H6: x sub y

=> C1: initial_subseq(unique_msg(x), unique_msg(y))

C2: size(unique_msg(y)) = size(unique_msg(x)) in \([0,1]\)

Prv => sProceeding

Backup point

(. U . FC . E . 1 .)

Prv => use sub-to-lg

Backup point

(. U . FC . E . 1 . U .)

Prv => theorem

H1: \[ \text{repeats}(y#4s) \]

& \[ \text{nchanges}(\text{senums}(y#4s)) = \text{nchanges}(\text{senums}(x#4s)) \]

in \([0,1]\) & x#4s sub y#4s

=> initial_subseq(unique_msg(x#1s), unique_msg(y#4s))

H2: \text{nchanges}(\text{senums}(y)) = \text{nchanges}(\text{senums}(x))

in \([0,1]\)

H3: \[ \text{repeats}(y#2s) \]

& \[ \text{nchanges}(\text{senums}(y#2s)) = \text{nchanges}(s#2s) \]

in \([0,1]\) & s#2s sub семумs(x#2s) & x#2s sub y#2s
Proofs of Supporting Lemmas in Gypsy

\[\Rightarrow n\text{changes}(\text{seqnums}(y^2s)) = n\text{changes}(\text{seqnums}(x^2s)) \quad\text{in}\ [0..1]\]

H4: \text{repeats}(y)

H5: \text{nchanges}(\text{seqnums}(y)) = \text{nchanges}(s) \quad\text{in}\ [0..1]

H6: s \subseteq \text{seqnums}(x)

H7: x \subseteq y

\[\Rightarrow C1: \text{initial-subseq}(\text{unique-msq}(x), \text{unique-msq}(y))\]

\text{Prv}r\Rightarrow \text{qed}

\[\quad\quad\emptyset\]

\text{proved}

\[\quad\quad\quad\emptyset\]

\text{prv}r\Rightarrow \text{sProceeding}

\[\quad\quad\emptyset\]

\text{initial-subseq}(\text{unique-msq}(x), \text{unique-msq}(y))

\text{prv}r\Rightarrow \text{sProceeding}

\text{backups point}

\[\quad\quad\emptyset\]

\text{prv}r\Rightarrow \text{theorem}

H1: \text{nchanges}(\text{seqnums}(y)) = \text{nchanges}(\text{seqnums}(x)) \quad\text{in}\ [0..1]

H2: \text{repeats}(y^2s)
Proofs of Supporting Lemmas in Gypsy

\[ \begin{align*}
\& n\text{changes} (\text{sequences (y\#s)}) = n\text{changes} (s\#2s) \\
\& \text{in } [0..1) \& s\#2s = s\#2s \text{ sub sequence (x\#s)} \& x\#2s = x\#2s \text{ sub } y\#2s \\
\rightarrow \ & n\text{changes} (\text{sequences (y\#2s)}) \\
\& = n\text{changes} (\text{sequences (x\#2s)}) \\
\& \text{in } [0..1) \\
H3: \ & \text{repeats (y)} \\
H4: \ & n\text{changes} (\text{sequences (y)}) = n\text{changes} (s) \text{ in } [0..1] \\
H5: \ & s \text{ sub sequence (x)} \\
H6: \ & x \text{ sub } y \\
\rightarrow \ & C1: \text{size (unique\_msg (y)) = size (unique\_msg (x)) in } [0..1]
\end{align*} \]

Prvr-> retain h1
Backup point
(, U, FC, E, 2, D, )

Prvr-> use nchanges\_unique
Backup point
(, U, FC, E, 2, D, U, )

Prvr-> use nchanges\_unique
Backup point
(, U, FC, E, 2, D, U, U, )

Prvr-> hypothesis

H1: \text{size (unique\_msg (x\#s)) = nchanges (sequences (x\#s))}
H2: \text{size (unique\_msg (x\#s)) = nchanges (sequences (x\#6s))}
H3: \text{nchanges (sequences (y)) = nchanges (sequences (x)) in } [0..1]

Prvr-> put
For what?* x\#s;
Put what?* y;

For what?* x\#s;
Put what?* x;

For what?* Sdone
Backup point
(, U, FC, E, 2, D, U, U, PUI, )

Prvr-> hypothesis

H1: \text{nchanges (sequences (y)) = nchanges (sequences (x)) in } [0..1]
H2: \text{size (unique\_msg (x)) = nchanges (sequences (x))}
H3: \text{size (unique\_msg (y)) = nchanges (sequences (y))}

Prvr-> equality substitute h2, h3
Proofs of Supporting Lemmas in Gyrsy

H2 yields
  size (unique_msg (x)) := nchanges (seqnums (x))
H3 yields
  size (unique_msg (y)) := nchanges (seqnums (y))
Backup point
  (, U , FC , E , 2 , 0 , U , U , PUT , =S , )

PrvrÆ $Proving$
  (, U , FC , E , 2 , )

  size (unique_msg (y)) = size (unique_msg (x)) in [0..1]
Proved

PrvrÆ $Proving$
  main_lemma proved in theorem prover.

ExecÆ prove interpolate
Entering Prover with lemma interpolate
all s : bit_seq,
  all x, y : pkt_seq,
    repeats (y)
   & nchanges (seqnums (y)) = nchanges (s) in [0..1]
   & s sub seqnums (x) & x sub y
  => nchanges (seqnums (y)) = nchanges (seqnums (x))
  in [0..1]

H1: repeats (y)
H2: nchanges (seqnums (y)) = nchanges (s) in [0..1]
H3: s sub seqnums (x)
H4: x sub y

  =>
  C1: nchanges (seqnums (y)) = nchanges (seqnums (x))
      in [0..1]
Backup point
  (, )

PrvrÆ use sub_seqnum
Backup point
  (, U , )

PrvrÆ use sub_nchanges
Backup point
  (, U , U , )

PrvrÆ hypothesis

H1: s#2s sub t#2s => nchanges (s#2s) le nchanges (t#2s)
H2: x#2s sub y#2s => seqnums (x#2s) sub seqnums (y#2s)
H3: repeats (y)
H4: nchanges (seqnums (y)) = nchanges (s) in [0..1]
H5: s sub seqnums (x)
H6: x sub y
Proofs of Supporting Lemmas in Gypsy

Prvř=> ne1
(. E . IFFC , 2 , P-> , =S , ( , QED )
(. E . IFFC , 2 , P-> , =S , ( , QED AC )
(. E . IFFC , 2 , P-> , =S , ( , QED AC )

v = v
Proved
QED

Prvř=> $proceeding
(. E . IFFC , 2 , )

u = v => size (u) = size (v) & initial_subseq (u , v)
Proved

Prvř=> $proceeding
msq_laeg proved in theorem prover.

Exec=> prove comp.ne
Entering Prover with lemma comp.ne
all b1 , b2 : bit , comp (b1) = b2 iff b1 ne b2

C1: comp (b1) = b2 iff b1 ne b2
Backup point
()

Prvř=> $proceeding
Backup point
(. IFFC , )

Prvř=> $proceeding
Backup point
(. IFFC , 1 , )

Prvř=> theorem

C1: comp (b1) = b2 -> b1 ne b2

Prvř=> $proceeding
Backup point
(. IFFC , 1 , P-> , )

Prvř=> theorem

H1: comp (b1) = b2
->
C1: b1 ne b2

prvř=> use bit_cases
Backup point
(. IFFC , 1 , P-> , U , )
Proofs of Supporting Lemmas in Gypsy

**Prv**r→ forwardchain h2
\[
\text{seqnums}(x) \text{ sub seqnums}(y)
\]
Backup point
\[
(\cdot, U, \cdot, FC, \cdot)
\]

**Prv**r→ hypothesis
\[
\begin{align*}
H1: & \text{ seqnums}(x) \text{ sub seqnums}(y) \\
H2: & s \#23 \text{ sub } t \#23 \Rightarrow \text{nchanges}(s \#23) \leq \text{nchanges}(t \#23) \\
H3: & x \#23 \text{ sub } y \#23 \Rightarrow \text{seqnums}(x \#23) \text{ sub seqnums}(y \#23) \\
H4: & \text{repeats}(v) \\
H5: & \text{nchanges}(\text{seqnums}(y)) = \text{nchanges}(s) \text{ in } [0, \cdot, 1] \\
H6: & s \text{ sub seqnums}(x) \\
H7: & x \text{ sub } y
\end{align*}
\]

**Prv**r→ forwardchain h2
\[: : : : \text{Forward chaining gives} : : : :\]
\[
\text{nchanges}(\text{seqnums}(x)) \leq \text{nchanges}(\text{seqnums}(y))
\]
Backup point
\[
(\cdot, U, \cdot, FC, \cdot)
\]

**Prv**r→ reorder h6
Backup point
\[
(\cdot, U, \cdot, FC, \cdot, \cdot, \text{REORDER}, \cdot)
\]

**Prv**r→ use sub-nchanges
Backup point
\[
(\cdot, U, \cdot, FC, \cdot, \cdot, \text{REORDER}, \cdot, \cdot)
\]

**Prv**r→ hypothesis
\[
\begin{align*}
H1: & s \#4S \text{ sub } t \#4S \Rightarrow \text{nchanges}(s \#4S) \leq \text{nchanges}(t \#4S) \\
H2: & s \text{ sub seqnums}(x) \\
H3: & \text{nchanges}(\text{seqnums}(x)) \leq \text{nchanges}(\text{seqnums}(y)) \\
H4: & \text{seqnums}(x) \text{ sub seqnums}(y) \\
H5: & s \#2S \text{ sub } t \#2S \Rightarrow \text{nchanges}(s \#2S) \leq \text{nchanges}(t \#2S) \\
H6: & x \#2S \text{ sub } y \#2S \Rightarrow \text{seqnums}(x \#2S) \text{ sub seqnums}(y \#2S) \\
H7: & \text{repeats}(v) \\
H8: & \text{nchanges}(\text{seqnums}(y)) = \text{nchanges}(s) \text{ in } [0, \cdot, 1] \\
H9: & x \text{ sub } y
\end{align*}
\]

**Prv**r→ forwardchain h1
\[: : : \text{Forward chaining gives} : : :\]
\[
\text{nchanges}(s) \leq \text{nchanges}(\text{seqnums}(x))
\]
Backup point
\[
(\cdot, U, \cdot, FC, \cdot, \cdot, \cdot, \text{REORDER}, \cdot, FC, \cdot)
\]
Proofs of Supporting Lemes in Gypsy

Prv→theorem

H1: \text{nchanges}(s) ≤ \text{nchanges}(\text{sequms}(x))
H2: s≠#s \implies \text{nchanges}(s≠#s) ≤ \text{nchanges}(t≠#s)
H3: s \subseteq \text{sequms}(x)
H4: \text{nchanges}(\text{sequms}(x)) ≤ \text{nchanges}(\text{sequms}(y))
H5: \text{sequms}(x) \subseteq \text{sequms}(y)
H6: s≠#2s \implies \text{nchanges}(s≠#2s) ≤ \text{nchanges}(t≠#2s)
H7: x≠#2s \implies \text{sequms}(x≠#2s) \subseteq \text{sequms}(y≠#2s)
H8: \text{repeats}(y)
H9: \text{nchanges}(\text{sequms}(y)) = \text{nchanges}(s) \text{ in } [0..1]
H10: x \subseteq y

→

C1: \text{nchanges}(\text{sequms}(y)) = \text{nchanges}(\text{sequms}(x))
    \text{ in } [0..1]

Prv→ retain h1,h4,h9

Backup point
(* U * FC * FC * REORDER * U * FC * )

Prv→theorem

H1: \text{nchanges}(s) ≤ \text{nchanges}(\text{sequms}(x))
H2: \text{nchanges}(\text{sequms}(x)) \leq \text{nchanges}(\text{sequms}(y))
H3: \text{nchanges}(\text{sequms}(y)) = \text{nchanges}(s) \text{ in } [0..1]

→

C1: \text{nchanges}(\text{sequms}(y)) = \text{nchanges}(\text{sequms}(x))
    \text{ in } [0..1]

Prv→ sProceeding

:::Interpolate proved in theorem prover.

Exec→ show status scope s

SCOPE LEMMAS

Waiting to be proved: APP_BIT_NONNULL, APP_MSG_NONNULL, APP_PKT_NONNULL,
BIT_CASES, COMP_ME, EQ_8SS, EQ_ISS_APP, HIST_SUB, ISS_APP, ISS_TRANS,
LAST_NEXT, LAST_UNIQUE, LAST_REPEATS, MSG_LAG_EQ, NE_NEXT, NEXT_COMP,
NCHANGES_UNIQUE, SUB_APP, SIZE_NULL, SUB_SEQNUM, SUB_TO_LAG,
SUB_NCHANGES

Proved: ABP, INTERPOLATE, MAIN_LEMA, PROP_REC_1, PROP_REC_2, PROP_REC_3,
PROP_REC_4, PROP_REC_5, PROP_TRANS_1, PROP_TRANS_2, PROP_TRANS_3,
PROP_TRANS_4, PROP_TRANS_5, PROP_TRANS_6, PROP_TRANS_7

Exec→ save abp.dmp

File ABP.DMP already exists. Rewrite it? → y

Saving........................................................................
........................................................................
Proofs of Supporting Lemmas in Gnosv

Exec-> prove next_comp
Entering Prover with lemma next_comp
all x : pkt_seq,
    all p : packet, comp (p,segno) = next_segunm (x @ [seq: p])

Cl: comp (p,segno) = next_segunm (x @ [seq: p])
Backup point
(.)

Provr-> qed
(., QED)
Ran out of tricks

Provr-> expand next_segunm
Backup point
(., QED E .)

Provr-> theorem

Cl: comp (p,segno)
    = comp (last_bit (segunms (x @ [seq: p])))

Provr-> qed
(., QED E . QED)
Ran out of tricks

Provr-> expand segunms
Backup point
(., QED E . QED E .)

Provr-> expand last_bit
Backup point
(., QED E . QED E . E .)

Provr-> simplify theorem

Provr-> theorem

Cl: true

Provr-> sProceeding
QED

Provr-> sProceeding
QED

Provr-> sProceeding
next-comp proved in theorem prover.

Exec-> prove ne_next
**Proofs of Supporting Lemmas in Gypsy**

**Entering Prover with lemma ne_next**

all x : pkt_seq,
   all p : packet,
   p.seqno ne next_seqnum (x) 
   => next_seqnum (x & [seq: p]) = next_seqnum (x)

H1: p.seqno ne next_seqnum (x)

=>

C1: next_seqnum (x & [seq: p]) = next_seqnum (x)

Backup point

(.,)

Prvr-> expand next_seqnum

More than one to expand.

Which do you want to expand? show-choices

1: next_seqnum (x)
2: next_seqnum (x & [seq: p])

Which do you want to expand? 2
Backup point

(., E ,)

Prvr-> theorem

H1: p.seqno ne next_seqnum (x)

=>

C1: comp (last_bit (seqnums (x & [seq: 0])))
    = next_seqnum (x)

Prvr-> expand seqnums

Backup point

(., E , E ,)

Prvr-> expand last_bit

Backup point

(., E , E , E ,)

Prvr-> simplify theorem

Prvr-> theorem

H1: p.seqno ne next_seqnum (x)

=>

C1: comp (p.seqno) = next_seqnum (x)

Prvr-> use comp_ne

Backup point

(., E , E , E , U ,)

Prvr-> hypothesis
Proofs of Supporting Lemmas in Gypsy

H1: comp (b1#2s) = b2#2s iff b1#2s ne b2#2s
H2: p.segno ne next.segnum (x)

Prvr-> forwardchaining h1
which way? (left or right)
left
::Forward chaining gives

comp (p.segno) = next.segnum (x)
Backup point
(. E . E . F . U . FC .)

Prvr-> $Proceeding
ne._next proved in theorem prover.

Exec-> prove last_next
Entering Prover with lemma last_next
all x : pkt_seq,
  all p : packet,
  x[size (x)] = p & null (pkt_seq) ne x
 => p.segno ne next.segnum (x)

H1: x[size (x)] = p
H2: null (pkt_seq) ne x

=>
C1: p.segno ne next.segnum (x)
Backup point
(. )

Prvr-> expand next.segnum
Backup point
(. E .)

Prvr-> simplify theorem

Prvr->theorem

H1: x[size (x)] = p
H2: null (pkt_seq) ne x

=>
C1: p.segno ne comp (last_bit (seqnums (x)))

Prvr-> expand seqnums
Backup point
(. E . F .)

Prvr-> expand last_bit
Backup point
(. E . E . F .)

Prvr-> simplify theorem
Proofs of Supporting Lemmas in Gypsy

Prvr->theorem

H1: x[size(x)] = p
H2: null (okt_seq) ne x

->
   C1: p.seqno ne comp (x[size(x)], seqno)

Prvr-> equality substitute h1
H1 can be solved for:
   T1: x[size(x)]
   T2: p

Which term (by label) do you want to substitute for?
   t1

   x[size(x)] := p

OK??

y

Backup point
   (* E E E E =S U )

Prvr-> theorem

H1: null (okt_seq) ne x

->
   C1: p.seqno ne comp (p, seqno)

Prvr-> use comp_ne
Backup point
   (* E E E E =S U )

Prvr-> theorem

H1: comp (b1#b2) = b2#b1 iff b1#b2 ne b2#b2
H2: null (okt_seq) ne x

->
   C1: p.seqno ne comp (p, seqno)

Prvr-> put

For what?* b1#b2;
Put what?* p.seqno;

For what?* b2#b2;
Put what?* p.seqno;

For what?* sdone

Backup point
   (* E E E E =S U PUT )

Prvr-> theorem
Proofs of Supporting Lemmas in Gymsy

H1: comp (p, seqno) = p, seqno iff p, seqno ne p, seqno
H2: null (pkt_seq) ne x

=>
C1: o, seqno ne comp (p, seqno)

Prvr=> sProceeding
last_next proved in theorem prover.

Exec=> prove last_unique
Entering Prover with lemma last_unique
all x : pkt_seq,
   all p : packet,
   x[size (x)] = p & null (pkt_seq) ne x
   => unique_msg (x @ [seq: p]) = unique_msg (x)

H1: x[size (x)] = p
H2: null (pkt_seq) ne x

=>
C1: unique_msg (x @ [seq: p]) = unique_msg (x)

Backup point
(. .)

Prvr=> expand unique_msg
More than one to expand,
Which do you want to expand? show=choices

1: unique_msg (x @ [seq: p])
2: unique_msg (x)

Which do you want to expand? 1
Backup point
(. . .)

Prvr=> simplify theorem

Prvr=> theorem

H1: x[size (x)] = p
H2: null (pkt_seq) ne x

=>
C1: unique_msg (x)
   = if p, seqno = next_seqnum (x)
      then unique_msg (x) @ [seq: p, msg]
      else unique_msg (x)
     fi

Prvr=> use last_next
Backup point
(. . .)

Prvr=> hypothesis
Proofs of Supporting Lemmas in Gypsy

H1: \( x \# 2s(\text{size}(x \# 2s)) = p \# 2s \& \text{null}(\text{pkt_seq}) \neq x \# 2s \)
   \( \Rightarrow p \# 2s, \text{segno ne next_segnum}(x \# 2s) \)
H2: \( x[\text{size}(x)] = p \)
H3: \( \text{null}(\text{pkt_seq}) \neq x \)

Prv
- forwardchain h1
  \( p, \text{segno ne next_segnum}(x) \)

Backup point
( (* E, U, FC, )

Prv
- simplify theorem

Prv
- theorem

C1: true

Prv
- $\text{proceeding}$
last_unique proved in theorem prover.

Exec
- prove last_repeats
Entering prover with lemma last_repeats
all x : pkt_seq,
   all p : packet,
      \( x[\text{size}(x)] = p \& \text{repeats}(x) \& \text{null}(\text{pkt_seq}) \neq x \)
   \( \Rightarrow \text{repeats}(x \@ ([\text{seq}: p])) \)

H1: \( x[\text{size}(x)] = p \)
H2: \( \text{repeats}(x) \)
H3: \( \text{null}(\text{pkt_seq}) \neq x \)

\( \Rightarrow \)
C1: \( \text{repeats}(x \@ ([\text{seq}: p])) \)
Backup point
( *

Prv
- expand repeats
more than one to expand,
which do you want to expand? show choices

1: \( \text{repeats}(x) \)
2: \( \text{repeats}(x \@ ([\text{seq}: p])) \)

which do you want to expand? 2
Backup point
( *, E, )

Prv
- simplify theorem

Prv
- theorem
Proofs of Supporting Lemmas in Gypsy

C1: true

Prvr-> sProceeding
last-repeats proved in theorem prover.

Exec->

% Now we need to stop and fill in one last specification function.
% The definition of "initial-subseq" was deliberately left pending
% to avoid certain unstable proper behavior when is was expanded.
% We now give it a definition in order to prove the lemmas which
% use it.

translate tty:

Gypsy Text: (terminate with "Z")

 Sextending scope alt-hit-specs =
 begin
 function initial-subseq (u, v: msg-seq) : boolean =
 begin
 exit (assume result iff some s: msg-seq, u & s = v);
 end;
 end;

"Z

No syntax errors detected
No semantic errors detected

Exec-> set scope lemmas

Exec-> prove eq-iss
Entering Prover with lemma eq-iss
all u, v : msg-seq, u = v -> initial-subseq (u, v)

Hit: u = v

=>
  C1: initial-subseq (u, v)
Backup point
(*)

Prvr-> expand initial-subseq
backup point
(*) (*

Prvr-> theorem
Proofs of Supporting Lemmas in Gypsy

H1:  u = v

=>

C1:  u @ s #4s = v

Prvr-> sProceeding
Ran out of tricks

Prvr-> put

For what?* s #4s;
Put what?* null(msg_seq);

For what?* sdone

Backup point
(.* F.* PUT .*)

Prvr-> sProceeding
eq-iss proved in theorem prover.

Exec-> prove eq-iss_app
Entering Prover with lemma eq-iss_app
all u, v : msg_seq,
  all m : message,
  u = v  =>  initial_subseq (u @ (seg; m), v @ (seg; m))

H1:  u = v

=>

C1:  initial_subseq (u @ (seg; m), v @ (seg; m))

Backup point
(*)

Prvr-> expand initial_subseq
Backup point
(.* E .*)

Prvr-> theorem

H1:  u = v

=>

C1:  u @ (seg; m) @ s #4s = v @ (seg; m)

Prvr-> put

For what?* s #4s;
Put what?* null(msg_seq);

For what?* sdone

Backup point
(.* F.* PUT .*)
Proofs of Supporting Lemmas in Gypsy

Prvr-> sProceeding
Conclusion simplified to:
u @ [seq: m] = v @ [seq: m] Ran out of tricks

Prvr-> theorem

H1: u = v
=>
C1: u @ [seq: m] @ null (msg_seq) = v @ [seq: m]

Prvr-> equality substitute h1
H1 can be solved for:
I1: u
I2: v
which term (by label) do you want to substitute for?
t1

u := v
OK??
y
Backup point
( . E . PUT . =S . )

Prvr-> sProceeding
eq iss_app proved in theorem prover.

Exec-> prove iss_app
Entering Prover with lemma iss_app
all u, v : msg_seq,
  all m : message,
  initial_subseq (u, v) -> initial_subseq (u, v @ [seq: m])

H1: initial_subseq (u, v)
=>
C1: initial_subseq (u, v @ [seq: m])
Backup point
( . )

Prvr-> expand initial_subseq
More than one to expand,
which do you want to expand? all
Backup point
( . E . )

Prvr-> theorem

H1: u @ s#2 = v
=>
C1: u @ s#7s = v @ [seq: m]

Prvr-> simplify theorem
Proofs of Supporting Lemmas in Gypsy

Prv⇒theorem

H1: u ⊕ s#2 = v
⇒
  C1: u ⊕ s#7s = v ⊕ [seq; m]

Prv⇒ equality substitute h1
H1 yields
  v := u ⊕ s#2
Backup point
  (, E . =S ,)

Prv⇒theorem

H1: true
⇒
  C1: u ⊕ s#2 ⊕ [seq; m] = u ⊕ s#7s

Prv⇒ sProceeding
iss-apo proved in theorem prover.

Exec⇒ prove iss_trans
Entering Prover with lemma iss_trans
all u, v, w : msg_seq,
  initial_subseq (u, v) & initial_subseq (v, w)
  ⇒ initial_subseq (u, w)

H1: initial_subseq (u, v)
H2: initial_subseq (v, w)
⇒
  C1: initial_subseq (u, w)
Backup point
(,)

Prv⇒ expand initial_subseq
more than one to expand.
Which do you want to expand? all
Backup point
(, E ,)

Prv⇒ simplify theorem

Prv⇒ theorem

H1: u ⊕ s#2 = v
H2: v ⊕ s#3 = w
⇒
  C1: u ⊕ s#10s = w

Prv⇒ sProceeding
::: Ran out of tricks
Proofs of Supporting Lemmas in Gopsy

PrvR → equality substitute h1
H1 yields
  v := u @ s#2
Backup point
( E . =S . )

PrvR → theorem

H1: u @ s#2 @ s#3 = w
⇒
  C1: u @ s#10s = w

PrvR → equality substitute h1
H1 yields
  w := u @ s#2 @ s#3
Backup point
( E . =S . =S . )

PrvR → theorem

H1: true
⇒
  C1: u @ s#2 @ s#3 = u @ s#10s

PrvR → sProceeding
iss_trans proved in theorem prover.

Exec → prove msg_lag_eq
Entering Prover with lemma msg_lag_eq
all u, v : msg_seq, u = v iff msg_lag (u, v, 0)

  C1: u = v iff msg_lag (u, v, 0)
Backup point
( . )

PrvR → expand msg_lag
Backup point
( E . )

PrvR → simplify theorem

PrvR → theorem

  C1: size (u) = size (v) & initial_subse (u, v) iff u = v

PrvR → sProceeding
Backup point
( E , IFFC . )

PrvR → theorem
Proofs of Supporting Lemmas in Gypsy

C1: size (u) = size (v) & initial-subseq (u, v) -> u = v
C2: u = v -> size (u) = size (v) & initial-subseq (u, v)

Prvr=> sProceeding
Backup point
(*. E. IFFC . 1 ,)

Prvr=> theorem

C1: size (u) = size (v) & initial-subseq (u, v) -> u = v

Prvr=> sProceeding
Typelist equalities

size (v) = size (u)
Backup point
(*. E. IFFC . 1 . p-> .)

Prvr=> theorem

H1: size (v) = size (u)
H2: size (u) = size (v)
H3: initial-subseq (u, v)

=>

C1: u = v

Prvr=> expand initial-subseq
Backup point
(*. E. IFFC . 1 . p-> . E .)

Prvr=> simplify theorem

Prvr=> theorem

H1: u @ s #2 = v
H2: size (u) = size (v)
H3: size (v) = size (u)

=>

C1: u = v

Prvr=> use size-null
Backup point
(*. E. IFFC . 1 . p-> . E . .

Prvr=> theorem

H1: null (msg-seq) = u #2 s iff () = size (u #2 s)
H2: u @ s #2 = v
H3: size (u) = size (v)
H4: size (v) = size (u)

=>

C1: u = v
Proofs of Supporting Lemmas in Gypsy

prv1=> procedure
    ... Ran out of tricks

prv1=> put

for what? u? 2:
put what? s? 2:

for what? s done

backup point

(. E . IFFC . 1 . p-> . E . U . PUT .)

prv1=> theorem

H1: null (msg-seq) = s? 2 iff 0 = size (s? 2)
H2: size (v) = size (u)
H3: size (u) = size (v)
H4: u ? s? 2 = v

=>
C1: u = v

prv1=> simplify theorem

prv1=> theorem

H1: u ? s? 2 = v
H2: size (u) = size (v)
H3: size (v) = size (u)
H4: null (msg-seq) = s? 2 iff 0 = size (s? 2)

=>
C1: u = v

prv1=> equality substitute h1
H1 yields
v := u ? s? 2

backup point


prv1=> theorem

H1: null (msg-seq) = s? 2
H2: 0 = size (s? 2)

=>
C1: u ? s? 2 = u

prv1=> simplify theorem

prv1=> theorem

C1: true
Prvr-> sProceeding
(. E . IFFC . 1 .)

size (u) = size (v) & initial_subseq (u, v) -> u = v
Proofed

Prvr-> sProceeding
Backup point
(. E . IFFC . 2 .)

Prvr-> theorem

C1: u = v -> size (u) = size (v) & initial_subseq (u, v)

Prvr-> sProceeding
Backup point
(. E . IFFC . 2 . p-> ,)

Prvr-> theorem

H1: u = v
->
  C1: size (u) = size (v)
  C2: initial_subseq (u, v)

Prvr-> equality substitute H1
H1 can be solved for:
  H1: u
  H2: v
which term (by label) do you want to substitute for?
  H1

  u := v
OK??

Backup point
(. E . IFFC . 2 . p-> . =S .)

Prvr-> sProceeding
Ran out of tricks

Prvr-> theorem

H1: true
->
  C1: initial_subseq (v, v)

Prvr-> use eq_iess
Backup point
(. E . IFFC . 2 . p-> . =S . u .)
Proofs of Supporting Lemmas in Cypsys

Prvr-> theorem

\[ H_1: b#2s = \text{one or } b#2s = \text{zero} \]
\[ H_2: \text{comp} (b_1) = b_2 \]

=>
\[ C_1: b_1 \neq b_2 \]

Prvr-> expand comp

Backup point
\[ (* \text{IFFC * 1 * P-> * J * E *}) \]

Prvr-> simplify theorem

Prvr-> theorem

\[ H_1: \text{if } b_1 = \text{zero then one else zero } f_1 = b_2 \]
\[ H_2: b#2s = \text{one or } b#2s = \text{zero} \]

=>
\[ C_1: b_1 \neq b_2 \]

Prvr-> put

For what?* b#2s;
Put what?* b_1
*

For what?* $\text{done}$

Backup point
\[ (* \text{IFFC * 1 * P-> * U * E * PUT *}) \]

Prvr-> simplify theorem

Prvr-> theorem

\[ H_1: \text{if } b_1 = \text{zero then one else zero } f_1 = b_2 \]
\[ H_2: b_1 = \text{one or } b_1 = \text{zero} \]

=>
\[ C_1: b_1 \neq b_2 \]

Prvr-> qed
\[ (* \text{IFFC * 1 * P-> * U * E * PUT * QED *}) \]

\[ \text{b}_2 := \text{if } b_1 = \text{zero then one else zero } f_1 \]
\[ (* \text{IFFC * 1 * P-> * U * E * PUT * QED =S *) \]

Attempting CASE1

\[ b_1 = \text{one} \]
\[ (* \text{IFFC * 1 * P-> * U * E * PUT * QED =S CASE1 *}) \]

Ran out of tricks
Proofs of Supporting Lemmas in Gypsy

Prvr-> theorem

H1: b1 = one

=>

C1: if b1 = zero then one else zero fi ne b1

Prvr-> simplify theorem

Prvr-> theorem

H1: b1 = one

=>

C1: if b1 = zero then one else zero fi ne b1

Prvr-> equality substitute h1

H1 can be solved for:

T1: b1
T2: one

which term (by label) do you want to substitute for?

b1

b1 := one

OK??

Y

Backup point

(* IFFC 1 P-> U E PUT QED =S CASE1 =S ,)

Prvr-> proceeding

(* IFFC 1 P-> U E PUT QED =S CASE1 ,)

b1 = one => if b1 = zero then one else zero fi ne b1

Proved

Attempting CASE2

b1 = zero

Backup point

(* IFFC 1 P-> U E PUT QED =S CASE2 ,)

Prvr-> theorem

H1: b1 = zero

=>

C1: if b1 = zero then one else zero fi ne b1

Prvr-> equality substitute h1

H1 can be solved for:

T1: b1
T2: zero

which term (by label) do you want to substitute for?

b1
Proofs of Supporting Lemmas in Gypsy

{\begin{verbatim}
\begin{prover}
\b1 := \text{zero} \\
\text{OK??} \\
V \\
\text{Backup point} \\
(* \text{IFFC \cdot 1 \cdot P\rightarrow \cdot U \cdot E \cdot PUT \cdot QED = S \ \text{CASE2} \cdot = S \cdot *)
\end{prover}

\begin{prover}
\b1 = \text{zero} \rightarrow \text{if } \b1 = \text{zero} \text{ then one else zero } \text{fi ne } \b1 \\
\text{Proved} \\
\text{Done with cases} \\
\text{QED}
\end{prover}

\begin{prover}
\begin{prover}
\text{comp (b1)} = b2 \rightarrow b1 ne b2 \\
\text{Proved}
\end{prover}
\text{Backup point} \\
(* \text{IFFC \cdot 2 \cdot *)
\begin{prover}
\text{theorem} \\
C1: b1 ne b2 \rightarrow \text{comp (b1)} = b2 \\
\text{Proved}
\end{prover}
\text{Backup point} \\
(* \text{IFFC \cdot 2 \cdot P\rightarrow \cdot *)
\begin{prover}
\text{theorem} \\
H1: b1 ne b2 \\
\rightarrow \\
C1: \text{comp (b1)} = b2 \\
\text{Proved}
\end{prover}
\text{Backup point} \\
(* \text{IFFC \cdot 2 \cdot P\rightarrow \cdot E \cdot *)
\begin{prover}
\text{simplify theorem} \\
\text{Proved} \\
H1: b1 ne b2 \\
\rightarrow \\
C1: \text{if } b1 = \text{zero} \text{ then one else zero } \text{fi } = b2 \\
\text{Proved}
\end{prover}
\text{use bit\_cases}
\end{verbatim}
Proofs of Supporting Lemmas in Gypsy

Backup point
(* IFFC . 2 . P-> . E . U .)

Prvrm-> hynothesis

H1: b*4s = one or b*4s = zero
H2: b1 ne b2

Prvrm-> put

For what?* b*4s;
Put what?* b1;

For what?* sdone

Backup point
(* IFFC . 2 . P-> . E . U . PUT .)

Prvrm-> theorem

H1: b1 = one or b1 = zero
H2: b1 ne b2

->
C1: if b1 = zero then one else zero f1 = b2

Prvrm-> qed
(* IFFC . 2 . P-> . E . U . PUT . QED) ::Attempting CASE1

b1 = one
(* IFFC . 2 . P-> . E . U . PUT . QED CASE1)
 ::: Ran out of tricks

Prvrm-> theorem

H1: b1 = one
H2: b1 ne b2

->
C1: if b1 = zero then one else zero f1 = b2

Prvrm-> equality substitute h1
41 yields
b1 := one

Backup point
(* IFFC . 2 . P-> . E . U . PUT . QED CASE1 =S .)

Prvrm-> sProceeding
Ran out of tricks

Prvrm-> theorem

H1: b2 ne one
Proofs of Supporting Lemmas in Gypsy

\[ \Rightarrow \]

C1: \( b_2 = 0 \)

Prv

\( \Rightarrow \) use hit_cases

Backup point

(\* IFFC 2 P\* E U PUT \* \*FU CASE1 \* S \* U \*)

Prv

\( \Rightarrow \) theorem

H1: \( b \neq 6s = 0 \) or \( b \neq 6s = 0 \)

H2: \( b_2 \neq 1 \)

\( \Rightarrow \)

C1: \( b_2 = 0 \)

Prv

\( \Rightarrow \) put

For \* what?* b \#6s;
Put \* what?* b2;
For \* what?* s\*done

Backup point

(\* IFFC 2 P\* E U PUT \* \*FU CASE1 \* S \* U \*)

Prv

\( \Rightarrow \) proceeding

\* Attempting CASE1

\( b_2 = 1 \)

Backup point

(\* IFFC 2 P\* E U PUT \* \*FU CASE1 \* S \* U \*)

Prv

\( \Rightarrow \) proceeding

(\* IFFC 2 P\* E U PUT \* \*FU CASE1 \* S \* U \*)

\( b_2 = \text{one} \Rightarrow b_2 = 0 \)

Proved

Attempting CASE2

\( b_2 = 0 \)

Backup point

(\* IFFC 2 P\* E U PUT \* \*FU CASE1 \* S \* U \*)

Prv

\( \Rightarrow \) proceeding

(\* IFFC 2 P\* E U PUT \* \*FU CASE1 \* S \* U \*)

\( b_2 = \text{zero} \Rightarrow b_2 = \text{zero} \)

Proved

Done with cases

(\* IFFC 2 P\* E U PUT \* \*FU CASE1 \*)

\( b_1 = \text{one} \Rightarrow \text{if } b_1 = \text{zero then one else zero } f_1 = b_2 \)

Proved
Proofs of Supporting Lemmas in Gypsy

Attempting CASE2

\[ b_1 = \text{zero} \]
Backup point
\[ (\text{IFFC} \ 2 \ \text{P} \rightarrow \ E \ 0 \ \text{PUT} \ \text{QED CASE2}) \]

Prvr\rightarrow sProceeding
:::Ran out of tricks

Prvr\rightarrow theorem

H1: \( b_1 = \text{zero} \)
H2: \( b_1 \neq b_2 \)

\[ \rightarrow \]
C1: if \( b_1 = \text{zero} \) then one else zero \( f_i = b_2 \)

Prvr\rightarrow equality substitute H1

H1 yields
\[ b_1 := \text{zero} \]
Backup point
\[ (\text{IFFC} \ 2 \ \text{P} \rightarrow \ E \ 0 \ \text{PUT} \ \text{QED CASE2} :=S :=S) \]

Prvr\rightarrow sProceeding
Ran out of tricks

Prvr\rightarrow theorem

H1: \( b_2 \neq \text{zero} \)

\[ \rightarrow \]
C1: \( b_2 = \text{one} \)

Prvr\rightarrow use bit_cases

Backup point
\[ (\text{IFFC} \ 2 \ \text{P} \rightarrow \ E \ 0 \ \text{PUT} \ \text{QED CASE2} :=S :=S :=S) \]

Prvr\rightarrow hypothesis

H1: \( b\#S = \text{one} \) or \( b\#S = \text{zero} \)
H2: \( b_2 \neq \text{zero} \)

Prvr\rightarrow put

For what?* \( b\#S; \)
Put what?* \( b?; \)

For what?* $done

Backup point
\[ (\text{IFFC} \ 2 \ \text{P} \rightarrow \ E \ 0 \ \text{PUT} \ \text{QED CASE2} :=S :=S :=S :=S) \]

Prvr\rightarrow sProceeding
:::Attempting CASE1
Proofs of Supporting Lemmas in Gypsy

\( b2 = \text{one} \)
Backup point
Prv\(v\)-> sProceeedings
\( b2 = \text{one} \rightarrow b2 = \text{one} \)
Proved
Attempting CASE2

\( b2 = \text{zero} \)
Backup point
Prv\(v\)-> sProceeedings
\( b2 = \text{zero} \rightarrow b2 = \text{one} \)
Proved
Done with cases
(.* IFFC . 2 . P=> . E . U . PUT . QED CASE2 .)
\( b1 = \text{zero} \rightarrow \text{if } b1 = \text{zero then one else zero } fi = b2 \)
Proved
Done with cases
QED
Prv\(v\)-> sProceeedings
(.* IFFC . 2 .)
\( b1 \neq b2 \rightarrow \text{comp (b1)} = b2 \)
Proved
Prv\(v\)-> sProceeedings
\text{comp-ne proved in theorem prover.}
Exec-> prove hist-sub
Entering Prover with lemma hist-sub
all x, y : pktbuf,
  allfrom (x) sub allfrom (y) \rightarrow allfrom (x) sub allto (y)
Cut: true
Backup point
(.*
Prv\(v\)-> sProceeedings
hist-sub proved in theorem prover.
Exec->
The following lemma constitutes the entire basis for the proof.

prove bit_cases
Entering Prover with lemma bit_cases
all b : bit, b = one or b = zero

C1: b = one or b = zero
Backup point
(.*

prvr-> simplify theorem
prvr-> theorem

C1: b = one or b = zero
prvr->

% This lemma must be assumed since the prover does not have the
% information needed about the enumeration type "bit".

assume

Select assumed subgoal name bit_cases

Enter justification (terminate with Escape)
see above.
sbit_cases proved in theorem prover.

exec->

% The following lemmas were once needed but no longer are
% due to improvements in the simplifier.

prove app_pktnonnull
Entering Prover with lemma app_pktnonnull true

C1: true
Backup point
(.*

prvr-> sProceeding
app_pktnonnull proved in theorem prover.

exec-> prove app_msgnonnull
Entering Prover with lemma app_msgnonnull true
Proofs of Supporting Lemmas in Gypsy

Cl: true
Backup point (~)

Prv-> sProceeding
app_msgnonnull proved in theorem prover.

Exec-> prove app_bitnonnull
Entering Prover with lemma app_bitnonnull
true

Cl: true
Backup point (~)

Prv-> sProceeding
app_bitnonnull proved in theorem prover.

Exec->

% Finally, we come to the remaining set of lemmas which are those
% proved under the Affirm system. Their definitions have noted
% that they are to be assumed so they will just fall through the
% prover here.

prove sub-app
Entering Prover with lemma sub-app
true

Cl: true
Backup point (~)

Prv-> sProceeding
sub-app proved in theorem prover.

Exec-> prove size-null
Entering Prover with lemma size-null
true

Cl: true
Backup point (~)

Prv-> sProceeding
size-null proved in theorem prover.

Exec-> prove sub-seqnum
Entering Prover with lemma sub-seqnum
true
Proofs of Supporting Lemmas in Gypsy

\[ C1: \text{true} \]
\[ \text{Backup point} \]
\[ (\ast) \]

Prvr -> SProceeding
\[ \text{sub-seqnum proved in theorem prover.} \]

Exec -> prove sub-seqnum
\[ \text{Entering Prover with lemma sub-seqnum true} \]

\[ C1: \text{true} \]
\[ \text{Backup point} \]
\[ (\ast) \]

Prvr -> SProceeding
\[ \text{sub-seqnum proved in theorem prover.} \]

Exec -> prove sub-nchanges
\[ \text{Entering Prover with lemma sub-nchanges true} \]

\[ C1: \text{true} \]
\[ \text{Backup point} \]
\[ (\ast) \]

Prvr -> SProceeding
\[ \text{sub-nchanges proved in theorem prover.} \]

Exec -> prove nchanges_unique
\[ \text{Entering Prover with lemma nchanges_unique true} \]

\[ C1: \text{true} \]
\[ \text{Backup point} \]
\[ (\ast) \]

Prvr -> SProceeding
\[ \text{nchanges_unique proved in theorem prover.} \]

Exec -> prove sub-to-laq
\[ \text{Entering Prover with lemma sub-to-laq true} \]

\[ C1: \text{true} \]
\[ \text{Backup point} \]
\[ (\ast) \]

Prvr -> SProceeding
\[ \text{sub-to-laq proved in theorem prover.} \]
Proofs of Supporting Lemmas in Gypsy

Exec> show status all

The current design and verification status is:

SCOPE ALT-BIT-PROTOCOL

Waiting for pending body to be filled in: MEDIUM, TIMER

Proved: AB-PROTOCOL, COMP, RECEIVER, SENDER

Types, constants: BIT, BIT_SEQ, CLK_BUF, MESSAGE, MSG_BUF, MSG_SEQ, PACKET, PKT_BUF, PKT_SEQ

SCOPE LEMMAS


**B**

**SCOPE ALT-BIT-SPECS**

For specifications only: INITIAL_SUBSEP, LAST_BIT, MSG_LAG, NCHANGES, NEXT_SEQNUM, PROPER_RECEPTION, PROPER_TRANSMISSION, REPEATS, SEQNUMS, UNIQUE_MSG

Exec> save abv.dmp

File ABV.DMP already exists. Rewrite it? -> Y

Saving..............................................................
..............................................................
..............................................................
..............................................................
..............................................................
..............................................................

Exec>

% This ends the Gyspy part of our proof of the Alternating Bit Protocol.
% Be sure to tune in next time when we try to prove bigger and better
% protocols.

[HOCOPY terminated at Sat 9-May-81 14:34:34]
AFFIRM Type Specifications and Proofs

Transcript file <DIVITO>AFFIRM=TRANSCRIPT,R=FER=81.1
is open in the AFFIRM system <AFFIRM>AFFIRM.EXE.120

1 U: load apcontext:
  file created for AFFIRM on 31-Jan-81 15:27:15
ABPCONTTEXT
  file created for AFFIRM on 4-Feb-81 21:48:05
BITCOMS
  <DIVITO>BIT..2
  file created for AFFIRM on 3-Jan-81 22:00:35
PACKETCOMS
  compiled for AFFIRM on 20-Dec-80 15:42:13
  file created for AFFIRM on 20-Dec-80 15:42:10
ELEMTYPEDCOMS

(File created under Affirm 121)
<DIVITO>PACKET..2
  file created for AFFIRM on 31-Jan-81 15:28:55
SEQUENCEOFELEMTYPEDCOMS
  <DIVITO>SEQUENCEOFELEMTYPE..5
  file created for AFFIRM on 31-Jan-81 15:25:36
SEQUENCEOFBITCOMS
  <DIVITO>SEQUENCEOFBIT..5
  file created for AFFIRM on 7-Feb-81 06:44:08
SEQUENCEOFPACKETCOMS
  <DIVITO>SEQUENCEOFPACKET..7
  <DIVITO>ABPCONTEXT..3

2 U: print type elmttype;
(type elmttype => ElemType)

type ElemType:

declaration dummy: ElemType;
axiom dummy = dummy = TRUE;
end (ElemType);

3 U: print type bit;
(type bit => Bit)

type Bit:

declaration dummy, b: Bit;
interfaces zero, one, comp(b): Bit;
interface NormalForm(b): Boolean;
axioms dummy = dummy = TRUE,
zero = one = FALSE,
AFFIRM Type Specifications and Proofs

one = zero == FALSE;

axioms
  comp(zero) == one,
  comp(one) == zero;

schema
  NormalForm(b) == cases(Prop(zero), Prop(one));

end {Bit} ;

4 U: print type packet;
(packet => Packet)

type Packet;

needs types bit, ElemType;

declare dummy, p: Packet;

interface msg(p): ElemType;

interface seqno(p): bit;

axiom dummy=dummy == TRUE;

end {Packet} ;

5 U: print type sequenceofElemType;
(sequenceofElemType => SequenceofElemType)

type SequenceofElemType;

needs types Integer, ElemType;

declare dummy, ss, s, s1, s2, s3, s4, s5: SequenceofElemType;

declare k, k1, k2: Integer;

declare li, i, l1, l2, j: ElemType;

interfaces
  NewSequenceofElemType, s aor i, i apl s, seq(i),
  s1 join s2, LessFirst(s), LessLast(s): SequenceofElemType;

infix
  join, apl, aor;

interfaces
  isNewSequenceofElemType(s), s1 subseq s2, FirstInduction(s),
  Induction(s), NormalForm(s), i in s, s1 iss s2: Boolean;

infix
  in, subseq, iss;

interface
  Length(s): Integer;

interfaces
  First(s), Last(s): ElemType;

axioms
  dummy=dummy == TRUE,
AFFIRM Type Specifications and Proofs

\[ \text{NewSequenceOfElemType} = s \text{ apr 1} = \text{FALSE}, \]
\[ s \text{ apr 1} = \text{NewSequenceOfElemType} = \text{FALSE}, \]
\[ s \text{ apr 1} = s1 \text{ apr 11} = ((s=s1) \text{ and } (i=11)); \]

axioms
\[ \text{api NewSequenceOfElemType} = \text{NewSequenceOfElemType} \text{ apr 1}, \]
\[ \text{api} (s \text{ apr 11}) = (i \text{ apr 1} s) \text{ apr 11}; \]

axiom
\[ \text{seq}(i) = \text{NewSequenceOfElemType} \text{ apr 1}; \]

axioms
\[ \text{NewSequenceOfElemType} \text{ join } s = s, \]
\[ (s \text{ apr 1}) \text{ join } s1 = s \text{ join } (i \text{ apr 1} s1); \]

axiom
\[ \text{LessFirst}(s \text{ apr 1}) \]
\[ = \text{if } s = \text{NewSequenceOfElemType} \]
\[ \text{ then } \text{NewSequenceOfElemType} \]
\[ \text{ else } \text{LessFirst}(s) \text{ apr 1}; \]

axiom
\[ \text{LessLast}(s \text{ apr 1}) = s; \]

axiom
\[ \text{isNewSequenceOfElemType}(s) = (s = \text{NewSequenceOfElemType}); \]

axioms
\[ s1 \text{ subseq } (s \text{ apr 1}) \]
\[ = ( (s1 = \text{NewSequenceOfElemType}) \text{ or } s1 \text{ subseq } s \]
\[ \text{ or } \text{LessLast}(s1) \text{ subseq } s \text{ and } \text{Last}(s1) = i), \]
\[ s \text{ subseq } \text{NewSequenceOfElemType} = (s = \text{NewSequenceOfElemType}); \]

axioms
\[ i \text{ in } \text{NewSequenceOfElemType} = \text{FALSE}, \]
\[ i \text{ in } (s \text{ apr 11}) = (i \text{ in } s \text{ or } (i=11)); \]

axioms
\[ s \text{ iss } \text{NewSequenceOfElemType} = (s = \text{NewSequenceOfElemType}), \]
\[ s1 \text{ iss } (s2 \text{ apr 1}) \]
\[ = ( (s1 = \text{NewSequenceOfElemType}) \text{ or } s1 \text{ iss } s2 \]
\[ \text{ or } (\text{LessLast}(s1) = s2) \text{ and } \text{Last}(s1) = i)); \]

axioms
\[ \text{Length(\text{NewSequenceOfElemType})} = 0, \]
\[ \text{Length}(s \text{ apr 1}) = \text{Length}(s) + 1; \]

axiom
\[ \text{First}(s \text{ apr 1}) \text{ = if } s = \text{NewSequenceOfElemType} \]
\[ \text{ then } i \]
\[ \text{ else } \text{First}(s); \]

axiom
\[ \text{Last}(s \text{ apr 1}) = i; \]

rulemmas
\[ \text{NewSequenceOfElemType} = i \text{ apr 1} = \text{FALSE}, \]
\[ i \text{ apr 1} s = \text{NewSequenceOfElemType} = \text{FALSE}; \]

rulemmas
\[ s \text{ join } (s1 \text{ apr 1}) = (s \text{ join } s1) \text{ apr 1}, \]
\[ s \text{ join } \text{NewSequenceOfElemType} = s, \]
\[ (i \text{ apr s1}) \text{ join } s2 = i \text{ apr } (s1 \text{ join } s2), \]
\[ (s \text{ join } (i \text{ apr s1})) \text{ join } s2 \]
\[ = s \text{ join } (i \text{ apr } (s1 \text{ join } s2)), \]
\[ s \text{ join } (s1 \text{ join } s2) = (s \text{ join } s1) \text{ join } s2; \]
 AFFIRM Type Specifications and Proofs

rulelemma \text{LessFirst}(i \ aol \ s) == s;

rulelemma \text{LessLast}(i \ aol \ s)
  == if \ s = \text{NewSequenceOfElemType} then \text{NewSequenceOfElemType}
        else \ i \ aol \ \text{LessLast}(s);

rulelemmas \text{NewSequenceOfElemType} \ \text{subset} \ s == \text{TRUE},
        s \ \text{subset} \ s == \text{TRUE};

rulelemma \ i \ \text{in} \ (ii \ aol \ s) == (i \ \text{in} \ s \ \text{or} \ (i == ii));

rulelemmas \text{NewSequenceOfElemType} \ \text{iss} \ s == \text{TRUE},
        s \ \text{iss} \ s == \text{TRUE};

rulelemma \ \text{First}(i \ aol \ s) == i;

rulelemma \ \text{Last}(i \ aol \ s) == if \ s = \text{NewSequenceOfElemType}
        then \ i
        else \ \text{Last}(s);

schemas \ \text{FirstInduction}(s)
  == \text{cases}(\text{Prop(\text{NewSequenceOfElemType})}, \ \text{all} \ ss, \ ii
            ( \ \text{IH}(ss)
              \ \text{imp} \ \text{Prop}( \ ii
                  aol \ ss))));

\text{Induction}(s)
  == \text{cases}(\text{Prop(\text{NewSequenceOfElemType})}, \ \text{all} \ ss, \ ii
            ( \ \text{IH}(ss)
              \ \text{imp} \ \text{Prop}( \ ss
                  aor \ ii))));

\text{NormalForm}(s)
  == \text{cases}(\text{Prop(\text{NewSequenceOfElemType})}, \ \text{all} \ ss, \ ii (\text{Prop}( \ ss
                  aor \ ii))))
;

end \ (\text{SequenceOfElemType}) ;

6 \ i: \text{print type sequenceOfbit};
(\text{sequenceOfbit} \Rightarrow \text{SequenceOfbit})

type \text{SequenceOfbit};

needs \text{type Bit};

declare \ \text{dummy}, ss, s, s1, s2, s3, s4, s5: \text{SequenceOfbit};
declare \ k, k1, k2: \text{Integer};
declare \ ii, i, ii, i2, j: \text{Bit};
interfaces NewSequenceOfBit, s apr i, i aol s, sen(i), s1 join s2, LessFirst(s), LessLast(s): SequenceOfBit;

infix join, aol, aor;

interfaces isNewSequenceOfBit(s), s1 subset s2, FirstInduction(s), Induction(s), NormalForm(s), i in s, s1 iss s2: Boolean;

infix in, subseq, iss;

interfaces Nchanges(s), Length(s): Integer;

interfaces First(s), Last(s), LastBit(s): Bit;

axioms dummy = dummy == TRUE,
NewSequenceOfBit = s aor 1 == FALSE,
s apr i = NewSequenceOfBit == FALSE,
s apr i = s1 aor i1 == ((s=s1) and (i=i1));

axioms i aol NewSequenceOfBit == NewSequenceOfBit aor i,
i aol (s apr i1) == (i aol s) apr i1;

axiom seq(i) == NewSequenceOfBit apr i;

axioms NewSequenceOfBit join s == s,
(s apr i) join s1 == s join (i aol s1);

axiom LessFirst(s apr i) == if s = NewSequenceOfBit
then NewSequenceOfBit
else LessFirst(s) apr i;

axiom LessLast(s apr i) == s;

axiom isNewSequenceOfBit(s) == (s = NewSequenceOfBit);

axioms s1 subset (s apr i) == (s1 = NewSequenceOfBit or s1 subset s
or LessLast(s1) subseq s and (Last(s1) = i)),
s subset NewSequenceOfBit == (s = NewSequenceOfBit);

axioms i in NewSequenceOfBit == FALSE,
i in (s apr i1) == (i in s or (i=i1));

axioms s iss NewSequenceOfBit == (s = NewSequenceOfBit),
s1 iss (s2 apr i) == (s1 = NewSequenceOfBit or s1 iss s2
or (LessLast(s1) = s2) and (Last(s1) = i));

axioms Nchanges(NewSequenceOfBit) == 0,
Nchanges(s apr i) == if LastBit(s) = i
then \texttt{Nchanges}(s)
else \texttt{Nchanges}(s) + 1;

\textbf{axioms}
\begin{align*}
\text{Length}(\text{NewSequenceOfBit}) & = 0, \\
\text{Length}(s \text{ aor } i) & = \text{Length}(s) + 1;
\end{align*}

\textbf{axiom}
First(s \text{ aor } i) = \begin{cases} 
0 & \text{if } s = \text{NewSequenceOfBit} \\
n & \text{else First}(s);
\end{cases}

\textbf{axiom}
Last(s \text{ aor } i) = i;

\textbf{axioms}
\begin{align*}
\text{LastBit}(\text{NewSequenceOfBit}) & = \text{zero}, \\
\text{LastBit}(s \text{ aor } i) & = i;
\end{align*}

\textbf{rulelemmas}
\begin{align*}
\text{NewSequenceOfBit} = i \text{ apl } s & = \text{FALSE}, \\
i \text{ apl } s = \text{NewSequenceOfBit} & = \text{FALSE};
\end{align*}

\textbf{rulelemmas}
\begin{align*}
\text{s join } (s1 \text{ aor } i) & = (\text{s join } s1) \text{ aor } i, \\
\text{s join } \text{NewSequenceOfBit} & = \text{s}, \\
(i \text{ apl } s1) \text{ join } s2 & = i \text{ apl } (s1 \text{ join } s2), \\
(s \text{ join } (i \text{ apl } s1)) \text{ join } s2 & = s \text{ join } (i \text{ apl } (s1 \text{ join } s2)), \\
s \text{ join } (s1 \text{ join } s2) & = (s \text{ join } s1) \text{ join } s2;
\end{align*}

\textbf{rulelemma}
LessFirst(i \text{ apl } s) = s;

\textbf{rulelemma}
LessLast(i \text{ apl } s)
\begin{cases} 
= s & \text{if } s = \text{NewSequenceOfBit} \\
= i \text{ aol } \text{LessLast}(s) & \text{else}
\end{cases};

\textbf{rulelemmas}
\begin{align*}
\text{NewSequenceOfBit} \text{ subset } s & = \text{TRUE}, \\
\text{subset } s & = \text{TRUE};
\end{align*}

\textbf{rulelemma}
\begin{cases} 
1 \text{ in } (i \text{ apl } s) & = (1 \text{ in } s \text{ or } (i=11)),
\end{cases};

\textbf{rulelemmas}
\begin{align*}
\text{NewSequenceOfBit} \text{ iss } s & = \text{TRUE}, \\
\text{iss } s & = \text{TRUE};
\end{align*}

\textbf{rulelemma}
First(i \text{ aol } s) = i;

\textbf{rulelemma}
Last(i \text{ aol } s) = \begin{cases} 
\text{if } s = \text{NewSequenceOfBit} \\
\text{else Last}(s);
\end{cases}

\textbf{schemas}
\textbf{FirstInduction}(s)
\begin{align*}
= & \text{cases} (\text{Prop}(\text{NewSequenceOfBit}), \text{all } ss, ii \\
& \begin{cases} 
\text{IH}(ss) & \text{imp } \text{Prop}(ii \text{ apl } ss)
\end{cases})
\end{align*},

\text{Induction}(s)
AFFIRM Type Specifications and Proofs

== cases(Prop(NewSequenceOfBit), all ss, i1
    (IH(ss)
        imp Prop(ss apr i1)),
    NormalForm(s)
    == cases(Prop(NewSequenceOfBit), all ss, i1 (Prop(ss apr i1)));
end {SequenceOfBit} ;

7 U: print type sequenceOfPacket;
(sequenceOfPacket => SequenceOfPacket)

type SequenceOfPacket;

needs types Integer, Packet, SequenceOfBit, SequenceOfElementType;

declare dummy, ss, s, s1, s2, s3, s4, s5: SequenceOfPacket;
declare k, k1, k2: Integer;
declare ii, i, ii, i2, j: Packet;

interfaces NewSequenceOfPacket, s apr i, i apr s, seq(i), sl join s2, LessFirst(s), LessLast(s): SequenceOfPacket;

infix join, apr, apr;

interfaces isNewSequenceOfPacket(s), s1 subset s2, FirstInduction(s),
Induction(s), NormalForm(s), i in s, s1 iss s2, Repetats(s): Boolean

infix in, subset, iss;

interface Seqnums(s): SequenceOfBit;

interface UniqueSeq(s): SequenceOfElementType;

interface Length(s): Integer;

interfaces First(s), Last(s): Packet;

axioms dummy = dummy == TRUE,
    NewSequenceOfPacket = s apr i == FALSE,
    s apr i = NewSequenceOfPacket == FALSE,
    s apr i = sl apr i1 == (((s==s1) and (i==i1)));

axioms  i apr NewSequenceOfPacket == NewSequenceOfPacket apr i,
    i apr (s apr i1) == (i apr s) apr i1;

axiom seq(i) == NewSequenceOfPacket apr i;

axioms NewSequenceOfPacket join s == s,
    (s apr i) join sl == s join (i apr sl);

axiom LessFirst(s apr i)
== if s = NewSequenceOfPacket
then NewSequenceOfPacket
else LessFirst(s) aor i;

axiom
LessLast(s aor i) == s;

axiom
isNewSequenceOfPacket(s) == (s = NewSequenceOfPacket);

axioms
s1 subseq (s aor i) ==
  (s1 = NewSequenceOfPacket) or s1 subseq s
  or LessLast(s1) subseq s and (Last(s1) = i),
s subseq NewSequenceOfPacket == (s = NewSequenceOfPacket);

axioms
i in NewSequenceOfPacket == FALSE,
i in (s aor i1) == (i in s or (i = i1));

axioms
s iss NewSequenceOfPacket == (s = NewSequenceOfPacket),
s1 iss (s2 aor i) ==
  (s1 = NewSequenceOfPacket) or s1 iss s2
  or (LessLast(s1) = s2) and (Last(s1) = i);

axioms
Repeats(NewSequenceOfPacket) == TRUE,
Repeats(s aor i) ==
  (Repeats(s)
   and s = NewSequenceOfPacket
   and seqno(i) = seqno(Last(s))
   impl i = Last(s));

axioms
Segnums(NewSequenceOfPacket) == NewSequenceOfPacket,
Segnums(s aor i) == Segnums(s) aor seqno(i);

axioms
UniqueSeq(NewSequenceOfPacket) == NewSequenceOfElemType,
UniqueSeq(s aor i) ==
  if seqno(i) = LastBit(Segnums(s))
  then UniqueSeq(s)
  else UniqueSeq(s) aor nssq(i);

axioms
Length(NewSequenceOfPacket) == 0,
Length(s aor i) == Length(s) + 1;

axiom
First(s aor i) == if s = NewSequenceOfPacket
then i
else First(s);

axiom
Last(s aor i) == i;

rulelemmas
NewSequenceOfPacket = i apl s == FALSE,
i apl s = NewSequenceOfPacket == FALSE;

rulelemmas
s join (s1 aor i) == (s join s1) aor i,
s join NewSequenceOfPacket == s,
(i apl s1) join s2 == i apl (s1 join s2),
(s join (i and s1)) join s2
  == s join (i join (s1 join s2)),
  s join (s1 join s2) == (s join s1) join s2;

rulelemma LessFirst(i and s) == s;

rulelemma LessLast(i and s)
  == if s = NewSequenceOfPacket
     then NewSequenceOfPacket
     else i and LessLast(s);

rulelemmas NewSequenceOfPacket subset s == TRUE,
  s subset s == TRUE;

rulelemma i in (i1 and s) == (i in s or (i==i1));

rulelemmas NewSequenceOfPacket i ss s == TRUE,
  s i ss s == TRUE;

rulelemma First(i and s) == i;

rulelemma Last(i and s) == if s = NewSequenceOfPacket
  then i
  else Last(s);

schemas
  FirstInduction(s)
  == cases(Prop(NewSequenceOfPacket), all ss, ii
    ( IH(ss)
      imp Prop(ii and ss)),
    Induction(s)
    == cases(Prop(NewSequenceOfPacket), all ss, ii
      ( IH(ss)
        imp Prop(ss and or ii)),
    NormalForm(s)
    == cases(Prop(NewSequenceOfPacket), all ss, ii (Prop( ss
        and ii)));

end (SequenceOfPacket) ;

end ;

U: print type aspcontext;
(aspcontext => AspContext)

type AspContext;

needs types Bit, Integer, Packet, SequenceOfElemType, SequenceOfBit,
SequenceOfPacket, ElemType;

declare dummy: AspContext;
declare b, b1, b2: Bit;
declare i, i1, i2, j, k, k1, k2: Integer;
AFFIRM Type Specifications and Proofs

```
declare n, p1, p2, l6: Packet;
declare m, m1, m2: SequenceOfElemType;
declare n, n1, n2: SequenceOfBit;
declare s, s1, s2: SequenceOfPacket;
declare e, e1, e2: ElemType;

interfaces Rounded(k1, k2, j), MsgBag(m1, m2, j): Boolean;

axiom dummy=dummy == TRUE;

rulemmas Rounded(k, k, 1) == TRUE,
    Rounded(k, k+1, 1) == TRUE,
    Rounded(k+1, k, 1) == FALSE,
    Rounded(k1+1, k2+1, 1) == Rounded(k1, k2, 1);

rulemmas MsgBag(m, m, 1) == TRUE,
    "MsgBag(m, m mpr e, 1) == TRUE,
    "MsgBag(m apr e, m, 1) == FALSE;

define Rounded(k1, k2, j)
    == ((k1 <= k2) and (k2 <= j+k1)),
    MsgBag(m1, m2, j)
    == (m1 iss m2 and Rounded(Length(m1), Length(m2), j));

end {AppContext};
```

9 U: read abocontext, theorems;
(Reading AFFIRM commands from <DIVITU>AppContext,THEOREMS.13)

New environment:
```
dummy: AppContext
p, p1, p2: Bit
i, i1, i2, j, k, k1, k2: Integer
m, m1, m2: SequenceOfElemType
n, n1, n2: SequenceOfBit
s, s1, s2: SequenceOfPacket
e, e1, e2: ElemType
```

```
theorem SubToBag, s1 subset s2
    and Repeats(s2)
    and Rounded(#changes(Seqnums(s1)),
                #changes(Seqnums(s2)), 1)
    imp UniqueMsg(s1) \iff UniqueMsg(s2);

theorem PktsSubBound, s1 subset s2
    and Rounded(#changes(Seqnums(s1)),
                #changes(Seqnums(s2)) + 1, 1)
    imp Rounded(#changes(Seqnums(s1)),
                #changes(Seqnums(s2)), 1)
    and #changes(Seqnums(s1)) = #changes(Seqnums(s2));
```
theorem BitSubBound, nl subset n2
  and Bounded(nchanges(n1), nchanges(n2) + 1, 1)
  imp Bounded(nchanges(n1), nchanges(n2), 1)
  and nchanges(n1) = nchanges(n2);

theorem UniqueEq, UniqueSeq(s1) iss UniqueSeq(s2)
  and s1 subset s2
  and Bounded(nchanges(Seqnums(s1)), nchanges(Seqnums(s2)), 1)
  and LastBit(Seqnums(s1)) = LastBit(Seqnums(s2))
  imp UniqueSeq(s1) = UniqueSeq(s2);

theorem LastEq, nl subset n2
  and Bounded(nchanges(n1), nchanges(n2), 1)
  imp LastBit(n1) = LastBit(n2) eqv nchanges(n1) = nchanges(n2);

theorem SubLess, nl subset n2 imp nchanges(n1) <= nchanges(n2);

theorem SameLastBit, nl subset n2
  and b = LastBit(n2)
  and "((n1 apr n) subset n2)
  imp LastBit(n1) = LastBit(n2);

theorem SameLastBit2, s1 subset s2
  and p = Last(s2)
  and "((s1 apr p) subset s2)
  imp LastBit(Seqnums(s1)) = LastBit(Seqnums(s2));

theorem SubSeqnum, s1 subset s2 imp Seqnums(s1) subset Seqnums(s2);

theorem NcUnique, nchanges(Seqnums(s)) = length(UniqueSeq(s));

theorem LastSeq, s "= NewSequencePacket
  imp seqnum(Last(s)) = LastBit(Seqnums(s));

theorem NcNonempty, nchanges(n) > 0;

theorem BoundedAY1, Bounded(k1, k2, 1)
  eqv (k1=k2) or (k1+1 = k2);

theorem IssLenEq, m1 iss m2 and (length(m1) = length(m2))
  eqv m1=m2;

theorem IssLenLe, m1 iss m2 imp length(m1) <= length(m2);

theorem BiValued, (a "= b1) and (a "= b2)
  imp b1=b2;

10 U: print status;
The untried theorems are BitSubBound, BiValued, BoundedAY1, IssLenEq, IssLenLe,
  LastEq, LastSeq, NcNonempty, NcUnique, PktSubBound, SameLastBit, SameLastBit2,
AFFIRM Type Specifications and Proofs

Subless, SubSeqnum, SubToLag, and UniqueMsg.
No theorems are tried.
No theorems are assumed.
No theorems are awaiting lemma proof.
No theorems are proved.

11 U: try SubToLag;
SubToLag is untried.

all s1, s2
    (s1 subseq s2 and Repeats(s2)
     and Bounded(#changes(Seqnums(s1)), #changes(Seqnums(s2)), 1)
     imp UniqueMsg(s1) iss UniqueMsg(s2))

12 U: employ Induction(s2);
Case NewSequenceOfPacket: Prop(NewSequenceOfPacket) remains to be shown.
Case apr: all ss, ii (IH(ss) imp Prop(ss apr ii)) remains to be shown.
(NewSequenceOfPacket)
all s1
    (s1 = NewSequenceOfPacket and Bounded(#changes(Seqnums(s1)),
     0, 1)
     imp UniqueMsg(s1) = NewSequenceOfElemType)

13 U: replace:
TRUE
Going to leaf apr:

all ss", ii", s1
    (IH(ss", 1 {SubToLag})
     and (s1 = NewSequenceOfPacket or s1 subseq ss"
      or LastLast(s1) subseq ss" and (Last(s1) = ii")
      and Repeats(ss")
     and (ss" = NewSequenceOfPacket) and segno(ii")
      = segno(Last(ss")))
     imp ii" = Last(ss")
     imp if LastBit(Seqnums(ss")) = segno(ii")
     then Bounded(#changes(Seqnums(s1)), #changes(Seqnums(ss")), 1)
      imp UniqueMsg(s1) iss UniqueMsg(ss")
     else Bounded(#changes(Seqnums(s1)),
      #changes(Seqnums(ss")) + 1, 1)
      imp UniqueMsg(s1) = NewSequenceOfElemType
     or UniqueMsg(s1) iss UniqueMsg(ss")
      or LastLast(UniqueMsg(s1)) = UniqueMsg(ss")
      and Last(UniqueMsg(s1)) = mssq(ii")

14 U: split;
(first:
all ss", ii", s1
    (IH(ss", 1 {SubToLag}) and (s1 = NewSequenceOfPacket) and Repeats(
     ss")
     and (ss" = NewSequenceOfPacket) and segno(ii")
15 U: replace;

TRUE
Going to leaf second:

all ss", ii", sl (IH(ss", 1 (SupToLog)) and (sl "= NewSequencePacket)
and sl subseq ss" or LessLast(sl) subseq ss" and (Last(sl) = ii")
and Repeats(ss")
and (ss" "= NewSequencePacket) and segno(ii")

imp ii" = Last(ss")

16 U: suppose LastBit(Seqnums(ss")) = segno(ii");
(yes!)
all ss", ii", sl (LastBit(Seqnums(ss")) = segno(ii")
and IH(ss", 1 (SupToLog))
and sl "= NewSequencePacket
and sl subseq ss"
or LessLast(sl) subseq ss"
and Last(sl) = ii"
and Repeats(ss")
and (ss" "= NewSequencePacket
and segno(ii") = segno(Last(ss")))

imp ii" = Last(ss")

imp UniqueAsg(sl) iss UniqueAsg(ss")
or UniqueAsg(sl) iss UniqueAsg(ss")
and Last(Uniquesg(sl)) = UniqueAsg(ss")
and Last(Uniquesg(sl)) = mssg(ii")

imp it LastBit(Seqnums(ss")) = segno(ii")
then
Bounded(\text{Changes(Seqnums(sl))), Changes(Seqnums(ss")), 1)
imp UniqueAsg(sl) iss UniqueAsg(ss")
else
Bounded(Changes(Seqnums(sl)), Changes(Seqnums(ss")) + 1, 1)
imp UniqueAsg(sl) = NewSequenceElementType
or UniqueAsg(sl) iss UniqueAsg(ss")
or LessLast(Uniquesg(sl)) = UniqueAsg(ss")
and Last(Uniquesg(sl)) = mssg(ii")

17 U: split;
(spllit => split)
AFFIRM Type Specifications and Proofs

(1st)
all ss", ii", s1
  (LastBit(Seqnums(ss")) = seqo(ii")
   and IH(ss", 1 (SubToLag))
   and s1 "= NewSequenceOfPacket
   and s1 subset ss"
   and Repeats(ss")
   and ss" "= NewSequenceOfPacket
   and seqo(ii") = seqo(last(ss"))
   imp ii" = last(ss")
   and Bounded(chnum(Seqnums(s1)), chnum(Seqnums(ss")), 1)
   imp UniqueMsg(s1) iss UniqueMsg(ss")

18 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/1: s1" = s1
Proved by chaining and narrowing
using the substitution

s1" = s1

TRUE
Going to leaf second:

all ss", ii", s1
  (LastBit(Seqnums(ss")) = seqo(ii")
   and IH(ss", 1 (SubToLag))
   and s1 "= NewSequenceOfPacket
   and "(s1 subset ss"
   and lesslast(s1) subset ss"
   and last(s1) = ii"
   and Repeats(ss")
   and ss" "= NewSequenceOfPacket
   and seqo(ii") = seqo(last(ss"))
   imp ii" = last(ss")
   and Bounded(chnum(Seqnums(s1)), chnum(Seqnums(ss")), 1)
   imp UniqueMsg(s1) iss UniqueMsg(ss")

19 U: employ NormalForm(s1);
Case NewSequenceOfPacket: Prop(NewSequenceOfPacket) proven.
Case apr: all ss, ii (Prop(ss apr ii)) remains to be shown.
(apr:)
all ss", ii", ss, ii
  (LastBit(Seqnums(ss)) = seqo(ii)
   and IH(ss, 1 (SubToLag))
   and "((ss" apr ii") subset ss"
   and ss" subset ss
   and ii" = ii"
   and Repeats(ss)
   and ss" "= NewSequenceOfPacket
   and seqo(ii") = seqo(last(ss"))
\[ \text{affirm Type Specifications and Proofs} \]

\[ \text{imp ii = last(ss)} \]
\[ \text{imp if lastbit(\text{sequence}(ss'')) = seqno(ii'')} \]
\[ \text{then Bounded(\text{changes(\text{sequence}(ss''))}, \text{changes(\text{sequence}(ss))}, 1)} \]
\[ \text{else UniqueMsg(ss'') iss UniqueMsg(ss)} \]
\[ \text{else Bounded(\text{changes(\text{sequence}(ss'')}) + 1, \text{changes(\text{sequence}(ss)), 1})} \]
\[ \text{imp (UniqueMsg(ss'') apr msg(ii'')) iss UniqueMsg(ss}) \]

20 U: suppose;
(first:)
all ss'', ii'', ss, ii
(\text{lastbit(\text{sequence}(ss)) = seqno(ii)}
\text{and IH(ss, 1 \{\text{subToLog}\})}
\text{and "((ss'' apr ii'') subset ss)
\text{and ss'' subset ss}
\text{and ii'' = ii}
\text{and Repeats(\text{ss})}
\text{and ss = \text{newsequenceOfPacket}}
\text{imp if lastbit(\text{sequence}(ss'')) = seqno(ii'’)}
\text{then Bounded(\text{changes(\text{sequence}(ss''))}, \text{changes(\text{sequence}(ss)), 1})}
\text{else UniqueMsg(ss'') iss UniqueMsg(ss)}
\text{else Bounded(\text{changes(\text{sequence}(ss''))} + 1, \text{changes(\text{sequence}(ss)), 1})}
\text{imp (UniqueMsg(ss'') apr msg(ii'’)) iss UniqueMsg(ss})

21 U: undo;
suppose undone.

22 U: suppose lastbit(\text{sequence}(ss'')) = seqno(ii’’);
(yes:)
all ss'', ii'', ss, ii

(\text{lastbit(\text{sequence}(ss'')) = seqno(ii’’)}
\text{and lastbit(\text{sequence}(ss)) = seqno(ii)}
\text{and IH(ss, 1 \{\text{subToLog}\})}
\text{and "((ss'' apr ii'’) subset ss)
\text{and ss'' subset ss}
\text{and ii'' = ii}
\text{and Repeats(\text{ss})}
\text{and ss'' = \text{newsequenceOfPacket}}
\text{and seqno(ii) = seqno(last(ss))}
\text{imp ii = last(ss)}
\text{and Bounded(\text{changes(\text{sequence}(ss''))}, \text{changes(\text{sequence}(ss)), 1})}
\text{imp UniqueMsg(ss'') iss UniqueMsg(ss})

23 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/2: \text{sl = ss'' apr ii’’}
2/2: sl = ss''
Proved by chaining and narrowing
using the substitution

s1 = ss'

TRUE
Going to leaf no.

all ss", ii", ss, ii
   ( LastBit(Seqnum\(\text{ss}\)) \# eq seqno(ii) 
and LastBit(Seqnum\(\text{ss}\)) = seqno(ii) 
and IH(ss, 1 (SubToLag)) 
and "((ss' ap\(\text{r} \text{ ii}\)') subseq ss) 
and ss' subseq ss 
and ii" = ii 
and Repeats(ss) 
and ss' = NewSequenceOfPacket 
   and seqno(ii) = seqno(Last(ss)) 
   imp ii = Last(ss) 
and Rounded(\# changes(Seqnums\(\text{ss}\)) + 1, 
\# changes(Seqnums\(\text{ss}\)), 1) 
   imp (Un\#ique\#sq(ss') ap\(\text{r} \text{ mssq(ii)'})) iss Unique\#sq(ss))

24 U: replace ii'=

all ss", ii", ss, ii
   ( LastBit(Seqnum\(\text{ss}\)) \# eq seqno(ii) 
and LastBit(Seqnum\(\text{ss}\)) = seqno(ii) 
and IH(ss, 1 (SubToLag)) 
and "((ss' ap\(\text{r} \text{ ii}\)') subseq ss) 
and ss' subseq ss 
and ii" = ii 
and Repeats(ss) 
and ss' = NewSequenceOfPacket 
   and seqno(ii) = seqno(Last(ss)) 
   imp ii = Last(ss) 
and Rounded(\# changes(Seqnums\(\text{ss}\)) + 1, 
\# changes(Seqnums\(\text{ss}\)), 1) 
   imp (Un\#ique\#sq(ss') ap\(\text{r} \text{ mssq(ii)'})) iss Unique\#sq(ss))

25 U: apply LastSeq;

some s ((s is NewSequenceOfPacket) or (seqno(Last(s)) = LastBit(Seqnums(s)))
Automatically search for instantiation? no [confirm]

26 U: put s=ss;

all ss", ii", ss, ii
   (if ss = NewSequenceOfPacket 
then LastBit(Seqnums\(\text{ss}\)) \# eq seqno(ii) 
and LastBit(Seqnums\(\text{ss}\)) = seqno(ii) 
and IH(ss, 1 (SubToLag)) 
and "((ss' ap\(\text{r} \text{ ii}\)') subseq ss)
and $ss^*$ subseq $ss$
and $ii^* = ii$
and Repeats(ss)
and Rounded($\#changes(Sequms(ss^*)) + 1$, $\#changes(Sequms(ss)), 1$
imp (UniqueMsg(ss^*) and mssq(ii)) iss UniqueMsg(ss)
seqno(Next(ss)) = LastBit(Sequms(ss))
and LastBit(Sequms(ss^*)) = seqno(ii)
and LastBit(Sequms(ss)) = seqno(ii)
and IH(ss, 1 {SubToLog})
and "((ss' apr ii) subseq ss)
and $ss^*$ subseq ss
and $ii^* = ii$
and Repeats(ss)
and seqno(ii) = seqno(Next(ss))
imp $ii = Last(ss)$
and Rounded($\#changes(Sequms(ss^*)) + 1$, $\#changes(Sequms(ss)), 1$
imp (UniqueMsg(ss^*) and mssq(ii)) iss UniqueMsg(ss))

27 U: suppose;
(first:)
all $ss^*$, $ii^*$, $ss$, ii
( $ss = NewSequenceOfPacket$
and LastBit(Sequms(ss^*)) = seqno(ii)
and LastBit(Sequms(ss)) = seqno(ii)
and IH(ss, 1 {SubToLog})
and "((ss' apr ii) subseq ss)
and $ss^*$ subseq ss
and $ii^* = ii$
and Repeats(ss)
and Rounded($\#changes(Sequms(ss^*)) + 1$, $\#changes(Sequms(ss)), 1$
imp (UniqueMsg(ss^*) and mssq(ii)) iss UniqueMsg(ss))

28 U: replace $ss$;
all $ss^*$, $ii^*$, $ss$, ii
( $ss = NewSequenceOfPacket$
and LastBit(Sequms(ss^*)) = seqno(ii)
and zero = seqno(ii)
and IH(NewSequenceOfPacket, 1 {SubToLog})
and $ss^* = NewSequenceOfPacket$
and $ii^* = ii$
imp "Rounded($\#changes(Sequms(ss^*)) + 1$, 0, 1))

29 U: replace $ss^*$;
TRUE
Going to leaf second:.
all $ss^*$, $ii^*$, $ss$, ii
AFFIRM Type Specifications and Proofs

```
(ss := NewSequenceOfPacket
 and segno(last(ss)) = lastbit(senums(ss))
 and lastbit(senums(ss")) = segno(ii)
 and lastbit(senums(ss)) = segno(ii)
 and IH(ss, i {SubToEq})
 and "((ss' apr ii) subseq ss)
 and ss' subseq ss
 and ii" = ii
 and Repeats(ss)
 and segno(ii) = segno(last(ss))
 imp ii = last(ss)
 and Rounded(nchanges(senums(ss")) + 1,
 nchanges(senums(ss)), 1)
 imp (UniqueMsg(ss") and msgo(ii)) iis UniqueMsg(ss))

30 U: replace segno(ii);

all ss", ii", ss, ii
(s := NewSequenceOfPacket
 and segno(last(ss)) = lastbit(senums(ss))
 and lastbit(senums(ss")) = lastbit(senums(ss))
 and lastbit(senums(ss)) = segno(ii)
 and IH(ss, i {SubToEq})
 and "((ss" apr ii) subseq ss)
 and ss" subseq ss
 and ii" = ii
 and Repeats(ss)
 and ii = last(ss)
 and Rounded(nchanges(senums(ss")) + 1,
 nchanges(senums(ss)), 1)
 imp (UniqueMsg(ss") and msgo(ii)) iis UniqueMsg(ss))

31 U: apply SameLastBit2;

some s1, s2, n
(s1 subseq s2 and (p = last(s2))
 imp (s1 apr p) subseq s2 or (lastbit(senums(s1)) = lastbit(senums(s2)))

Automatically search for instantiation? yes [confirm]

1/4: (s2 = ss) and (s1 = ss" apr ii)
2/4: (s2 = ss) and (s1 = ss")
1/1: n = ii

Proved by chaining and narrowing
using the substitution

(s2 = ss) and (s1 = ss") and (n = ii)

TRUE

32 U: nxt;
(nxt => next)

Going to leaf no:
```
all $ss^*$, $ii^*$, $s1$

$$\text{lastBit(Sequums}(ss^*)) = \text{seqno}(ii^*)$$
and $\text{IH}(ss^*)$, 1 (SubToLag)
and $s1 = \text{NewSequenceOfPacket}$
and $s1 \text{ subseq } ss^*$
or $\text{LessLast}(s1) \text{ subseq } ss^*$
and $\text{Last}(s1) = ii^*$
and $\text{Repeats}(ss^*)$
and $ss^* = \text{NewSequenceOfPacket}$
and $\text{seqno}(ii^*) = \text{seqno}(\text{Last}(ss^*))$
imp $ii^* = \text{Last}(ss^*)$
and $\text{Bounded}(\text{Nchanges}(\text{Sequums}(s1)))$
$\text{Nchanges}(\text{Sequums}(ss^*)) + 1, 1$
imp $\text{UniqueSeq}(s1) = \text{NewSequenceOfELEMType}$
or $\text{UniqueSeq}(s1) \text{ iss UniqueSeq}(ss^*)$
or $\text{LessLast}(\text{UniqueSeq}(s1)) = \text{UniqueSeq}(ss^*)$
and $\text{Last}(\text{UniqueSeq}(s1)) = mss3(ii^*)$

33 U; split;
(first;)
all $ss^*$, $ii^*$, $s1$

$$\text{lastBit(Sequums}(ss^*)) = \text{seqno}(ii^*)$$
and $\text{IH}(ss^*)$, 1 (SubToLag)
and $s1 = \text{NewSequenceOfPacket}$
and $s1 \text{ subseq } ss^*$
and $\text{Repeats}(ss^*)$
and $ss^* = \text{NewSequenceOfPacket}$
and $\text{seqno}(ii^*) = \text{seqno}(\text{Last}(ss^*))$
imp $ii^* = \text{Last}(ss^*)$
and $\text{Bounded}(\text{Nchanges}(\text{Sequums}(s1)))$
$\text{Nchanges}(\text{Sequums}(ss^*)) + 1, 1$
imp $\text{UniqueSeq}(s1) = \text{NewSequenceOfELEMType}$
or $\text{UniqueSeq}(s1) \text{ iss UniqueSeq}(ss^*)$
or $\text{LessLast}(\text{UniqueSeq}(s1)) = \text{UniqueSeq}(ss^*)$
and $\text{Last}(\text{UniqueSeq}(s1)) = mss3(ii^*)$

34 U; apply PktSubBound;

some $s1^*$, $s2$
( $s1^* \text{ subseq } s2$
and $\text{Bounded}(\text{Nchanges}(\text{Sequums}(s1^*))), \text{Nchanges}(\text{Sequums}(s2)) + 1, 1$
imp $\text{Bounded}(\text{Nchanges}(\text{Sequums}(s1^*))), \text{Nchanges}(\text{Sequums}(s2)), 1$
and $\text{Nchanges}(\text{Sequums}(s1^*))) = \text{Nchanges}(\text{Sequums}(s2))$
Automatically search for instantiation? no [confirm]

35 U; put $s1^* = s1$, $s2 = ss^*$;

all $ss^*$, $ii^*$, $s1$
( $s1 \text{ subseq } ss^*$
and $\text{Bounded}(\text{Nchanges}(\text{Sequums}(s1)))$,)
```
\text{Nchanges}(\text{Seqnums}(ss')) + 1, 1)
\text{and Bounded}(\text{Nchanges}(\text{Seqnums}(s1)), \text{Nchanges}(\text{Seqnums}(ss')), 1)
\text{and Nchanges}(\text{Seqnums}(s1)) = \text{Nchanges}(\text{Seqnums}(ss'))
\text{and LastBit}(\text{Seqnums}(ss')) = \text{seqno}(ii')
\text{and IH}(ss', 1 \{\text{SubToLog}\})
\text{and s1} = \text{NewSequenceOfPacket}
\text{and Repeats}(ss')
\text{and ss'} = \text{NewSequenceOfPacket}
\text{and seqno}(ii') = \text{seqno}(\text{Last}(ss'))
\implies \text{ii'} = \text{Last}(ss')
\implies \text{UniqueMsg}(s1) = \text{NewSequenceOfElemType}
\text{or UniqueMsg}(s1) \text{ is } \text{UniqueMsg}(ss')
\text{or LessLast}(\text{UniqueMsg}(s1)) = \text{UniqueMsg}(ss')
\text{and Last}(\text{UniqueMsg}(s1)) = \text{msg}(ii')
```

36 UI: invoke IH;
Automatically search for instantiation? yes [confirm]

1/1: s1' = s1
Proved by chaining and narrowing
using the substitution

s1' = s1

TRUE
Going to leaf second:

all ss', ii', s1
( 
\text{LastBit}(\text{Seqnums}(ss')) = \text{seqno}(ii')
\text{and IH}(ss', 1 \{\text{SubToLog}\})
\text{and s1} = \text{NewSequenceOfPacket}
\text{and } (s1 \text{ subseq } ss')
\text{and LessLast}(s1) \text{ subseq } ss'
\text{and Last}(s1) = ii'
\text{and Repeats}(ss')
\text{and ss'} = \text{NewSequenceOfPacket}
\text{and seqno}(ii') = \text{seqno}(\text{Last}(ss'))
\implies \text{ii'} = \text{Last}(ss')
\implies \text{Bounded}(\text{Nchanges}(\text{Seqnums}(s1)), \text{Nchanges}(\text{Seqnums}(ss')) + 1, 1)
\implies \text{UniqueMsg}(s1) = \text{NewSequenceOfElemType}
\text{or UniqueMsg}(s1) \text{ is } \text{UniqueMsg}(ss')
\text{or LessLast}(\text{UniqueMsg}(s1)) = \text{UniqueMsg}(ss')
\text{and Last}(\text{UniqueMsg}(s1)) = \text{msg}(ii')
```

37 UI: employ NormalForm(s1);
Case NewSequenceOfPacket: Prop("NewSequenceOfPacket") proven.
Case apr: all ss, ii (Prop(ss apr ii)) remains to be shown.
(apr:)
all ss', ii', ss, ii
( 
\text{LastBit}(\text{Seqnums}(ss)) = \text{seqno}(ii)
\text{and IH}(ss, 1 \{\text{SubToLog}\})
```
and \((ss\text{ and } ii)\) subset ss
and ss' subset ss
and ii' = ii
and Repeats(ss)
and \(ss' = \text{newSequence}\uparrow\text{Packet}\)
and seqno(ii) = seqno(last(ss))
\[\text{imp } ii = \text{last}(ss)\]
\[\text{imp if lastBit}(\text{SeqNum}(ss')) = \text{seqno}(ii')\]
then Rounded(\(\text{changes}(\text{SeqNums}(ss'))\),
\(\text{changes}(\text{SeqNums}(ss)) + 1, 1\))
\[\text{imp } \text{UniqueMsg}(ss') = \text{newSequence}\uparrow\text{ElemType}\]
or \(\text{UniqueMsg}(ss')\) iss UniqueMsg(ss)
or \(\text{lessLast}(\text{UniqueMsg}(ss')) = \text{UniqueMsg}(ss)\)
and \(\text{last}(\text{UniqueMsg}(ss')) = \text{msg}(ii)\)
else Rounded(\(\text{changes}(\text{SeqNums}(ss'))\), \(\text{changes}(\text{SeqNums}(ss)), 1\))
\[\text{imp } (\text{UniqueMsg}(ss')\text{ and msg}(ii'))\text{ iss UniqueMsg}(ss)
or \(\text{UniqueMsg}(ss') = \text{UniqueMsg}(ss)\)
and \(\text{msg}(ii') = \text{msg}(ii)\)

38 U: Suppose lastBit(SeqNums(ss')) = seqno(ii');
(yes!)
all ss', ii', ss, ii
\((\text{lastBit}(\text{SeqNums}(ss')) = \text{seqno}(ii')\)
and lastBit(SeqNums(ss')) = seqno(ii)
and IH(ss, i (SubInLog))
and \((ss'\text{ and } ii')\) subset ss
and ss' subset ss
and ii' = ii
and Repeats(ss)
and \(ss' = \text{newSequence}\uparrow\text{Packet}\)
and seqno(ii) = seqno(last(ss))
\[\text{imp } ii = \text{last}(ss)\]
and Rounded(\(\text{changes}(\text{SeqNums}(ss'))\),
\(\text{changes}(\text{SeqNums}(ss)) + 1, 1\))
\[\text{imp } \text{UniqueMsg}(ss') = \text{newSequence}\uparrow\text{ElemType}\]
or \(\text{UniqueMsg}(ss')\) iss UniqueMsg(ss)
or \(\text{lessLast}(\text{UniqueMsg}(ss')) = \text{UniqueMsg}(ss)\)
and \(\text{last}(\text{UniqueMsg}(ss')) = \text{msg}(ii)\)

39 U: apply PktSubBound;

some s1, s2
\((s1\text{ subset } s2\)
and Rounded(\(\text{changes}(\text{SeqNums}(s1)), \text{changes}(\text{SeqNums}(s2)) + 1, 1\))
\[\text{imp } \text{changes}(\text{SeqNums}(s1)) = \text{changes}(\text{SeqNums}(s2))\]
and \(\text{changes}(\text{SeqNums}(s1)) = \text{changes}(\text{SeqNums}(s2))\)
Automatically search for instantiation? no [confirm]

40 U: put s1=ss', s2=ss;
all ss', ii', ss, ii
AFFIRM Type Specifications and Proofs

( ss' subset ss
and Bounded(\(\forall\text{changes}(\text{Sequms}(ss'))\),
   \(\text{changes}(\text{Sequms}(ss)) + 1, 1\))
and Bounded(\(\forall\text{changes}(\text{Sequms}(ss'))\), \(\forall\text{changes}(\text{Sequms}(ss))\), 1)
and \(\forall\text{changes}(\text{Sequms}(ss')) = \forall\text{changes}(\text{Sequms}(ss))\)
and LastBit(\text{Sequms}(ss')) = \text{seqno}(\text{ii'})
and LastBit(\text{Sequms}(ss)) = \text{seqno}(\text{ii})
and IH(ss', 1 \{\text{SubLog}\})
and "((ss' apr ii') subset ss)
and ii' = ii
and Repeats(ss)
and ss' = \text{NewSequenceOfPacket}
   and \text{seqno}(\text{ii}) = \text{seqno}(\text{Last}(ss))
   \text{imp} \text{ii} = \text{Last}(ss)
imp UniqueMsg(ss') = \text{NewSequenceOfElemType}
or UniqueMsg(ss') iss UniqueMsg(ss)
or LastLast(UniqueMsg(ss')) = UniqueMsg(ss)
and Last(UniqueMsg(ss')) = \text{msf}(\text{ii})

41 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/2: s1 = ss'
Proved by chaining and narrowing
using the substitution
s1 = ss'

TRUE
Going to leaf no:

all ss', ii', ss, ii
( \text{LastBit}(\text{Sequms}(ss')) = \text{seqno}(\text{ii'})
and \text{LastBit}(\text{Sequms}(ss)) = \text{seqno}(\text{ii})
and IH(ss', 1 \{\text{SubLog}\})
and "((ss' apr ii') subset ss)
and ss' subset ss
and ii' = ii
and Repeats(ss)
and ss' = \text{NewSequenceOfPacket}
   and \text{seqno}(\text{ii}) = \text{seqno}(\text{Last}(ss))
   \text{imp} \text{ii} = \text{Last}(ss)
and Bounded(\(\forall\text{changes}(\text{Sequms}(ss'))\), \(\forall\text{changes}(\text{Sequms}(ss))\), 1)
\text{imp} (UniqueMsg(ss') apr mssg(ii')) iss UniqueMsg(ss)
or UniqueMsg(ss') = UniqueMsg(ss)
and mssg(ii') = mssg(ii))

42 U: replace;
(replace => replace)

all ss', ii', ss, ii
( \text{LastBit}(\text{Sequms}(ss')) = \text{seqno}(ii)
and LastBit(Sequences(ss)) = seqno(ii)
and IH(ss, 1 {SubTolag})
and \((ss' \text{ and } ii) \subseteq \text{subseq } ss\)
and \(ss' \subseteq \text{subseq } ss\)
and \(ii' = ii\)
and Repeats(ss)
and \(ss' = \text{NewSequenceOfPacket}\)
and seqno(ii) = seqno(Last(ss))
implies \(ii = \text{Last}(ss)\)
and Bounded(\(|\text{changes}(\text{Sequences}(ss'))\|, \text{changes}(\text{Sequences}(ss)), 1\))
implies (UniqueMsg(ss') and \(\text{mseq}(ii)\) iss UniqueMsg(ss)
or UniqueMsg(ss') = UniqueMsg(ss))

43 U: apply bivalued;
(bivalued => Bivalued)

some b, b1, b2 ((b=b1) or (b=b2) or (b1=b2))
Automatically search for instantiation? no [confirm]

44 U: put b=seqno(ii), b1=LastBit(Sequences(ss')), b2=LastBit(Sequences(ss));

all ss', ii', ss, ii
( seqno(ii) = LastBit(Sequences(ss'))
and seqno(ii) = LastBit(Sequences(ss))
and LastBit(Sequences(ss')) = LastBit(Sequences(ss))
and IH(ss, 1 {SubTolag})
and \((ss' \text{ and } ii) \subseteq \text{subseq } ss\)
and \(ss' \subseteq \text{subseq } ss\)
and \(ii' = ii\)
and Repeats(ss)
and \(ss' = \text{NewSequenceOfPacket}\)
and seqno(ii) = seqno(Last(ss))
implies \(ii = \text{Last}(ss)\)
and Bounded(\(|\text{changes}(\text{Sequences}(ss'))\|, \text{changes}(\text{Sequences}(ss)), 1\))
implies (UniqueMsg(ss') and \(\text{mseq}(ii)\) iss UniqueMsg(ss)
or UniqueMsg(ss') = UniqueMsg(ss))

45 U: apply UniqueMsg;

some s1, s2
( UniqueMsg(s1) iss UniqueMsg(s2) and \(s1 \subseteq \text{subseq } s2\)
and Bounded(\(|\text{changes}(\text{Sequences}(s1))\|, \text{changes}(\text{Sequences}(s2)), 1\))
and LastBit(Sequences(s1)) = LastBit(Sequences(s2))
implies UniqueMsg(s1) = UniqueMsg(s2))
Automatically search for instantiation? no [confirm]

46 U: put s1=ss', s2=ss;

all ss', ii', ss, ii
( \((\text{UniqueMsg}(ss')) \text{ iss UniqueMsg}(ss)\)
and seqno(ii) = LastBit(Sequences(ss'))
and seqno(ii) = LastBit(Sequences(ss))

and LastBit(Seqnums(ss)) = LastBit(Seqnums(ss))
and IH(ss, i {SubToLag})
and "((ss' and i' subset ss)
and ss' subset ss
and i' = i'
and Repeats(ss)
and ss = NewSequenceOfPacket
and seqno(i') = seqno(Last(ss))
imp ii = Last(ss)
and Bounded("changes(Seqnums(ss'))", \changes(Seqnums(ss)), 1)
imp ("unique\msgs(ss') and msgl(ii) iss unique\msgs(ss)
or Unique\msgs(ss') = Unique\msgs(ss))

47 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/2: s1 = ss'
Proved by chaining and narrowing
using the substitution
s1 = ss'

TRUE
SubToLag is awaiting the proof of lemmas LastSeg, SameLastBit2, PktSubbound, Rivalued, and UniqueEq.
Going to lemma PktSubRound.
PktSubRound is untried.
all s1, s2
(s1 subset s2
and Bounded("changes(Seqnums(s1)), \changes(Seqnums(s2)) + 1, 1)
imp Bounded("changes(Seqnums(s1)), \changes(Seqnums(s2)), 1)
and \changes(Seqnums(s1)) = \changes(Seqnums(s2))

48 U: note
The nump has been passed but there still are a few nontrivial lemmas left. ;

49 U: freeze abp.frz;
writing file ... <DIVITO>ABP.FRZ.1

50 U: quit;
Automatically summarize the proof attempts? no [confirm]
Type CONTINUE to return to AFFIRM.

Transcript file <DIVITO>AFFLOG.12-FILE=R1.1
is open in the AFFIRM system <AFFIRM>AFFIRM.EXE.120
(<DIVITO>ABP.FRZ.1 , <AFFIRM>AFFIRM.EXE.120 )

50 U: print status;
The untried theorems are RtsSubRound, Rivalued, BoundedBy1, IssLenEQ, IssLenLE
, LastEq, LastSeg, \changes, \unique, PktSubRound, SameLastBit, SameLastBit2, SubLess, SubSegnum, and UniqueEq.
No theorems are tried.
No theorems are assumed.
The awaiting lemma proof theorem is SubStrong.
No theorems are proved.

51 U: print result;

all s1, s2
  ( s1 subset s2
      and Bounded(
        nchanges(Sequems(s1)), nchanges(Sequems(s2)) + 1, 1
      )
      impl
      Bounded(
        nchanges(Sequems(s1)), nchanges(Sequems(s2)), 1
      )
      and nchanges(Sequems(s1)) = nchanges(Sequems(s2)))

52 U: apply SubStrong;

some s1", s2" (s1" subset s2" impl Sequems(s1") subset Sequems(s2")( )
Automatically search for instantiation? no [confirm]

53 U: put s1"=s1, s2"=s2;

all s1, s2
  ( s1 subset s2 and Sequems(s1) subset Sequems(s2)
      and Bounded(
        nchanges(Sequems(s1)),
        nchanges(Sequems(s2)) + 1, 1
      )
      impl
      Bounded(
        nchanges(Sequems(s1)), nchanges(Sequems(s2)), 1
      )
      and nchanges(Sequems(s1)) = nchanges(Sequems(s2)))

54 U: invoke Bounded 1all;

all s1, s2
  ( s1 subset s2 and Sequems(s1) subset Sequems(s2)
      and nchanges(Sequems(s1)) <= nchanges(Sequems(s2)) + 1
      and nchanges(Sequems(s2)) <= nchanges(Sequems(s1))
      impl nchanges(Sequems(s1)) <= nchanges(Sequems(s2)))

55 U: apply Subless;

some n1, n2 (n1 subset n2 impl nchanges(n1) <= nchanges(n2))
Automatically search for instantiation? yes [confirm]

1/2: (n2 = Sequems(s2)) and (n1 = Sequems(s1))
Proved by chaining and narrowing
using the substitution

(n2 = Sequems(s2)) and (n1 = Sequems(s1))

TRUE
PktSubbound is awaiting the proof of lemmas SubSequem and Subless.

56 U: next;
Going to lemma Subless.
Subless is untried.
affirm type specifications and proofs

all n1, n2 (n1 subset n2) imp \( \# \text{changes}(n1) \leq \# \text{changes}(n2) \)

57 U: employ induction(n2);

case newsequenceofbit: prop(newsequenceofbit) remains to be shown.

case aor: all ss, i1 (IH(ss) imp prop(ss aor i1)) remains to be shown.

(newsequenceofbit)

all n1 (n1 = newsequenceofbit imp \# \text{changes}(n1) \leq 0)

58 U: replace;

true

going to leaf aor:

all ss, iis, n1

\( \text{IH}(ss, 6 \{\text{subless}\}) \) and (n1 = newsequenceofbit or n1 subset ss or lesslast(n1) subset ss and (last(n1) = iis)

imp if lastbit(sss) = iis

then \# \text{changes}(n1) \leq \# \text{changes}(sss)

else \# \text{changes}(n1) \leq \# \text{changes}(sss) + 1)

59 U: split;

(first:

all ss, iis, n1

\( \text{IH}(ss, 6 \{\text{subless}\}) \) and (n1 = newsequenceofbit)

imp if lastbit(sss) = iis

then \# \text{changes}(n1) \leq \# \text{changes}(sss)

else \# \text{changes}(n1) \leq \# \text{changes}(sss) + 1)

60 U: apply nchonned;

some n (0 \leq \# \text{changes}(n))

automatically search for instantiation? no [confirm]

61 U: put n = sss;

all ss, iis, n1

\( (0 \leq \# \text{changes}(ss)) \) and \( \text{IH}(ss, 6 \{\text{subless}\}) \)

and n1 = newsequenceofbit

imp if lastbit(sss) = iis

then \# \text{changes}(n1) \leq \# \text{changes}(sss)

else \# \text{changes}(n1) \leq \# \text{changes}(sss) + 1)

62 U: replace;

true

going to leaf second:

all ss, iis, n1

\( \text{IH}(ss, 6 \{\text{subless}\}) \) and (n1 \# newsequenceofbit)

and n1 subset ss or lesslast(n1) subset ss and (last(n1) = iis)
AFFIRM Type Specifications and Proofs

```plaintext
imp if LastBit(sss) = iis
  then \#changes(n1) <= \#changes(sss)
  else \#changes(n1) <= \#changes(sss) + 1

63 U: split;
  (first:)
all sss, iis, n1
  (IH(sss, 6 (SubLess)) and (n1 = NewSequenceOfBit) and n1 subset sss
    imp if LastBit(sss) = iis
    then \#changes(n1) <= \#changes(sss)
    else \#changes(n1) <= \#changes(sss) + 1)

64 U: invoke IH;
  Automatically search for instantiation? yes [confirm]

1/1: n1" = n1
Unsuccessful.

all sss, iis, n1 (some n1")
  (n1" subset sss imp \#changes(n1") <= \#changes(sss)
    and n1" = NewSequenceOfBit
    and n1 subset sss)
  imp if LastBit(sss) = iis
    then \#changes(n1) <= \#changes(sss)
    else \#changes(n1) <= \#changes(sss) + 1))

65 U: put n1"=n1;
TRUE
Going to leaf second:

all sss, iis, n1
  (IH(sss, 6 (SubLess)) and (n1 = NewSequenceOfBit)
    and "(n1 subset sss)
    and LessLast(n1) subset sss
    and Last(n1) = iis
    imp if LastBit(sss) = iis
    then \#changes(n1) <= \#changes(sss)
    else \#changes(n1) <= \#changes(sss) + 1)

66 U: employ NormalForm(n1);
Case NewSequenceOfBit: Prop(NewSequenceOfBit) proven.
Case apri: all sss, iis (Prop(sss apri iis)) remains to be shown.
apri:
all sss, iis, sss", iis"
  (IH(sss", 6 (SubLess)) and "((sss apri iis) subset sss")
    and sss subset sss"
    and iis=iis"
  imp if LastBit(sss") = iis"
    then if LastBit(sss) = iis
      then \#changes(sss) <= \#changes(sss")
      else \#changes(sss) < \#changes(sss")
```
Proofs of Supporting Lemmas in Gypsy

C1: true

Prvr-> $proceeding
last-repeats proved in theorem prover.

Exec->

% Now we need to stop and fill in one last specification function.
% The definition of "initial_subseq" was deliberately left pending
% to avoid certain unstable prover behavior when is was expanded.
% We now give it a definition in order to prove the lemmas which
% use it.

translate tty:

Gypsy Text: (terminate with "Z"

$extending scope alt-bit-specs =
begin

function initial_subseq (u, v: msg_seq) : boolean =
begin
  exit (assume result iff some s: msg_seq, u & s = v);
end;
end;

"Z

No syntax errors detected
No semantic errors detected

Exec-> set scope lemmas

Exec-> prove eq_iss
Entering Prover with lemma eq_iss
all u, v : msg_seq, u = v -> initial_subseq (u, v)

H1: u = v

->
C1: initial_subseq (u, v)
Backup point
(*)

Prvr-> expand initial_subseq
Backup point
(* E *)

Prvr-> theorem
Proofs of Supporting Lemmas in Gypsy

H1: \( u = v \)

\[\rightarrow\]

C1: \( u \oplus s \# 4s = v \)

Prv→ sProceeding
Ran out of tricks

Prv→ put

For what?* s\#4s;
Put what?* null(msg_seq);

For what?* sdone

Backup point
(. E . PUT .)

Prv→ sProceeding
eq_iss proved in theorem prover.

Exec→ prove eq_iss_app
Entering Prover with lemma eq_iss_app
all u, v : msg_seq,
    all m : message,
    \( u = v \rightarrow \text{initial_subseq}(u \oplus [seq: m], v \oplus [seq: m]) \)

H1: \( u = v \)

\[\rightarrow\]

C1: \( \text{initial_subseq}(u \oplus [seq: m], v \oplus [seq: m]) \)

Backup point
(.)

Prv→ expand initial_subseq
Backup point
(. E .)

Prv→ theorem

H1: \( u = v \)

\[\rightarrow\]

C1: \( u \oplus [seq: m] \oplus s \# 4s = v \oplus [seq: m] \)

Prv→ put

For what?* s\#4s:
Put what?* null(msg_seq);

For what?* sdone

Backup point
(. E . PUT .)
Proofs of Supporting lemmas in Gypsy

Prvr-> sProceeding
Conclusion simplified to:
u @ [seq: m] = v @ [seq: m]
Ran out of tricks

Prvr-> theorem

H1: u = v
->
C1: u @ [seq: m] @ null (msg_seq) = v @ [seq: m]

Prvr-> equality substitute h1
H1 can be solved for:
T1: u
T2: v
which term (by label) do you want to substitute for?
t1

u := v
OK??
y
Backup point
(., E . PUT . =S .)

Prvr-> sProceeding
eq-iss-app proved in theorem prover.

Exec-> prove iss-app
Entering Prover with lemma iss-app
all u, v : msg_seq,
all m : message,
initial_subseq (u, v) -> initial_subseq (u, v @ [seq: m])

H1: initial_subseq (u, v)
->
C1: initial_subseq (u, v @ [seq: m])
Backup point
(., )

Prvr-> expand initial_subseq
More than one to expand.
which do you want to expand? all
Backup point
(., E .)

Prvr-> theorem

H1: u @ s#2 = v
->
C1: u @ s#7s = v @ [seq: m]

Prvr-> simplify theorem
Proofs of Supporting Lemmas in Gypsy

Prvr->theorem

H1: u @ s#2 = v
->
C1: u @ s#7s = v @ [seq: m]

Prvr-> equality substitute h1
H1 yields
v := u @ s#2
Backup point
( . E . =S . )

Prvr->theorem

H1: true
->
C1: u @ s#2 @ [seq: m] = u @ s#7s

Prvr-> sProceeding
iss_app proved in theorem prover.

Exec-> prove iss_trans
Entering Prover with lemma iss_trans
all u, v, w : msg_seq,
   initial_subseq (u, v) & initial_subseq (v, w)
   -> initial_subseq (u, w)

H1: initial_subseq (u, v)
H2: initial_subseq (v, w)
->
C1: initial_subseq (u, w)
Backup point
( . )

Prvr-> expand initial_subseq
More than one to expand.
Which do you want to expand? all
Backup point
( . E . )

Prvr->simplify theorem

Prvr->theorem

H1: u @ s#2 = v
H2: v @ s#3 = w
->
C1: u @ s#10s = w

Prvr-> sProceeding
::: Ran out of tricks
Proofs of Supporting Lemmas in Gypsy

Prvr=> equality substitute h1
H1 yields
   v := u @ s#2
Backup point
   (. E . =S .)

Prvr=> theorem

   H1: u @ s#2 @ s#3 = w
->
   C1: u @ s#10s = w

Prvr=> equality substitute h1
H1 yields
   w := u @ s#2 @ s#3
Backup point
   (. E . =S . =S .)

Prvr=> theorem

   H1: true
->
   C1: u @ s#2 @ s#3 = u @ s#10s

Prvr=> sProceeding
iss-trans proved in theorem prover.

Exec=> prove msglag_eq
Entering Prover with lemma msglag_eq
all u, v : msg_seq, u = v iff msglag (u, v, 0)

   C1: u = v iff msglag (u, v, 0)
Backup point
   (.)

Prvr=> expand msglag
Backup point
   (. E .)

Prvr=> simplify theorem

Prvr=> theorem

   C1: size (u) = size (v) & initial_subseq (u, v) iff u = v

Prvr=> sProceeding
Backup point
   (. E . IFFC .)

Prvr=> theorem
Proofs of Supporting Lemmas in Gypsy

C1: \text{size}(u) = \text{size}(v) \& \text{initial_subseq}(u, v) \rightarrow u = v
C2: u = v \rightarrow \text{size}(u) = \text{size}(v) \& \text{initial_subseq}(u, v)

Prvr=\rightarrow \text{superceeding}
Backup point
(\ast, E, IFFC, 1, )

Prvr=\rightarrow \text{theorem}

C1: \text{size}(u) = \text{size}(v) \& \text{initial_subseq}(u, v) \rightarrow u = v

Prvr=\rightarrow \text{superceeding}
Typelist equalitites

\text{size}(v) = \text{size}(u)
Backup point
(\ast, E, IFFC, 1, P\rightarrow , )

Prvr=\rightarrow \text{theorem}

H1: \text{size}(v) = \text{size}(u)
H2: \text{size}(u) = \text{size}(v)
H3: \text{initial_subseq}(u, v)

\rightarrow

C1: u = v

Prvr=\rightarrow \text{expand initial_subseq}
Backup point
(\ast, E, IFFC, 1, P\rightarrow , E, )

Prvr=\rightarrow \text{simplify theorem}

Prvr=\rightarrow \text{theorem}

H1: u \# s\#2 = v
H2: \text{size}(u) = \text{size}(v)
H3: \text{size}(v) = \text{size}(u)

\rightarrow

C1: u = v

Prvr=\rightarrow \text{use size_null}
Backup point
(\ast, E, IFFC, 1, P\rightarrow , E, U, )

Prvr=\rightarrow \text{theorem}

H1: \text{null}(\text{msg_seq}) = u\#2s \; \text{iff} \; 0 = \text{size}(u\#2s)
H2: u \# s\#2 = v
H3: \text{size}(u) = \text{size}(v)
H4: \text{size}(v) = \text{size}(u)

\rightarrow

C1: u = v
Proofs of Supporting Lemmas in Gypsy

Prv:→ sProceeding
:::...Ran out of tricks

Prv:→ put

For what? u#2$:
Put what? s#2;

For what? sdone

Backup point

Prv:→ theorem

H1: null (msg_seq) = s#2 iff 0 = size (s#2)
H2: size (v) = size (u)
H3: size (u) = size (v)
H4: u @ s#2 = v

→
C1: u = v

Prv:→ simplify theorem

Prv:→ theorem

H1: u @ s#2 = v
H2: size (u) = size (v)
H3: size (v) = size (u)
H4: null (msg_seq) = s#2 iff 0 = size (s#2)

→
C1: u = v

Prv:→ equality substitute h1
H1 yields
v := u @ s#2
Backup point

Prv:→ theorem

H1: null (msg_seq) = s#2
H2: 0 = size (s#2)

→
C1: u @ s#2 = u

Prv:→ simplify theorem

Prv:→ theorem

C1: true
Proofs of Supporting Lemmas in Gypsy

Prvr→ sProceeeding
( E . IFPC . 1 .)

size (u) = size (v) & initial-subseq (u, v) → u = v
Proved

Prvr→ sProceeeding
Backup point
( E . IFPC . 2 .)

Prvr→ theorem

C1: u = v → size (u) = size (v) & initial-subseq (u, v)

Prvr→ sProceeeding
Backup point
( E . IFPC . 2 . P→ .)

Prvr→ theorem

H1: u = v

→

C1: size (u) = size (v)
C2: initial-subseq (u, v)

Prvr→ equality substitute h1
H1 can be solved for:
T1: u
T2: v
Which term (by label) do you want to substitute for?
T1

u := v
OK??
Y
Backup point
( E . IFPC . 2 . P→ . =S .)

Prvr→ sProceeeding
Ran out of tricks

Prvr→ theorem

H1: true

→

C1: initial-subseq (v, v)

Prvr→ use eq-iss
Backup point
( E . IFPC . 2 . P→ . =S . u .)
Proofs of Supporting Lemmas in Gypsy

Prvr=> theorem

H1: b#2s = one or b*2s = zero
H2: comp (b1) = b2
->
   C1: b1 ne b2

Prvr=> expand comp
Backup point
( . IFFC . 1 . P-> . U . E . )

Prvr=> simplify theorem

Prvr=> theorem

H1: if b1 = zero then one else zero f1 = b2
H2: b#2s = one or b*2s = zero
->
   C1: b1 ne b2

Prvr=> put

For what?* b#2s;
Put what?* b1
* ;

For what?* $done
Backup point
( . IFFC . 1 . P-> . U . E . PUT . )

Prvr=> simplify theorem

Prvr=> theorem

H1: if b1 = zero then one else zero f1 = b2
H2: b1 = one or b1 = zero
->
   C1: b1 ne b2

Prvr=> qed
( . IFFC . 1 . P-> . U . E . PUT . QED )
:::
   b2 := if b1 = zero then one else zero f1
( . IFFC . 1 . P-> . U . E . PUT . QED =S )
Attempting CASE1

   b1 = one
( . IFFC . 1 . P-> . U . E . PUT . QED =S CASE1 )
Ran out of tricks
Proofs of Supporting Lemmas in Goby

Prvr-> theorem

H1: b1 = one

=> C1: if b1 = zero then one else zero fi ne b1

Prvr-> simplify theorem

Prvr-> theorem

H1: b1 = one

=> C1: if b1 = zero then one else zero fi ne b1

Prvr-> equality substitute h1

H1 can be solved for:

T1: b1
T2: one

which term (by label) do you want to substitute for?

b1 := one

OK??

y

Backup point

(.*IFFC . 1 . P-> . U . E . PUT . QED =S CASE2 =S .)

Prvr-> $proceeding

(.*IFFC . 1 . P-> . U . E . PUT . QED =S CASE1 .)

b1 = one -> if b1 = zero then one else zero fi ne b1

Proved

Attempting CASE2

b1 = zero

Backup point

(.*IFFC . 1 . P-> . U . E . PUT . QED =S CASE2 .)

Prvr-> theorem

H1: b1 = zero

=> C1: if b1 = zero then one else zero fi ne b1

Prvr-> equality substitute h1

H1 can be solved for:

T1: b1
T2: zero

which term (by label) do you want to substitute for?

b1
Proofs of Supporting Lemmas in Gypsy

b1 := zero
OK??

Backup point

Prv⇒ sProceeding

b1 = zero ⇒ if b1 = zero then one else zero f1 ne b1

Proved
Done with cases
QED

Prv⇒ sProceeding
(* IFFC . 1 . )

cmp (b1) = b2 ⇒ b1 ne b2

Proved

Prv⇒ sProceeding
Backup point
(* IFFC . 2 . )

Prv⇒ theorem

C1: b1 ne b2 ⇒ cmp (b1) = b2

Prv⇒ sProceeding
Backup point
(* IFFC . 2 . P⇒ . )

Prv⇒ theorem

H1: b1 ne b2
⇒

C1: cmp (b1) = b2

Prv⇒ expand cmp
Backup point
(* IFFC . 2 . P⇒ . E . )

Prv⇒ simplify theorem

Prv⇒ theorem

H1: b1 ne b2
⇒

C1: if b1 = zero then one else zero f1 = b2

Prv⇒ use bit_cases
Proofs of Supporting Lemmas in Gypsy

Backup point
(. IFFC . 2 . P-> . E . U . )

Prvr-> hypothesis

H1: b#4$ = one or b#4$ = zero
H2: b1 ne b2

Prvr-> put

For what?* b#4$;
Put what?* b1;

For what?* sdone

Backup point
(. IFFC . 2 . P-> . E . U . PUT . )

Prvr-> theorem

H1: b1 = one or b1 = zero
H2: b1 ne b2

->
C1: if b1 = zero then one else zero fi = b2

Prvr-> qed
(. IFFC . 2 . P-> . E . U . PUT . QED)
::: Attempting CASE1

:::
(. IFFC . 2 . P-> . E . U . PUT . QED CASE1)
::: Ran out of tricks

Prvr-> theorem

H1: b1 = one
H2: b1 ne b2

->
C1: if b1 = zero then one else zero fi = b2

Prvr-> equality substitute h1
41 yields
b1 := one
Backup point
(. IFFC . 2 . P-> . E . U . PUT . QED CASE1 =S . )

Prvr-> sProceeding
Ran out of tricks

Prvr-> theorem

H1: b2 ne one
Proofs of Supporting Lemmas in Gypsy

=>

C1: b2 = zero

PrvR => use bit_cases
Backup point
( . IFFC . 2 . P-> . E . U . PUT . QED CASE1 =S . U . )

PrvR => theorem

H1: b#68 = one or b#68 = zero
H2: b2 ne one

=>

C1: b2 = zero

PrvR => put

For what?* b#68;
Put what?* b2;

For what?* sdone

Backup point
( . IFFC . 2 . P-> . E . U . PUT . QED CASE1 =S . U . )

PrvR => sProceeding

::: Attempting CASE1

b2 = one
Backup point
( . IFFC . 2 . P-> . E . U . PUT . QED CASE1 =S . U . )

PrvR => sProceeding

( . IFFC . 2 . P-> . E . U . PUT . QED CASE1 =S . U . )

b2 = one => b2 = zero
Proved

Attempting CASE2

b2 = zero
Backup point
( . IFFC . 2 . P-> . E . U . PUT . QED CASE1 =S . U . )

PrvR => sProceeding

( . IFFC . 2 . P-> . E . U . PUT . QED CASE1 =S . U . )

b2 = zero => b2 = zero
Proved

Done with cases

( . IFFC . 2 . P-> . E . U . PUT . QED CASE1 . )

b1 = one => if b1 = zero then one else zero f1 = b2
Proved
Proofs of Supporting Lemmas in Gypsy

Attempting CASE2

\[ b_1 = \text{zero} \]
Backup point
\[ (\text{IFFC } 2 \text{ P=} E \text{ U PUT QED CASE2.)} \]

Prvr=>$ s$Proceeding
:::Ran out of tricks

Prvr=>$ \text{theorem}$

\[ H_1: \ b_1 = \text{zero} \]
\[ H_2: \ b_1 \text{ ne } b_2 \]
\[ \Rightarrow \]
\[ C_1: \text{if } b_1 = \text{zero then one else zero } f_1 = b_2 \]

Prvr=>$ \text{equality substitute } n_1$

H1 yields
\[ b_1 := \text{zero} \]
Backup point
\[ (\text{IFFC } 2 \text{ P=} E \text{ U PUT QED CASE2.)} = S, ) \]

Prvr=>$ s$Proceeding
Ran out of tricks

Prvr=>$ \text{theorem}$

\[ H_1: \ b_2 \text{ ne zero} \]
\[ \Rightarrow \]
\[ C_1: \ b_2 = \text{one} \]

Prvr=>$ \text{use bit\_cases}$
Backup point
\[ (\text{IFFC } 2 \text{ P=} E \text{ U PUT QED CASE2.)} = S, U, ) \]

Prvr=>$ \text{hypothesis}$

\[ H_1: \ b\#S = \text{one or } b\#S = \text{zero} \]
\[ H_2: \ b_2 \text{ ne zero} \]

Prvr=>$ \text{put}$

For what?* $b\#S$;
Put what?* $b_2$;

For what?* sdone

Backup point
\[ (\text{IFFC } 2 \text{ P=} E \text{ U PUT QED CASE2.)} = S, U, PUT .) \]

Prvr=>$ s$Proceeding
:::Attempting CASE1
Proofs of Supporting Lemmas in Gypsy

b2 = one
Backup point

Prvr-> sProceeding

b2 = one -> b2 = one
Proved
Attempting CASE2

b2 = zero
Backup point

Prvr-> sProceeding

b2 = zero -> b2 = one
Proved
Done with cases
(, IFFC . 2 . P-> . E . U . PUT . QED CASE2 .)

b1 = zero -> if b1 = zero then one else zero fi = b2
Proved
Done with cases
QED

Prvr-> sProceeding
(, IFFC . 2 .)

b1 ne b2 -> comp (b1) = b2
Proved

Prvr-> sProceeding
comp_ne proved in theorem prover.

Exec-> prove hist_sub
Entering Prover with lemma hist_sub
all x, y : pkt_buf,
    allfrom (x) sub allfrom (y) -> allfrom (x) sub allto (y)

C1: true
Backup point
()

Prvr-> sProceeding
hist_sub proved in theorem prover.

Exec->
% The following lemma constitutes the entire basis for the proof.

prover bit_cases
Entering Prover with lemma bit_cases
all b : bit, b = one or b = zero

C1: b = one or b = zero
Backup point
(*)

prvr=> simplify theorem
prvr=> theorem

C1: b = one or b = zero

prvr=>

% This lemma must be assumed since the prover does not have the
% information needed about the enumeration type "bit".

assume

Select assumed subgoal name bit_cases

Enter justification (terminate with ESCape)
see above.
$bit_cases proved in theorem prover.

exec=>

% The following lemmas were once needed but no longer are
% due to improvements in the simplifier.

prover app_pktnonnull
Entering Prover with lemma app_pktnonnull
true

C1: true
Backup point
(*)

prvr=> sProceeding
app_pktnonnull proved in theorem prover.

exec=> prove app_msgnonnull
Entering Prover with lemma app_msgnonnull
true
Proofs of Supporting Lemmas in Gypsy

C1: true
Backup point
(,)

PrvR-> $proceeding
app_msg_nonnull proved in theorem prover.

Exec-> prove app_bit_nonnull
Entering Prover with lemma app_bit_nonnull true

C1: true
Backup point
(,)

PrvR-> $proceeding
app_bit_nonnull proved in theorem prover.

Exec->

% Finally, we come to the remaining set of lemmas which are those
% proved under the Affirm system. Their definitions have noted
% that they are to be assumed so they will just fall through the
% prover here.

prove sub_app
Entering Prover with lemma sub_app true

C1: true
Backup point
(,)

PrvR-> $proceeding
sub_app proved in theorem prover.

Exec-> prove size_null
Entering Prover with lemma size_null true

C1: true
Backup point
(,)

PrvR-> $proceeding
size_null proved in theorem prover.

Exec-> prove sub-seqnum
Entering Prover with lemma sub-seqnum true
Proofs of Supporting Lemmas in Gypsy

C1: true
Backup point
(•)

Prvr->$proceeding
sub_seqnum proved in theorem prover.

Exec-> prove sub_seqnum
Entering Prover with lemma sub_seqnum true

C1: true
Backup point
(•)

Prvr->$proceeding
sub_seqnum proved in theorem prover.

Exec-> prove sub_nchanges
Entering Prover with lemma sub_nchanges true

C1: true
Backup point
(•)

Prvr->$proceeding
sub_nchanges proved in theorem prover.

Exec-> prove nchanges_unique
Entering Prover with lemma nchanges_unique true

C1: true
Backup point
(•)

Prvr->$proceeding
nchanges_unique proved in theorem prover.

Exec-> prove sub_to_lag
Entering Prover with lemma sub_to_lag true

C1: true
Backup point
(•)

Prvr->$proceeding
sub_to_lag proved in theorem prover.
Proofs of Supporting Lemmas in Gypsy

Exec-> show status all

The current design and verification status is:

SCOPE ALT_BIT_PROTOCOL

Waiting for pending body to be filled in: MEDIUM, TIMER

Proved: AB_PROTOCOL, COMP, RECEIVER, SENDER

Types, constants: BIT, BIT_SEQ, CLK_BUF, MESSAGE, MSG_BUF, MSG_SEQ, PACKET, PKT_BUF, PKT_SEQ

SCOPE LEMMAS

Proved: APB_1, APP_BIT_NONNULL, APP_MSG_NONNULL, APP_PKT_NONNULL, BIT_CASES, COMP_NE, EQ_ISS, EQ_ISS_APP, HIST_SUB, ISS_APP, ISS_TRANS, INTERPOLATE, LAST_NEXT, LAST_UNIQUE, LAST_REPEATS, MAIN_LEMMA, MSG_LAG_EQ, NE_NEXT, NEXT_COMP, NCHANGES_UNIQUE, PROP_REC_1, PROP_REC_2, PROP_REC_3, PROP_REC_4, PROP_REC_5, PROP_TRANS_1, PROP_TRANS_2, PROP_TRANS_3, PROP_TRANS_4, PROP_TRANS_5, PROP_TRANS_6, PROP_TRANS_7, SUB_APP, SIZE_NULL, SUB_SEQNUM, SUB_TO_LAG, SUB_NCHANGES

*~B~*

SCOPE ALT_BIT_SPECS

For specifications only: INITIAL_SUBSEQ, LAST_BIT, MSG_LAG, NCHANGES, NEXT_SEQNUM, PROPER_RECEPTION, PROPER_TRANSMISSION, REPEATS, SEQNUMS, UNIQUE_MSG

Exec-> save abv.dmp

File ABV.DMP already exists. Rewrite it? -> y

Saving.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
Exec->

% This ends the Gypsy part of our proof of the Alternating Bit Protocol.
% Be sure to tune in next time when we try to prove bigger and better
% protocols.

[HCOPY terminated at Sat 9-May-81 14:38:34]
Transcript file <DIVITO>AFFIRM.TRANSCRIPT.R=FEB-81.1 is open in the AFFIRM system <AFFIRM>AFFIRM.EXE.120

1 U: load appcontext;
   file created for AFFIRM on 31-Jan-81 15:27:15
   ABPCONTEXT.COMS
   file created for AFFIRM on 4-Feb-81 21:48:05
   BITCOMS
   <DIVITO>BIT..2
   file created for AFFIRM on 8-Jan-81 22:00:35
   PACKETCHANS
   compiled for AFFIRM on 20-Dec-80 15:42:13
   file created for AFFIRM on 20-Dec-80 15:42:10
   ELEMTYPECOMS

(File created under Affirm 121)
<PVLIBRARY>ELEMTYPE.COM.4<DIVITO>PACKET..2
   file created for AFFIRM on 31-Jan-81 15:26:55
   SEQUENCEOFELEMTYPE.COMS
   <DIVITO>SEQUENCEOFELEMTYPE..5
   file created for AFFIRM on 31-Jan-81 15:25:36
   SEQUENCEOFBITCOMS
   <DIVITO>SEQUENCEOFBIT..5
   file created for AFFIRM on 7-Feb-81 06:44:08
   SEQUENCEOFPACKETCHANS
   <DIVITO>SEQUENCEOFPACKET..7<DIVITO>ABPCONTEXT..3

2 U: print type elemtype;
   (elemtype => ElemType)

type ElemType;

declare dummy: ElemType;
axiom dummy=dummy == TRUE;
end (ElemType);

3 U: print type bit;
   (bit => Bit)

type Bit;

declare dummy, b: Bit;
interfaces zero, one, comp(b): Bit;
interface NormalForm(b): Boolean;
axioms dummy=dummy == TRUE,
zero = one == FALSE,
AFFIRM Type Specifications and Proofs

one = zero == FALSE;

axioms
  comp(zero) == one,
  comp(one) == zero;

schema
  NormalForm(n) == cases(Prop(zero), Prop(one));

end {Bit} ;

4 U: print type packet;
(packet => Packet)

type Packet;

needs types Bit, ElemType;

declare
dummy, p: Packet;

interface
  mssq(p): ElemType;

interface
  seqno(p): Bit;

axiom
dummy=dummy == TRUE;

end {Packet} ;

5 U: print type sequenceofElemType;
(sequenceofElemType => SequenceofElemType)

type SequenceofElemTyne;

needs types Integer, ElemType;

declare
dummy, ss, s, s1, s2, s3, s4, s5: SequenceofElemType;

declare
  k, k1, k2: Integer;

declare
  ii, i, i1, i2, j: ElemType;

interfaces
  NewSequenceofElemType, s aor i, i apl s, seq(i),
  s1 join s2, LessFirst(s), LessLast(s): SequenceofElemType;

infix
  join, apl, aor;

interfaces
  IsNewSequenceofElemType(s), s1 subseq s2, FirstInduction(s),
  Induction(s), NormalForm(s), i in s, s1 iss s2: Boolean;

infix
  in, subseq, iss;

interface
  Length(s): Integer;

interfaces
  First(s), Last(s): ElemType;

axioms
dummy=dummy == TRUE,
NewSequenceOfElementType = s apr i == FALSE, s apr i = NewSequenceOfElementType == FALSE, s apr i = s1 apr i1 == ((s=s1) and (i=i1));

axioms
i apr NewSequenceOfElementType == NewSequenceOfElementType apr i, i aprl (s apr i1) == (i aprl s) apr i1;

axiom
seq(i) == NewSequenceOfElementType apr i;

axioms
NewSequenceOfElementType join s == s, (s apr i) join s1 == s join (i aprl s1);

axiom
LessFirst(s apr i) 
== if s = NewSequenceOfElementType 
then NewSequenceOfElementType 
else LessFirst(s) apr i;

axiom
LessLast(s apr i) == s;

axiom
isNewSequenceOfElementType(s) == (s = NewSequenceOfElementType);

axioms
s1 subseq (s apr i) 
== ( (s1 = NewSequenceOfElementType) or s1 subseq s 
or LessLast(s1) subseq s and (Last(s1) = i)) , s subseq NewSequenceOfElementType == (s = NewSequenceOfElementType);

axioms
i in NewSequenceOfElementType == FALSE, i in (s apr i1) == (i in s or (i=i1));

axioms
s iss NewSequenceOfElementType == (s = NewSequenceOfElementType), s1 iss (s2 apr i) 
== ( (s1 = NewSequenceOfElementType) or s1 iss s2 
or (LessLast(s1) = s2) and (Last(s1) = i));

axioms
Length(NewSequenceOfElementType) == 0, 
Length(s apr i) == Length(s) + i;

axiom
First(s apr i) == if s = NewSequenceOfElementType 
then i 
else First(s);

axiom
Last(s apr i) == i;

rulelemmas
NewSequenceOfElementType = i aprl s == FALSE, i aprl s = NewSequenceOfElementType == FALSE;

rulelemmas
s join (s1 apr i) == (s join s1) apr i, s join NewSequenceOfElementType == s, (i aprl s1) join s2 == i aprl (s1 join s2), (s join (i aprl s1)) join s2 
== s join (i aprl (s1 join s2)), s join (s1 join s2) == (s join s1) join s2;
AFFIRM Type Specifications and Proofs

rulelemma LessFirst(i apl s) == s;

rulelemma LessLast(i apl s)
  == if s = NewSequenceOfElementType
    then NewSequenceOfElementType
    else i apl LessLast(s);

rulelemmas NewSequenceOfElementType subseq s == TRUE,
  s subseq s == TRUE;

rulelemma i in (i1 apl s) == (i in s or (i=i1));

rulelemmas NewSequenceOfElementType iss s == TRUE,
  s iss s == TRUE;

rulelemma First(i apl s) == i;

rulelemma Last(i apl s) == if s = NewSequenceOfElementType
  then i
  else Last(s);

schemas FirstInduction(s)
  == cases(Prop(NewSequenceOfElementType), all ss, ii
    ( IH(ss)
      imp Prop(
        apl ss))),

Induction(s)
  == cases(Prop(NewSequenceOfElementType), all ss, ii
    ( IH(ss)
      imp Prop(
        apr ii))),

NormalForm(s)
  == cases(Prop(NewSequenceOfElementType), all ss, ii (Prop(
        ss
        apr ii))))

end (SequenceOfElementType) ;

6 U: print type sequenceofbit;
  (sequenceofbit => SequenceOfBit)

 type SequenceOfBit;

needs type Bit;

declare dummy, ss, s, s1, s2, s3, s4, s5: SequenceOfBit;
declare k, k1, k2: Integer;
declare ii, i, i1, i2, j: Bit;
AFFIRM Type Specifications and Proofs

interfaces NewSequenceOfBit, s apr i, i aol s, seq(i), s1 join s2, LessFirst(s), LessLast(s): SequenceOfBit;

infix join, aol, aor;

interfaces isNewSequenceOfBit(s), s1 subset s2, FirstInduction(s), Induction(s), NormalForm(s), i in s, s1 iss s2: Boolean;

infix in, subseq, iss;

interfaces Nchanges(s), Length(s): Integer;

interfaces First(s), Last(s), LastBit(s): Bit;

axioms dummy=dummy == TRUE,
    NewSequenceOfBit = s apr i == FALSE,
    s apr i = NewSequenceOfBit == FALSE,
    s apr i = s1 aor i1 == (((s=s1) and (i=i1)));

axioms i aol NewSequenceOfBit == NewSequenceOfBit apr i,
    i aol (s apr i1) == (i aol s) apr i1;

axiom seq(i) == NewSequenceOfBit apr i;

axioms NewSequenceOfBit join s == s,
    (s apr i) join s1 == s join (i aol s1);

axiom LessFirst(s apr i) ==
    if s = NewSequenceOfBit
        then NewSequenceOfBit
        else LessFirst(s) apr i;

axiom LessLast(s apr i) == s;

axiom isNewSequenceOfBit(s) == (s = NewSequenceOfBit);

axioms s1 subset (s apr i) ==
    ( s1 = NewSequenceOfBit) or s1 subset s
    or LessLast(s1) subseq s and (Last(s1) = i),
    s subseq NewSequenceOfBit == (s = NewSequenceOfBit);

axioms i in NewSequenceOfBit == FALSE,
    i in (s apr i1) == (i in s or (i=i1));

axioms s iss NewSequenceOfBit == (s = NewSequenceOfBit),
    s1 iss (s2 apr i) ==
    ( s1 = NewSequenceOfBit) or s1 iss s2
    or (LessLast(s1) = s2) and (Last(s1) = i);

axioms Nchanges(NewSequenceOfBit) == 0,
    Nchanges(s apr i) == if LastBit(s) = i
then \( \text{Nchanges}(s) \)
else \( \text{Nchanges}(s) + 1; \)

**Axioms**

\[
\begin{align*}
\text{Length}(\text{NewSequenceOfBit}) &= 0, \\
\text{Length}(s \text{ apr } i) &= \text{Length}(s) + 1;
\end{align*}
\]

**Axiom**

\[
\text{First}(s \text{ apr } i) = \begin{cases} 
\text{if } s = \text{NewSequenceOfBit} \\
\text{then } i \\
\text{else } \text{First}(s);
\end{cases}
\]

**Axiom**

\[
\text{Last}(s \text{ aor } i) = i;
\]

**Axioms**

\[
\begin{align*}
\text{LastBit}(\text{NewSequenceOfBit}) &= \text{zero}, \\
\text{LastBit}(s \text{ apr } i) &= i;
\end{align*}
\]

**Rule Lemmas**

\[
\begin{align*}
\text{NewSequenceOfBit} = i \text{ apl } s &= \text{FALSE}, \\
i \text{ apl } s &= \text{NewSequenceOfBit} = \text{FALSE};
\end{align*}
\]

\[
\begin{align*}
\text{rules}\text{lemmas } s \text{ join } (s1 \text{ apr } i) &= \text{ (s join s1) aor } i, \\
\text{rules}\text{lemmas } s \text{ join } \text{NewSequenceOfBit} &= s, \\
\text{rules}\text{lemmas } (i \text{ apl } s1) \text{ join } s2 &= i \text{ apl } (s1 \text{ join } s2), \\
\text{rules}\text{lemmas } (s \text{ join } (i \text{ apl } s1)) \text{ join } s2 &= s \text{ join } (i \text{ apl } (s1 \text{ join } s2)) \\
\text{rules}\text{lemmas } s \text{ join } (s1 \text{ join } s2) &= (s \text{ join } s1) \text{ join } s2;
\end{align*}
\]

**Rule Lemma**

\[
\text{LessFirst}(i \text{ apl } s) = s;
\]

**Rule Lemma**

\[
\text{LessLast}(i \text{ aor } s) = \begin{cases} 
\text{if } s = \text{NewSequenceOfBit} \\
\text{then } \text{NewSequenceOfBit} \\
\text{else } i \text{ aor } \text{LessLast}(s);
\end{cases}
\]

**Rule Lemmas**

\[
\begin{align*}
\text{NewSequenceOfBit} \text{ subset } s &= \text{TRUE}, \\
s \text{ subset } s &= \text{TRUE};
\end{align*}
\]

**Rule Lemma**

\[
\begin{align*}
\text{in } (i1 \text{ aor } s) &= \text{ (i in } s \text{ or } (i=i1));
\end{align*}
\]

**Rule Lemmas**

\[
\begin{align*}
\text{NewSequenceOfBit} \text{ iss } s &= \text{TRUE}, \\
\text{iss } s &= \text{TRUE};
\end{align*}
\]

**Rule Lemma**

\[
\begin{align*}
\text{First}(i \text{ aor } s) &= i;
\end{align*}
\]

**Rule Lemma**

\[
\begin{align*}
\text{Last}(i \text{ aor } s) &= \begin{cases} 
\text{if } s = \text{NewSequenceOfBit} \\
\text{then } i \\
\text{else } \text{Last}(s);
\end{cases}
\end{align*}
\]

**Schemas**

\[
\begin{align*}
\text{FirstInduction}(s) &= \text{cases}(\text{Prop}(\text{NewSequenceOfBit}), \text{all } ss, i_i \\
&\quad (\text{IH}(ss) \\
&\quad \quad \text{imp } \text{Prop}(i_i \text{ aor } ss))),
\end{align*}
\]

\[
\text{Induction}(s)
\]
AFFIRM Type Specifications and Proofs

== cases(Prop(NewSequenceOfBit), all ss, ii
   (   IH(ss)
     imp Prop(ss apr ii)),

NormalForm(s)
== cases(Prop(NewSequenceOfBit), all ss, ii (Prop(ss apr ii)));
end {SequenceOfBit} ;

7 U: print type sequenceofpacket;
(sequenceofpacket => SequenceOfPacket)

type SequenceOfPacket;

needs types Integer, Packet, SequenceOfBit, SequenceOfElemType;

declare dummy, ss, s, s1, s2, s3, s4, s5: SequenceOfPacket;
declare k, k1, k2: Integer;
declare ii, i, ii, 12, 1: Packet;

interfaces NewSequenceOfPacket, s apr i, 1 apl s, seq(1), s1 join s2,
LessFirst(s), LessLast(s): SequenceOfPacket;

infix join, apl, apr;

interfaces isNewSequenceOfPacket(s), s1 subseq s2, FirstInduction(s),
Induction(s), NormalForm(s), i in s, s1 iss s2, Repeats(s): Boolean;

infix in, subseq, iss;

interface Seqnums(s): SequenceOfBit;

interface UniqueMsg(s): SequenceOfElemType;

interface Length(s): Integer;

interfaces First(s), Last(s): Packet;

axioms dummy=dummy == TRUE,
   NewSequenceOfPacket = s apr i == FALSE,
   s apr i = NewSequenceOfPacket == FALSE,
   s apr i = s1 apr ii == ((s=s1) and (i=ii));

axioms i apl NewSequenceOfPacket == NewSequenceOfPacket apr i,
   i apl (s apr ii) == (i apl s) apr ii;

axiom seq(1) == NewSequenceOfPacket apr 1;

axioms NewSequenceOfPacket join s == s,
   (s apr i) join s1 == s join (i apl s1);

axiom LessFirst(s apr i)
== if s = NewSequenceOfPacket 
    then NewSequenceOfPacket 
    else LessFirst(s) aor i;

axiom LessLast(s aor i) == s;

axiom isNewSequenceOfPacket(s) == (s = NewSequenceOfPacket);

axioms s1 subseq (s aor i) 
    == (   (s1 = NewSequenceOfPacket) or s1 subseq s 
        or LessLast(s1) subseq s and (Last(s1) = i)),
        s subseq NewSequenceOfPacket == (s = NewSequenceOfPacket);

axioms i in NewSequenceOfPacket == FALSE, 
    i in (s aor 11) == (i in s or (i=11));

axioms s iss NewSequenceOfPacket == (s = NewSequenceOfPacket),
    s1 iss (s2 aor i) 
    == (   (s1 = NewSequenceOfPacket) or s1 iss s2 
        or (LessLast(s1) = s2) and (Last(s1) = i));

axioms Repeats(NewSequenceOfPacket) == TRUE, 
    Repeats(s aor i) 
    == (   Repeats(s) 
        and   s "= NewSequenceOfPacket 
        and seqno(i) = seqno(Last(s)) 
        imp i = Last(s));

axioms Seqnums(NewSequenceOfPacket) == NewSequenceOfHit, 
    Seqnums(s aor i) == Seqnums(s) aor seqno(i);

axioms UniqueMsg(NewSequenceOfPacket) == NewSequenceOfElemType, 
    UniqueMsg(s aor i) 
    == if seqno(i) = LastHit(Seqnums(s)) 
        then UniqueMsg(s) 
        else UniqueMsg(s) aor mssq(i);

axioms Length(NewSequenceOfPacket) == 0, 
    Length(s aor i) == Length(s) + 1;

axiom First(s aor i) == if s = NewSequenceOfPacket 
    then i 
    else First(s);

axiom Last(s aor i) == i;

rulelemmas NewSequenceOfPacket = i aor s == FALSE, 
    i aor s = NewSequenceOfPacket == FALSE;

rulelemmas s join (s1 aor i) == (s join s1) aor i, 
    s join NewSequenceOfPacket == s, 
    (i aor s1) join s2 == i aor (s1 join s2),
(s join (i apl s1)) join s2  
  == s join (i apl (s1 join s2)),  
  s join (s1 join s2) == (s join s1) join s2;

rulelemma LessFirst(i apl s) == s;

rulelemma LessLast(i apl s)  
  == if s = NewSequenceOfPacket
     then NewSequenceOfPacket
     else i apl LessLast(s);

rulelemmas NewSequenceOfPacket subseq s == TRUE,  
  s subseq s == TRUE;

rulelemma i in (i1 apl s) == (i in s or (i=i1));

rulelemmas NewSequenceOfPacket iss s == TRUE,  
  s iss s == TRUE;

rulelemma First(i apl s) == i;

rulelemma Last(i apl s) == if s = NewSequenceOfPacket  
  then i
     else Last(s);

schemas FirstInduction(s)  
  == cases(Prop(NewSequenceOfPacket), all ss, ii  
     (  
       IH(ss)  
       imp Prop(ii apl ss))),

Induction(s)  
  == cases(Prop(NewSequenceOfPacket), all ss, ii  
     (  
       IH(ss)  
       imp Prop(ss apr ii))),

NormalForm(s)  
  == cases(Prop(NewSequenceOfPacket), all ss, ii (Prop(  
       ss  
       apr ii)));

eend (SequenceOfPacket) ;

8 U: print type abpcontext;  
(abpcontext => AbpContext)

type AbpContext:

needs types Bit, Integer, Packet, SequenceOfElemType, SequenceOfBit,  
SequenceOfPacket, ElemType;

declare dummy: AbpContext;
declare b, b1, b2: Bit;
declare i, i1, i2, j, k, k1, k2: Integer;
AFFIRM Type Specifications and Proofs

declare \( p, p_1, p_2, i: \text{Packet} \);
declare \( m, m_1, m_2: \text{SequenceOfElemType} \);
declare \( n, n_1, n_2: \text{SequenceOfBit} \);
declare \( s, s_1, s_2: \text{SequenceOfPacket} \);
declare \( e, e_1, e_2: \text{ElemType} \);

interfaces Bounded\((k_1, k_2, j)\), MsgLag\((m_1, m_2, j)\): Boolean;

axiom dummy==dummy == TRUE;

rulelemmas Bounded\((k, k, j) == TRUE,\
Bounded\((k, k+1, j) == TRUE,\
Bounded\((k+1, k, j) == FALSE,\
Bounded\((k+1, k+2, j) == Bounded\((k_1, k_2, j)\);\

rulelemmas MsgLag\((m, m, j) == TRUE,\
MsgLag\((m, m \cap n, e, 1) == TRUE,\
MsgLag\((m \cap n, e, m, 1) == FALSE;\

define Bounded\((k_1, k_2, j)\
== ((k_1 <= k_2) and (k_2 <= j+k_1)),\
MsgLag\((m_1, m_2, j)\
== (m_1 \cup m_2 and Bounded\((\text{Length}(m_1), \text{Length}(m_2), j)\));

end \{\text{AbpContext} \} ;

9 U: read \text{abocontext, theorems};
(Reading AFFIRM commands from \text{<DIVITU>ABPCONTEXT,THEOREMS,13})

New environment:
dummy: AbpContext
b, b_1, b_2: Bit
l, l_1, l_2, j, k, k_1, k_2: Integer
p, p_1, p_2, i: \text{Packet}
\( m, m_1, m_2: \text{SequenceOfElemType} \)
\( n, n_1, n_2: \text{SequenceOfBit} \)
\( s, s_1, s_2: \text{SequenceOfPacket} \)
e, e_1, e_2: \text{ElemType}

theorem SubToLag, \( s_1 \text{ subseq } s_2 \)
and Repeat(s_2)
and Bounded\((\text{Nchanges}(\text{Seqnums}(s_1))),\
\text{Nchanges}(\text{Seqnums}(s_2)), 1)\
imp \text{UniqueMsg}(s_1) \text{ iss UniqueMsg}(s_2);

theorem PktSubBound, \( s_1 \text{ subseq } s_2 \)
and Bounded\((\text{Nchanges}(\text{Seqnums}(s_1))),\
\text{Nchanges}(\text{Seqnums}(s_2)), 1)\
imp \text{Nchanges}(\text{Seqnums}(s_1)) = \text{Nchanges}(\text{Seqnums}(s_2));
AFFIRM Type Specifications and Proofs

**Theorem** BitSubBound,
\[ n_1 \subseteq n_2 \]
and \( \text{Bounded}(\text{Nchanges}(n_1), \text{Nchanges}(n_2) + 1, 1) \)
implies \( \text{Bounded}(\text{Nchanges}(n_1), \text{Nchanges}(n_2), 1) \)
and \( \text{Nchanges}(n_1) = \text{Nchanges}(n_2) \);

**Theorem** UniqueEq,
\[ \text{UniqueMsg}(s_1) \equiv \text{UniqueMsg}(s_2) \]
and \( s_1 \subseteq s_2 \)
and \( \text{Bounded}(\text{Nchanges}(\text{Seqnums}(s_1)), \text{Nchanges}(\text{Seqnums}(s_2)), 1) \)
and \( \text{LastBit}(\text{Seqnums}(s_1)) = \text{LastBit}(\text{Seqnums}(s_2)) \)
implies \( \text{UniqueMsg}(s_1) = \text{UniqueMsg}(s_2) \);

**Theorem** LastEq,
\[ n_1 \subseteq n_2 \]
and \( \text{Bounded}(\text{Nchanges}(n_1), \text{Nchanges}(n_2), 1) \)
implies \( \text{LastBit}(n_1) = \text{LastBit}(n_2) \) and \( \text{Nchanges}(n_1) = \text{Nchanges}(n_2) \);

**Theorem** SubLess, \[ n_1 \subseteq n_2 \]
implies \( \text{Nchanges}(n_1) \subseteq \text{Nchanges}(n_2) \);

**Theorem** SameLastBit,
\[ n_1 \subseteq n_2 \]
and \( b = \text{LastBit}(n_2) \)
and \( \neg ((n_1 \setminus b) \subseteq n_2) \)
implies \( \text{LastBit}(n_1) = \text{LastBit}(n_2) \);

**Theorem** SameLastBit2,
\[ s_1 \subseteq s_2 \]
and \( p = \text{Last}(s_2) \)
and \( \neg ((s_1 \setminus p) \subseteq s_2) \)
implies \( \text{LastBit}(\text{Seqnums}(s_1)) = \text{LastBit}(\text{Seqnums}(s_2)) \);

**Theorem** SubSeqnum,
\[ s_1 \subseteq s_2 \]
implies \( \text{Seqnums}(s_1) \subseteq \text{Seqnums}(s_2) \);

**Theorem** NcUnique,
\( \text{Nchanges}(\text{Seqnums}(s)) = \text{Length}(\text{UniqueMsg}(s)) \);

**Theorem** LastSeq,
\( s \equiv \text{NewSequenceOfPacket} \)
implies \( \text{seqno}(\text{Last}(s)) = \text{LastBit}(\text{Seqnums}(s)) \);

**Theorem** NcNonneg,
\( \text{Nchanges}(n) \geq 0 \);

**Theorem** BoundedBy1,
\( \text{Bounded}(k_1, k_2, 1) \)
eqv \( (k_1 = k_2) \) or \( (k_1 + 1 = k_2) \);

**Theorem** IssLenEq,
\[ m_1 \equiv m_2 \]
and \( \text{Length}(m_1) = \text{Length}(m_2) \)
eqv \( m_1 = m_2 \);

**Theorem** IssLenLe,
\[ m_1 \equiv m_2 \]
implies \( \text{Length}(m_1) \leq \text{Length}(m_2) \);

**Theorem** BiValued,
\( b \equiv b_1 \) and \( b \equiv b_2 \)
implies \( b_1 = b_2 \);

10 UI print status;
The untried theorems are BitSubBound, BiValued, BoundedBy1, IssLenEq, IssLenLe, LastEq, LastSeq, NcNonneg, NcUnique, PktSubBound, SameLastBit, SameLastBit2,
Subless, SubSeqnum, SubToLast, and Uniqueness.
No theorems are tried.
No theorems are assumed.
No theorems are awaiting lemma proof.
No theorems are proved.

11 U: try SubToLast;
SubToLast is untried.

all s1, s2
( s1 subseq s2 and Repeats(s2)
   and Bounded(nchanges(Seqnums(s1)), nchanges(Seqnums(s2)), 1)
   imp UniqueMsg(s1) iss UniqueMsg(s2))

12 U: employ Induction(s2);
Case NewSequenceOfPacket: Prop(NewSequenceOfPacket) remains to be shown.
Case apr: all ss, ii (IH(ss) imp Prop(ss apr ii)) remains to be shown.
(NewSequenceOfPacket):
all s1
( (s1 = NewSequenceOfPacket) and Bounded(nchanges(Seqnums(s1)), 0, 1)
   imp UniqueMsg(s1) = NewSequenceOfElemType)

13 U: replace:

TRUE
Going to leaf apr:

all ss', ii', s1
( IH(ss', 1 (SubToLast))
   and (s1 = NewSequenceOfPacket) or s1 subseq ss'
   or LessLast(s1) subseq ss' and (Last(s1) = ii')
   and Repeats(ss')
   and (ss' = NewSequenceOfPacket) and seqno(ii')
   = seqno(Last(ss'))
   imp ii' = Last(ss')
   imp if LastBit(Seqnums(ss')) = seqno(ii')
   then Bounded(nchanges(Seqnums(s1)), nchanges(Seqnums(ss'))), 1)
   else Bounded(nchanges(Seqnums(s1)), nchanges(Seqnums(ss')) + 1, 1)
   imp UniqueMsg(s1) iss UniqueMsg(ss')
   or UniqueMsg(s1) = UniqueMsg(ss')
   or LessLast(UniqueMsg(s1)) = UniqueMsg(ss')
   and Last(UniqueMsg(s1)) = mssq(ii'))

14 U: split;
(first:)
all ss', ii', s1
( IH(ss', 1 (SubToLast)) and (s1 = NewSequenceOfPacket) and Repeats(ss')
   and (ss' = NewSequenceOfPacket) and seqno(ii')

   imp if LastBit(Seqnums(ss')) = seqno(ii')
   then Bounded(nchanges(Seqnums(s1)), nchanges(Seqnums(ss'))), 1)
   else Bounded(nchanges(Seqnums(s1)), nchanges(Seqnums(ss')) + 1, 1)
   imp UniqueMsg(s1) iss UniqueMsg(ss')
   or UniqueMsg(s1) = UniqueMsg(ss')
   or LessLast(UniqueMsg(s1)) = UniqueMsg(ss')
   and Last(UniqueMsg(s1)) = mssq(ii'))
AFFIRM Type Specifications and Proofs

\[
\text{imp ii\text{"}} = \text{Last(ss\text{"})} = \text{seqno(}\text{Last(ss\text{"})})
\]

\[
\text{imp if LastBit(Sequums(s{s\text{"}})) = seqno(ii\text{"}) then}
\]
\[
\text{Bounded(}\text{Nchanges(Sequums(s1)), Nchanges(Sequums(s{s\text{"}})), 1)}
\]
\[
\text{imp UniqueMsg(s1) iss UniqueMsg(s{s\text{"}})}
\]
\[
\text{else}
\]
\[
\text{Bounded(}\text{Nchanges(Sequums(s1)), Nchanges(Sequums(s{s\text{"}})) + 1, 1)}
\]
\[
\text{imp UniqueMsg(s1) = NewSequenceOfEltEnumType}
\]
\[
\text{or UniqueMsg(s1) iss UniqueMsg(s{s\text{"}})}
\]
\[
\text{or LessLast(UniqueMsg(s1)) = UniqueMsg(s{s\text{"}})}
\]
\[
\text{and Last(UniqueMsg(s1)) = msg(s1\text{"})}
\]

15 U: replace;

TRUE

Going to leaf second:

all ss\text{"}, ii\text{"}, s1

( IH(ss\text{"}, 1 (SubToTag)) and (s1 \text{"} = NewSequenceOfPacket)

and s1 subseq ss\text{"} or LessLast(s1) subseq ss\text{"} and (Last(s1) = ii\text{"})

and Repeats(ss\text{"})

and (ss\text{"} = NewSequenceOfPacket) and seqno(ii\text{"})

= seqno(}\text{Last(ss\text{"})})

\[
\text{imp ii\text{"}} = \text{Last(ss\text{"})}
\]

\[
\text{imp if LastBit(Sequums(s{s\text{"}})) = seqno(ii\text{"}) then}
\]
\[
\text{Bounded(}\text{Nchanges(Sequums(s1)), Nchanges(Sequums(s{s\text{"}})), 1)}
\]
\[
\text{imp UniqueMsg(s1) iss UniqueMsg(s{s\text{"}})}
\]
\[
\text{else}
\]
\[
\text{Bounded(}\text{Nchanges(Sequums(s1)), Nchanges(Sequums(s{s\text{"}})) + 1, 1)}
\]
\[
\text{imp UniqueMsg(s1) = NewSequenceOfEltEnumType}
\]
\[
\text{or UniqueMsg(s1) iss UniqueMsg(s{s\text{"}})}
\]
\[
\text{or LessLast(UniqueMsg(s1)) = UniqueMsg(s{s\text{"}})}
\]
\[
\text{and Last(UniqueMsg(s1)) = msg(s1\text{"})}
\]

16 U: suppose LastBit(Sequums(s{s\text{"}})) = seqno(ii\text{"});

(yes:)

all ss\text{"}, ii\text{"}, s1

( LastBit(Sequums(s{s\text{"}})) = seqno(ii\text{"})

and IH(ss\text{"}, 1 (SubToTag))

and s1 \text{"} = NewSequenceOfPacket

and s1 subseq ss\text{"}

or LessLast(s1) subseq ss\text{"}

and Last(s1) = ii\text{"}

and Repeats(ss\text{"})

and (ss\text{"} = NewSequenceOfPacket

and seqno(ii\text{"}) = seqno(}Last(ss\text{"})}

\[
\text{imp ii\text{"}} = \text{Last(ss\text{"})}
\]

\[
\text{imp UniqueMsg(s1) iss UniqueMsg(s{s\text{"}})}
\]

17 U: split;

(split \Rightarrow \text{split})
AFFIRM Type Specifications and Proofs

(first:)

all ss', ii', s1
(
    LastBit(Sequences(ss')) = seqno(ii')
    and IH(ss', 1 {SubToLog})
    and s1 "= NewSequenceOfPacket
    and s1 subset ss'
    and Repeats(ss')
    and ss" "= NewSequenceOfPacket
    and seqno(ii") = seqno(Last(ss"))
    imp ii" = Last(ss")
    and
    Bounded(Nchanges(Sequences(s1)), Nchanges(Sequences(ss")), 1)
    imp UniqueMsg(s1) iss UniqueMsg(ss")
)

18 U: invoke IH;

Automatically search for instantiation? yes [confirm]

1/1: s1" = s1

Proved by chaining and narrowing using the substitution

s1" = s1

TRUE

Going to leaf second:

all ss", ii", s1
(
    LastBit(Sequences(ss")) = seqno(ii")
    and IH(ss", 1 {SubToLog})
    and s1 "= NewSequenceOfPacket
    and "(s1 subset ss")
    and LessLast(s1) subset seq ss"
    and Last(s1) = ii"
    and Repeats(ss")
    and ss" "= NewSequenceOfPacket
    and seqno(ii") = seqno(Last(ss"))
    imp ii" = Last(ss")
    and
    Bounded(Nchanges(Sequences(s1)), Nchanges(Sequences(ss")), 1)
    imp UniqueMsg(s1) iss UniqueMsg(ss")
)

19 U: employ NormalForm(s1);

Case NewSequenceOfPacket: Prop(NewSequenceOfPacket) proven.

Case apr: all ss, ii (Prop(ss apr ii)) remains to be shown.

(apr:)

all ss", ii", ss, ii
(
    LastBit(Sequences(ss)) = seqno(ii)
    and IH(ss, 1 {SubToLog})
    and "((ss' apr ii") subset seq ss"
    and ss' subset seq ss
    and ii" = ii
    and Repeats(ss)
    and ss" = NewSequenceOfPacket
    and seqno(ii") = seqno(Last(ss"))
)
AFFIRM Type Specifications and Proofs

\begin{verbatim}
imp ii = last(ss)
imp if lastbit(sequnums(ss')) = seqno(ii)
    then bounded(nchanges(sequnums(ss')), \(nchanges\)(sequnums(ss)), 1)
        imp uniquemsg(ss') iss uniquemsg(ss)
    else bounded(nchanges(sequnums(ss')) + 1, \(nchanges\)(sequnums(ss)), 1)
        imp (uniquemsg(ss') apr msg(ii')) iss uniquemsg(ss)

20 U: suppose:
   (first:)
all ss', ii', ss, ii
   (lastbit(sequnums(ss)) = seqno(ii)
     and ih(ss, 1 {subtolag})
     and "((ss' apr ii') subseq ss)
     and ss' subseq ss
     and ii' = ii
     and repeats(ss)
     and ss = newsequenceofpacket
     imp if lastbit(sequnums(ss')) = seqno(ii')
         then bounded(nchanges(sequnums(ss')), \(nchanges\)(sequnums(ss)), 1)
             imp uniquemsg(ss') iss uniquemsg(ss)
         else bounded(nchanges(sequnums(ss')) + 1, \(nchanges\)(sequnums(ss)), 1)
             imp (uniquemsg(ss') apr msg(ii')) iss uniquemsg(ss)

21 U: undo;
   suppose undone.

22 U: suppose lastbit(sequnums(ss')) = seqno(ii');
   (yes:)
all ss', ii', ss, ii
   (lastbit(sequnums(ss')) = seqno(ii')
     and lastbit(sequnums(ss)) = seqno(ii)
     and ih(ss, 1 {subtolag})
     and "((ss' apr ii') subseq ss)
     and ss' subseq ss
     and ii' = ii
     and repeats(ss)
     and ss' = newsequenceofpacket
     and seqno(ii) = seqno(last(ss))
     imp ii = last(ss)
     and bounded(nchanges(sequnums(ss')), \(nchanges\)(sequnums(ss)), 1)
     imp uniquemsg(ss') iss uniquemsg(ss)

23 U: invoke TH;
   Automatically search for instantiation? yes [confirm]

1/2: s1 = ss' apr ii'
2/2: s1 = ss'
   Proved by chaining and narrowing
\end{verbatim}
using the substitution

$s1 = ss'$

TRUE

Going to leaf no:

all $ss'$, $ii'$, $ss$, $ii$

(  LastBit(Seqnums(ss')) $ =$ seqno($ii'$)
  and LastBit(Seqnums(ss)) = seqno($ii$)
  and IH(ss, 1 (SubToTag))
  and "((ss' apr $ii'$) subseq ss)
  and $ss'$ subseq $ss$
  and $ii'$ = $ii$
  and Repeats($ss$)
  and $ss$ $=$ NewSequenceOfPacket
  and seqno($ii$) = seqno(Last($ss$))
  imp $ii$ = Last($ss$)
  and Bounded("changes(Seqnums(ss')) + 1,
               "changes(Seqnums(ss)), 1)"
  imp (UniqueMsg($ss'$) apr msgo($ii'$)) iss UniqueMsg($ss$))

24 U: replace $ii'$;

all $ss'$, $ii'$, $ss$, $ii$

(  LastBit(Seqnums(ss')) $ =$ seqno($ii$)
  and LastBit(Seqnums(ss)) = seqno($ii$)
  and IH(ss, 1 (SubToTag))
  and "((ss' apr $ii$) subseq ss)
  and $ss'$ subseq $ss$
  and $ii'$ = $ii$
  and Repeats($ss$)
  and $ss$ $=$ NewSequenceOfPacket
  and seqno($ii$) = seqno(Last($ss$))
  imp $ii$ = Last($ss$)
  and Bounded("changes(Seqnums(ss')) + 1,
               "changes(Seqnums(ss)), 1)"
  imp (UniqueMsg($ss'$) apr msgo($ii'$)) iss UniqueMsg($ss$))

25 U: apply LastSeq;

some $s ($(s = NewSequenceOfPacket) or (seqno(Last(s)) = LastBit(Seqnums(s))))$"
and ss' subseq ss
and ii'' = ii
and Repeats(ss)
and Bounded(nchanges(Seqnums(ss')) + 1,
    nchanges(Seqnums(ss)), 1)
    imp (UniqueMsg(ss') and msga(ii)) iss UniqueMsg(ss)

else
    seqno(Last(ss)) = LastBit(Seqnums(ss))
    and LastBit(Seqnums(ss')) = seqno(ii)
    and LastBit(Seqnums(ss)) = seqno(ii)
    and IH(ss, 1 (SubToLag))
    and "((ss' apr ii) subseq ss)
    and ss' subseq ss
    and ii'' = ii
    and Repeats(ss)
    and seqno(ii) = seqno(Last(ss))
    imp ii = Last(ss)
    and Bounded(nchanges(Seqnums(ss')) + 1,
        nchanges(Seqnums(ss)), 1)
    imp (UniqueMsg(ss') and msga(ii)) iss UniqueMsg(ss)

27 U: suppose;
(first:)
all ss', ii', ss, ii
    ( ss = NewSequenceOfPacket
        and LastBit(Seqnums(ss')) = seqno(ii)
        and LastBit(Seqnums(ss)) = seqno(ii)
        and IH(ss, 1 (SubToLag))
        and "((ss' apr ii) subseq ss)
        and ss' subseq ss
        and ii'' = ii
        and Repeats(ss)
        and Bounded(nchanges(Seqnums(ss')) + 1,
            nchanges(Seqnums(ss)), 1)
        imp (UniqueMsg(ss') and msga(ii)) iss UniqueMsg(ss)
    )

28 U: replace ss;

all ss', ii', ss, ii
    ( ss = NewSequenceOfPacket
        and LastBit(Seqnums(ss')) = seqno(ii)
        and zero = seqno(ii)
        and IH(NewSequenceOfPacket, 1 (SubToLag))
        and ss' = NewSequenceOfPacket
        and ii'' = ii
        imp "Bounded(nchanges(Seqnums(ss')) + 1, 0, 1))

29 U: replace ss';

TRUE
Going to leaf second:.

all ss', ii'', ss, ii
AFFIRM Type Specifications and Proofs

\[
\begin{align*}
\text{ss} &= \text{NewSequenceOfPacket} \\
\text{and segno(last(ss))} &= \text{LastBit(Sequences}(ss)) \\
\text{and LastBit(Sequences}(ss)) &= \text{segno(ii)} \\
\text{and LastBit(Sequences}(ss)) &= \text{segno(ii)} \\
\text{and IH(ss, 1 (SubToLau))} \\
\text{and } &((ss' \text{ apr ii}) \text{ subseq ss}) \\
\text{and ss' subseq ss} \\
\text{and ii' = ii} \\
\text{and Repeats(ss)} \\
\text{and segno(ii) = segno(last(ss))} \\
\text{imp ii = Last(ss)} \\
\text{and Ranged(Nchanges(Sequences}(ss')) + 1,} \\
\text{Nchanges(Sequences}(ss)), 1) \\
\text{imp (UniqueMsg(ss') apr mssq(ii)) iss UniqueMsg(ss))}
\end{align*}
\]

30 U: replace segno(ii);

all ss', ii', ss, ii

\[
\begin{align*}
\text{ss} &= \text{NewSequenceOfPacket} \\
\text{and segno(last(ss))} &= \text{LastBit(Sequences}(ss)) \\
\text{and LastBit(Sequences}(ss)) &= \text{LastBit(Sequences}(ss)) \\
\text{and LastBit(Sequences}(ss)) &= \text{segno(ii)} \\
\text{and IH(ss, 1 (SubToLau))} \\
\text{and } &((ss' \text{ apr ii}) \text{ subseq ss}) \\
\text{and ss' subseq ss} \\
\text{and ii' = ii} \\
\text{and Repeats(ss)} \\
\text{and ii = Last(ss)} \\
\text{and Ranged(Nchanges(Sequences}(ss')) + 1,} \\
\text{Nchanges(Sequences}(ss)), 1) \\
\text{imp (UniqueMsg(ss') apr mssq(ii)) iss UniqueMsg(ss))}
\end{align*}
\]

31 U: apply SameLastBit2;

some s1, s2, p

\[
\begin{align*}
s1 \text{ subseq s2 and (p = Last(s2))} \\
\text{imp (s1 apr p) subseq s2 or (LastBit(Sequences}(s1)) = LastBit(Sequences}(s2)))
\end{align*}
\]

Automatically search for instantiation? Yes [confirm]

1/4: (s2 = ss) and (s1 = ss apr ii)
2/4: (s2 = ss) and (s1 = ss')

1/1: p = ii

Proved by chaining and narrowing

using the substitution

(s2 = ss) and (s1 = ss') and (p = ii)

TRUE

32 U: nxt;
(nxt => next)

Going to leaf no:
all ss', ii', s1
  ( LastBit(Seqnums(ss')) = segno(ii')
    and IH(ss', 1 {SubToInq})
    and s1 = NewSequenceOfPacket
    and s1 subseq ss'
    or LessLast(s1) subseq ss'
      and Last(s1) = ii'
    and Repeats(ss')
    and ss' = NewSequenceOfPacket
      and segno(ii') = segno(Last(ss'))
    imp ii' = Last(ss')
    and Bounded(Nchanges(Seqnums(s1)),
      Nchanges(Seqnums(ss')) + 1, 1)
  imp UniqueMsg(s1) = NewSequenceOfElemType
  or UniqueMsg(s1) iss UniqueMsg(ss')
  or LessLast(UniqeMsg(s1)) = UniqueMsg(ss')
    and Last(UniqeMsg(s1)) = msg(s(ii'))

33 U: split;
  (first:)
all ss', ii', s1
  ( LastBit(Seqnums(ss')) = segno(ii')
    and IH(ss', 1 {SubToInq})
    and s1 = NewSequenceOfPacket
    and s1 subseq ss'
    and Repeats(ss')
    and ss' = NewSequenceOfPacket
      and segno(ii') = segno(Last(ss'))
    imp ii' = Last(ss')
    and Bounded(Nchanges(Seqnums(s1)),
      Nchanges(Seqnums(ss')) + 1, 1)
  imp UniqueMsg(s1) = NewSequenceOfElemType
  or UniqueMsg(s1) iss UniqueMsg(ss')
  or LessLast(UniqeMsg(s1)) = UniqueMsg(ss')
    and Last(UniqeMsg(s1)) = msg(s(ii'))

34 U: apply PktSubBound;

some s1', s2
  ( s1' subseq s2
    and Bounded(Nchanges(Seqnums(s1')), Nchanges(Seqnums(s2)) + 1, 1)
  imp Bounded(Nchanges(Seqnums(s1')), Nchanges(Seqnums(s2)), 1)
    and Nchanges(Seqnums(s1')) = Nchanges(Seqnums(s2))
  Automatically search for instantiation? no [confirm]

35 U: put s1'=s1, s2=ss';

all ss', ii', s1
  ( s1 subseq ss'
    and Bounded(Nchanges(Seqnums(s1)),

\textbf{AFFIRM Type Specifications and Proofs}

\begin{verbatim}
Mchanges(Seqnums(s)) + 1, 1
and Bounded(Mchanges(Seqnums(s)), Mchanges(Seqnums(s'))), 1)
and Mchanges(Seqnums(s)) = Mchanges(Seqnums(s'))
and LastBit(Seqnums(s)) = segno(ii)
and IH(ss', 1 \{SubToLag\})
and s1 = NewSequenceOfPacket
and Repeats(ss')
and ss' = NewSequenceOfPacket
and segno(ii') = segno(Last(ss'))
imp ii' = Last(ss')
imp UniqueMsg(s1) = NewSequenceOfElemType
or UniqueMsg(s1) = segno(ss')
or LessLast(UniqueMsg(s1)) = UniqueMsg(ss')
and Last(UniqueMsg(s1)) = msg(ii')

36 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/1: s1'' = s1
Proved by chaining and narrowing
using the substitution

s1'' = s1

TRUE
Going to leaf second:

all ss', ii', s1
( LastBit(Seqnums(ss')) = segno(ii')
and IH(ss', 1 \{SubToLag\})
and s1 = NewSequenceOfPacket
and "(s1 subseq ss')
and LessLast(s1) subseq ss'
and Last(s1) = ii'
and Repeats(ss')
and ss' = NewSequenceOfPacket
and segno(ii') = segno(Last(ss'))
imp ii' = Last(ss')
and Bounded(Mchanges(Seqnums(s1)),
Mchanges(Seqnums(s'))), 1, 1
imp UniqueMsg(s1) = NewSequenceOfElemType
or UniqueMsg(s1) = segno(ss')
or LessLast(UniqueMsg(s1)) = UniqueMsg(ss')
and Last(UniqueMsg(s1)) = msg(ii')

37 U: employ NormalForm(s1);
Case NewSequenceOfPacket: Prop("NewSequenceOfPacket") proven.
Case apr: all ss, ii (Prop(ss apr ii)) remains to be shown.
(apr:)
all ss', ii', ss, ii
( LastBit(Seqnums(ss)) = segno(ii)
and IH(ss, 1 \{SubToLag\})
\end{verbatim}
and "((ss' apr ii') subseq ss)
and ss' subseq ss
and ii' = ii
and Repeats(ss)
and ss " = NewSequenceOfPacket
and seqno(ii') = seqno(Last(ss))

imp ii = Last(ss)

imp if LastBit(Seqnums(ss')) = seqno(ii')

then Bounded(Mchanges(Seqnums(ss'))),
Mchanges(Seqnums(ss')) + 1, 1)

imp UniqueMsg(ss') = NewSequenceOfElemType
or UniqueMsg(ss') iss UniqueMsg(ss)

or LessLast(UniqueMsg(ss')) = UniqueMsg(ss)
and Last(UniqueMsg(ss')) = msgg(ii)

else Bounded(Mchanges(Seqnums(ss'))), Mchanges(Seqnums(ss')) + 1, 1)

imp (UniqueMsg(ss') apr msgg(ii')) iss UniqueMsg(ss)
or UniqueMsg(ss') = UniqueMsg(ss)
and msgg(ii') = msgg(ii))

38 U: suppose LastBit(Seqnums(ss')) = seqno(ii');
(yes!)
all ss', ii', ss, ii

( LastBit(Seqnums(ss')) = seqno(ii')
and LastBit(Seqnums(ss)) " = seqno(ii)
and IH(ss, 1 (SubioLog))
and "((ss' apr ii') subseq ss)
and ss' subseq ss
and ii' = ii
and Repeats(ss)
and ss " = NewSequenceOfPacket
and seqno(ii) = seqno(Last(ss))

imp ii = Last(ss)
and Bounded(Mchanges(Seqnums(ss'))),
Mchanges(Seqnums(ss')) + 1, 1)

imp UniqueMsg(ss') = NewSequenceOfElemType
or UniqueMsg(ss') iss UniqueMsg(ss)

or LessLast(UniqueMsg(ss')) = UniqueMsg(ss)
and Last(UniqueMsg(ss')) = msgg(ii))

39 U: apply PktSubaound;

some s1, s2

( s1 subseq s2
and Bounded(Mchanges(Seqnums(s1)), Mchanges(Seqnums(s2)) + 1, 1)

imp Bounded(Mchanges(Seqnums(s1)), Mchanges(Seqnums(s2)), 1)
and Mchanges(Seqnums(s1)) = Mchanges(Seqnums(s2))

Automatically search for instantiation? no [confirm]

40 U: put s1=ss', s2=ss;

all ss', ii', ss, ii
AFFIRM Type Specifications and Proofs

\[
\begin{align*}
\text{ss} \quad \text{subseq ss} \\
\text{and Bounded(}'n\text{changes(Sequences(ss)))}, \\
\text{'n\text{changes(Sequences(ss))} + 1, 1) \\
\text{and Bounded(}'n\text{changes(Sequences(ss))}, 'n\text{changes(Sequences(ss))}, 1) \\
\text{and } 'n\text{changes(Sequences(ss))}' = 'n\text{changes(Sequences(ss))} \\
\text{and LastBit(Sequences(ss))} = \text{seqno(ii')} \\
\text{and LastBit(Sequences(ss))} = \text{seqno(ii) } \\
\text{and IH(ss, 1 {SubToLog})} \\
\text{and } '((ss) \text{ apr ii') subseq ss} \\
\text{and ii'' = ii} \\
\text{and Repeats(ss)} \\
\text{and } ss = \text{newSequenceOfPacket} \\
\text{and seqno(ii) = seqno(Last(ss))} \\
\text{imp ii = Last(ss)} \\
\text{imp } UniqueMsg(ss') = \text{newSequenceOfType} \\
\text{or UniqueMsg(ss') iss UniqueMsg(ss)} \\
\text{or } LessLast(UniqueMsg(ss')) = UniqueMsg(ss) \\
\text{and Last(UniqueMsg(ss'))} = mssq(ii))
\end{align*}
\]

41 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/2:  s1 = ss'
Proved by chaining and narrowing
using the substitution

s1 = ss'

TRUE
Going to leaf no.

all ss'', ii'', ss, ii
(  \text{LastBit(Sequences(ss'))} = \text{seqno(ii'')} \\
\text{and LastBit(Sequences(ss))} = \text{seqno(ii)} \\
\text{and IH(ss, 1 {SubToLog})} \\
\text{and } '((ss) \text{ apr ii'')} subseq ss \\
\text{and ss'' subseq ss} \\
\text{and ii'' = ii} \\
\text{and Repeats(ss)} \\
\text{and } ss = \text{newSequenceOfPacket} \\
\text{and seqno(ii) = seqno(Last(ss))} \\
\text{imp ii = Last(ss)} \\
\text{imp } (UniqueMsg(ss')) apr mssq(ii'')) iss UniqueMsg(ss) \\
\text{or } UniqueMsg(ss') = UniqueMsg(ss) \\
\text{and mssq(ii'') = mssq(ii))}

42 U: replace;
(replace => replace)

all ss'', ii'', ss, ii
(  \text{LastBit(Sequences(ss'))} = \text{seqno(ii)}


and LastBit(Seqnums(ss)) = seqno(ii)
and IH(ss, 1 (SubToLag))
and "(((ss') app ii) subseq ss)
and ss' subseq ss
and ii' = ii
and Repeats(ss)
and ss' = NewSequenceOfPacket
and seqno(ii) = seqno(Last(ss))
    imp ii = Last(ss)
and Bounded(Nchanges(Seqnums(ss')), Nchanges(Seqnums(ss)), 1)
    imp (UniqueMsg(ss') app msg(ii)) iss UniqueMsg(ss)
or UniqueMsg(ss') = UniqueMsg(ss))

43 U: apply oivalued;
(bivalued => B1Valued)

some b, b1, b2 ((b=b1) or (b=b2) or (b1=b2))
Automatically search for instantiation? no [confirm]

44 U: put b=seqno(ii), b1=LastBit(Seqnums(ss')), b2=LastBit(Seqnums(ss));

all ss', ii', ss, ii
    seqno(ii) = LastBit(Seqnums(ss'))
    and seqno(ii) = LastBit(Seqnums(ss'))
    and LastBit(Seqnums(ss')) = LastBit(Seqnums(ss'))
and IH(ss, 1 (SubToLag))
and "(((ss') app ii') subseq ss')
and ss' subseq ss
and ii' = ii
and Repeats(ss)
and ss' = NewSequenceOfPacket
and seqno(ii) = seqno(Last(ss))
    imp ii = Last(ss)
and Bounded(Nchanges(Seqnums(ss')), Nchanges(Seqnums(ss')), 1)
    imp (UniqueMsg(ss') app msg(ii)) iss UniqueMsg(ss)
or UniqueMsg(ss') = UniqueMsg(ss'))

45 U: apply UniqueEq;

some s1, s2
do UniqueMsg(s1) iss UniqueMsg(s2) and s1 subseq s2
and Bounded(Nchanges(Seqnums(s1)), Nchanges(Seqnums(s2)), 1)
and LastBit(Seqnums(s1)) = LastBit(Seqnums(s2))
    imp UniqueMsg(s1) = UniqueMsg(s2))
Automatically search for instantiation? no [confirm]

46 U: put s1=ss', s2=ss;

all ss', ii', ss, ii
    "(UniqueMsg(ss') iss UniqueMsg(ss))
and seqno(ii) = LastBit(Seqnums(ss'))
and seqno(ii) = LastBit(Seqnums(ss'))

AFFIRM Type Specifications and Proofs

and LastBit(Seqnums (ss\')) = LastBit(Seqnums (ss))
and TH (ss, 1 (SubToLog))
and ''((ss \" ap r ii) subseq ss)
and ss" subseq ss
and ii = ii
and Repeats (ss)
and ss " = NewSequenceOfPacket
and segno (ii) = segno (Last (ss))
imp ii = Last (ss)
and Bounded (Nachanges (Seqnums (ss\'))), Nachanges (Seqnums (ss)), 1)
imp ("UniqueMsg (ss) ap r msg (ii) iss UniqueMsg (ss)
or UniqueMsg (ss\') = UniqueMsg (ss))

47 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/2: s1 = ss\'
Proved by chaining and narrowing
using the substitution

s1 = ss\'

TRUE
SubToLog is awaiting the proof of lemmas LastSeq, SameLastBit2, PktSubBound,
BiValued, and UniqueEq.
Going to lemma PktSubBound.
PktSubBound is untried.

c1 s1, s2
( s1 subseq s2
and Bounded (Nachanges (Seqnums (s1)), Nachanges (Seqnums (s2)) + 1, 1)
imp Bounded (Nachanges (Seqnums (s1)), Nachanges (Seqnums (s2)), 1)
and Nachanges (Seqnums (s1)) = Nachanges (Seqnums (s2)))

48 U: note The hump has been passed but there still are
a few nontrivial lemmas left.

49 U: freeze abp.frz;
writing file ... <DIVITO>ABP.FRZ.1

50 U: quit;
Automatically summarize the proof attempts? no [confirm]
Type CONTINUE to return to AFFIRM.

Transcript file <DIVITO>AFFLOG.12-FEB-81.1
is open in the AFFIRM system <AFFIRM>AFFIRM.EXE.120
(<DIVITO>ABP.FRZ.1 . <AFFIRM>AFFIRM.EXE.120)

50 U: print status;
The untried theorems are hitSubBound, BiValued, BoundedBy1, IssLenEq, IssLenLe
, LastEq, LastSeq, McNonneg, McUnique, PktSubBound, SameLastBit, SameLastBit2,
SubLess, SubSeqnum, and UniqueEq.
No theorems are tried.
No theorems are assumed.
The awaiting lemma proof theorem is SubSeqnum.
No theorems are proved.

51 U: print results:

all s1, s2
   ( s1 subset s2
     and Bounded(\text{nchanges}(\text{Seqnums}(s1)), \text{nchanges}(\text{Seqnums}(s2)) + 1, 1)
     imp Bounded(\text{nchanges}(\text{Seqnums}(s1)), \text{nchanges}(\text{Seqnums}(s2)), 1)
     and \text{nchanges}(\text{Seqnums}(s1)) = \text{nchanges}(\text{Seqnums}(s2)))

52 U: apply SubSeqnum;

some s1', s2' (s1' subset s2' imp \text{Seqnums}(s1') subset \text{Seqnums}(s2'))
Automatically search for instantiation? no [confirm]

53 U: put s1'=s1, s2'=s2;

all s1, s2
   ( s1 subset s2 and \text{Seqnums}(s1) subset \text{Seqnums}(s2)
     and Bounded(\text{nchanges}(\text{Seqnums}(s1)),
                 \text{nchanges}(\text{Seqnums}(s2)) + 1, 1)
     imp Bounded(\text{nchanges}(\text{Seqnums}(s1)), \text{nchanges}(\text{Seqnums}(s2)), 1)
     and \text{nchanges}(\text{Seqnums}(s1)) = \text{nchanges}(\text{Seqnums}(s2)))

54 U: invoke Bounded 1all1;

all s1, s2
   ( s1 subset s2 and \text{Seqnums}(s1) subset \text{Seqnums}(s2)
     and \text{nchanges}(\text{Seqnums}(s1)) <= \text{nchanges}(\text{Seqnums}(s2)) + 1
     and \text{nchanges}(\text{Seqnums}(s2)) <= \text{nchanges}(\text{Seqnums}(s1))
     imp \text{nchanges}(\text{Seqnums}(s1)) <= \text{nchanges}(\text{Seqnums}(s2)))

55 U: apply Subless;

some n1, n2 (n1 subset n2 imp \text{nchanges}(n1) <= \text{nchanges}(n2))
Automatically search for instantiation? yes [confirm]

1/2: (n2 = \text{Seqnums}(s2)) and (n1 = \text{Seqnums}(s1))
Proved by chaining and narrowing
using the substitution

(n2 = \text{Seqnums}(s2)) and (n1 = \text{Seqnums}(s1))

TRUE
PktSubseqnum is awaiting the proof of lemmas SubSeqnum and Subless.

56 U: next;
Going to lemma Subless.
Subless is untried.
all n1, n2 (n1 subseq n2 imp \( \text{Nchanges}(n1) \leq \text{Nchanges}(n2) \))

57 U: employ Induction(n2);
Case NewSequenceOfBit: Prop(NewSequenceOfBit) remains to be shown.
Case apr: all ss, ii (IH(ss) imp Prop(ss apr ii)) remains to be shown.
(NewSequenceOfBit)
all n1 (n1 = NewSequenceOfBit imp \( \text{Nchanges}(n1) \leq 0 \))

58 U: replace;
TRUE
Going to leaf apr:;

all ss$, ii$, n1
( IH(ss$, 6 \{SubLess\})
  and (n1 = NewSequenceOfBit or n1 subseq ss$
  or LessLast(n1) subseq ss$ and (Last(n1) = ii$)
  imp if LastBit(ss$) = ii$
    then \( \text{Nchanges}(n1) \leq \text{Nchanges}(ss$)
    else \( \text{Nchanges}(n1) \leq \text{Nchanges}(ss$) + 1 \))

59 U: split;
(first:)
all ss$, ii$, n1
( IH(ss$, 6 \{SubLess\}) and (n1 = NewSequenceOfBit)
  imp if LastBit(ss$) = ii$
    then \( \text{Nchanges}(n1) \leq \text{Nchanges}(ss$)
    else \( \text{Nchanges}(n1) \leq \text{Nchanges}(ss$) + 1 \))

60 U: apply NNonneg;

some n (0 \leq \text{Nchanges}(n))
Automatically search for instantiation? no [confirm]

61 U: put n=ss$;

all ss$, ii$, n1
( 0 \leq \text{Nchanges}(ss$) and IH(ss$, 6 \{SubLess\})
  and n1 = NewSequenceOfBit
  imp if LastBit(ss$) = ii$
    then \( \text{Nchanges}(n1) \leq \text{Nchanges}(ss$)
    else \( \text{Nchanges}(n1) \leq \text{Nchanges}(ss$) + 1 \))

62 U: replace;
TRUE
Going to leaf second:;

all ss$, ii$, n1
( IH(ss$, 6 \{SubLess\}) and (n1 = NewSequenceOfBit)
  and n1 subseq ss$ or LessLast(n1) subseq ss$ and (Last(n1) = ii$)
imp if \text{LastBit}(ss) = \text{iis}
then \text{Nchanges}(n1) \leq \text{Nchanges}(ss)
else \text{Nchanges}(n1) \leq \text{Nchanges}(ss) + 1

63 U: split;
(\text{first}:)
all ss, iis, n1
(\text{IH}(ss, s \{\text{SubLess}\}) \text{and} (n1 = \text{NewSequenceOfBit}) \text{and} n1 \text{ subseq } ss
\text{imp if } \text{LastBit}(ss) = \text{iis}
then \text{Nchanges}(n1) \leq \text{Nchanges}(ss)
else \text{Nchanges}(n1) \leq \text{Nchanges}(ss) + 1)

64 U: invoke IH:
Automatically search for instantiation? yes [confirm]

1/1: n1' = n1
Unsuccessful.

all ss, iis, n1 (some n1')
(\text{n1' subseq ss} \text{imp } \text{Nchanges}(n1') \leq \text{Nchanges}(ss)
\text{and } n1' = \text{NewSequenceOfBit}
\text{and } n1 \text{ subseq ss}
\text{imp if } \text{LastBit}(ss) = \text{iis}
then \text{Nchanges}(n1) \leq \text{Nchanges}(ss)
else \text{Nchanges}(n1) \leq \text{Nchanges}(ss) + 1)

65 U: put n1'=n1;
TRUE
Going to leaf second:

all ss, iis, n1
(\text{IH}(ss, s \{\text{SubLess}\}) \text{and } (n1 = \text{NewSequenceOfBit})
\text{and } (n1 \text{ subseq } ss)
\text{and } \text{LastLast}(n1) \text{ subseq } ss
\text{and } \text{Last}(n1) = \text{iis}
\text{imp if } \text{LastBit}(ss) = \text{iis}
then \text{Nchanges}(n1) \leq \text{Nchanges}(ss)
else \text{Nchanges}(n1) \leq \text{Nchanges}(ss) + 1)

66 U: employ NormalForm(n1);
Case \text{NewSequenceOfBit}: \text{Prop}(\text{NewSequenceOfBit}) \text{proven.}
Case \text{apr}: all ss, ii \text{ (Prop(ss apr ii)) remains to be shown.}
(apr):
all ss, iis, ss', ii'
(\text{IH}(ss', s \{\text{SubLess}\}) \text{and } ((ss \text{ apr } ii) \text{ subseq } ss')
\text{and } ss \text{ subseq } ss'
\text{and } ii'=iis'
\text{imp if } \text{LastBit}(ss') = ii'
then if \text{LastBit}(ss) = iis
then \text{Nchanges}(ss) \leq \text{Nchanges}(ss')
else \text{Nchanges}(ss) < \text{Nchanges}(ss')
else if LastBit(sss) = iis
  then \( \text{Nchanges}(sss) \leq \text{Nchanges}(sss^*) + 1 \)
  else \( \text{Nchanges}(sss) \leq \text{Nchanges}(sss^*) \)

67 U: invoke IH:
Automatically search for instantiation? no [confirm]

all sss, iis, sss", iis" (some n1
  n1 subseq sss" imp \( \text{Nchanges}(n1) \leq \text{Nchanges}(sss^*) \)
  and "((sss and iis) subseq sss")
  and sss subseq sss"
  and iis=iis"
  imp if LastBit(sss") = iis"
    then if LastBit(sss) = iis
      then \( \text{Nchanges}(sss) \leq \text{Nchanges}(sss^*) \)
      else \( \text{Nchanges}(sss) < \text{Nchanges}(sss^*) \)
    else if LastBit(sss) = iis
      then \( \text{Nchanges}(sss) \leq \text{Nchanges}(sss^*) + 1 \)
      else \( \text{Nchanges}(sss) \leq \text{Nchanges}(sss^*) \))

68 U: put n1=sss;

all sss, iis, sss", iis" (sssubseq sss" and \( \text{Nchanges}(sss) \leq \text{Nchanges}(sss^*) \))
  and "((sss and iis) subseq sss")
  and iis = iis"
  and LastBit(sss") = iis"
  imp (LastBit(sss) = iis") or \( \text{Nchanges}(sss) < \text{Nchanges}(sss^*) \))

69 U: replace:

all sss, iis, sss", iis" (sssubseq sss" and \( \text{Nchanges}(sss) \leq \text{Nchanges}(sss^*) \))
  and "((sss and iis") subseq sss")
  and iis = iis"
  and LastBit(sss") = iis"
  imp (LastBit(sss) = iis") or \( \text{Nchanges}(sss) < \text{Nchanges}(sss^*) \))

70 U: apply SameLastBit:

some n1, n2, b
  n1 subseq n2 and (b = LastBit(n2))
  imp (n1 apr b) subseq n2 or (LastBit(n1) = LastBit(n2))
Automatically search for instantiation? yes [confirm]

1/4: (n2 = sss") and (n1 = sss)
  1/2: b = LastBit(sss)
  2/2: b = iis"
2/4: (n2 = sss") and (n1 = sss apr iis")
3/4: b = LastBit(n1)
4/4: (n2 = sss") and (b = iis") and (n1 = sss)
Unsuccessful.
71 UI: choose 4;

4/4: \((n_2 = sss') \land (b = iis') \land (n_1 = sss)\)
all sss, iis, sss', iis'

( sss' subseq sss and (iis' = LastBit(sss'))
and "((sss apr iis') subseq sss')
and LastBit(sss) = LastBit(sss')
and `\(\text{changes}(sss) <= \text{changes}(sss')\)`
and iis = iis'
imp (LastBit(sss) = iis') or (`\(\text{changes}(sss) < \text{changes}(sss')\)`)

72 UI: replace;
(replace => replace)

SubLess is awaiting the proof of lemmas NewNonneg and SameLastBit.

TRUE

Going to lemma SameLastBit.

SameLastBit is untried.

all n1, n2, b

( n1 subseq n2 and (b = LastBit(n2))
imp (n1 apr b) subseq n2 or (LastBit(n1) = LastBit(n2))

73 UI: employ NormalForm(n2);

Case NewSequenceOfBit: Prop(NewSequenceOfBit) remains to be shown.
Case apr: all ss, ii (Prop(ss apr ii)) remains to be shown.

(NewSequenceOfBit:)
all n1, b ( \((n_1 = \text{NewSequenceOfBit}) \land (b = \text{zero})\)
imp LastBit(n1) = zero)

74 UI: replace;

TRUE

Going to leaf apr:.

all sss, iis, n1, b

(if n1 = NewSequenceOfBit
then \(b=iis\)
imp (n1 apr b) subseq sss or n1 subseq sss
or LastBit(n1) = iis
else ((n1 subseq sss) and LessLast(n1) subseq sss
and Last(n1) = iis
and b=iis
imp (n1 apr b) subseq sss or (LastBit(n1) = iis))

75 UI: employ NormalForm(n1);

Case NewSequenceOfBit: Prop(NewSequenceOfBit) proven,
Case apr: all ss, ii (Prop(ss apr ii)) proven.

SameLastBit proven.

TRUE
Going to unproven ancestor \texttt{\textit{NcNonneg}}.
\texttt{\textit{NcNonneg}} is untried.

\texttt{\textit{all n \{0 <= nChanges(n)\}}}

\texttt{\textbf{76 U: employ Induction(n);}}
\texttt{\textbf{Case NewSequenceOfBit: Prop(NewSequenceOfBit) proven.}}
\texttt{\textbf{Case apr: all ss, i1 ( IH(ss) imp Prop(ss apr i1) ) remains to be shown.}}
\texttt{\textbf{(apr:)}}
\texttt{\textbf{all sss, i1s}}
\begin{verbatim}
    ( IH(sss, 12 (\texttt{\textit{NcNonneg}}))
      imp if LastBit(sss) = i1s
         then 0 <= nChanges(sss)
         else 0 <= nChanges(sss) + 1)
\end{verbatim}

\texttt{\textbf{77 U: invoke IH;}}
\texttt{\textbf{\textit{NcNonneg}} proved.}
\texttt{\textbf{SubLess proved.}}

\texttt{\textbf{TRUE}}
\texttt{\textbf{Going to unproven ancestor SubSeqnum.}}
\texttt{\textbf{SubSeqnum is untried.}}

\texttt{\textbf{all s1, s2 ( s1 subseq s2 imp Seqnums(s1) subseq Seqnums(s2) )}}

\texttt{\textbf{78 U: employ Induction(s2);}}
\texttt{\textbf{Case NewSequenceOfPacket: Prop(NewSequenceOfPacket) remains to be shown.}}
\texttt{\textbf{Case apr: all ss, i1 ( IH(ss) imp Prop(ss apr i1) ) remains to be shown.}}
\texttt{\textbf{(NewSequenceOfPacket:)}}
\texttt{\textbf{all s1 ( s1 = NewSequenceOfPacket imp Seqnums(s1) = NewSequenceOfBit )}}

\texttt{\textbf{79 U: replace;}}
\texttt{\textbf{(replace => replace)}}

\texttt{\textbf{TRUE}}
\texttt{\textbf{Going to leaf apr:.}}

\texttt{\textbf{all ss', ii', s1}}
\begin{verbatim}
    ( IH(ss', 9 (SubSeqnum))
      and ( s1 = NewSequenceOfPacket ) or s1 subseq ss'
      or LessLast(s1) subseq ss' and (Last(s1) = ii')
      imp ( Seqnums(s1) = NewSequenceOfBit ) or Seqnums(s1) subseq Seqnums(ss')
      or LessLast(Seqnums(s1)) subseq Seqnums(ss') and Last(Seqnums(s1)) = segno(ii')
\end{verbatim}

\texttt{\texttt{\textbf{80 U: split;}}}
\texttt{\textbf{(first:)}}
\texttt{\textbf{all ss', ii', s1}}
\begin{verbatim}
    ( IH(ss', 9 (SubSeqnum)) and ( s1 = NewSequenceOfPacket )
      imp ( Seqnums(s1) = NewSequenceOfBit ) or Seqnums(s1) subseq Seqnums(ss')
      or LessLast(Seqnums(s1)) subseq Seqnums(ss') and Last(Seqnums(s1)) = segno(ii')
\end{verbatim}
81 U: replace:

TRUE
Going to leaf second:

all ss", ii", s1
  ( IH(ss", 9 (SubSeqnum)) and (s1 = NewSequenceOfPacket) 
    and s1 subseq ss" or LessLast(s1) subseq ss" and (last(s1) = ii") 
    imp (Seqnums(s1) = NewSequenceOfPacket) or Seqnums(s1) subseq Seqnums(ss") 
    or LessLast(Seqnums(s1)) subseq Seqnums(ss") and last(Seqnums(s1)) = seqno(ii") 

82 U: split;
  (first):
all ss", ii", s1
  ( IH(ss", 9 (SubSeqnum)) and (s1 = NewSequenceOfPacket) and subseq ss" 
    imp (Seqnums(s1) = NewSequenceOfPacket) or Seqnums(s1) subseq Seqnums(ss") 
    or LessLast(Seqnums(s1)) subseq Seqnums(ss") and last(Seqnums(s1)) = seqno(ii") 

83 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/1: s1" = s1 
Proved by chaining and narrowing
using the substitution

s1" = s1 
TRUE
Going to leaf second:

all ss", ii", s1
  ( IH(ss", 9 (SubSeqnum)) and (s1 = NewSequenceOfPacket) 
    and (s1 subseq ss") 
    and LessLast(s1) subseq ss" 
    and last(s1) = ii" 
    imp (Seqnums(s1) = NewSequenceOfPacket) or Seqnums(s1) subseq Seqnums(ss") 
    or LessLast(Seqnums(s1)) subseq Seqnums(ss") and last(Seqnums(s1)) = seqno(ii") 

84 U: employ NormalForm(s1); 
Case NewSequenceOfPacket: Prop(NewSequenceOfPacket) proven. 
Case apr: all ss, ii (Prop(ss apr ii)) remains to be shown, 
  (apr:)
all ss", ii", ss, ii
  ( IH(ss, 9 (SubSeqnum)) and ((ss" apr ii") subseq ss) 
    and ss" subseq ss 
    and ii" = ii 
    imp (Seqnums(ss") apr seqno(ii") subseq Seqnums(ss)
or Seqnums(ss'\) subseq Seqnums(ss) and (seqno(ii') = seqno(ii))

85 U: replace ;

all ss', ii', ss, ii
  ( IH(ss, 9 {SubSeqnum})
    and '((ss' apr ii) subseq ss)
    and ss' subseq ss
    and ii' = ii
    imp (Seqnums(ss') apr seqno(ii)) subseq Seqnums(ss)
    or Seqnums(ss') subseq Seqnums(ss))

86 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/2: s1 = ss' apr ii
2/2: s1 = ss'
Proved by chaining and narrowing
using the substitution
s1 = ss'

TRUE
SubSeqnum proved.
PktSubBound proved.
There's more than one unproven ancestor. You may pick one of UniqueEq,
SameLastBit2, LastSeq, or HValued.

87 U: note
The toughest lemmas remain in the path which
begins with UniqueEq, ;

88 U: try uniqueeq;
(UniqueEq =⇒ UniqueEq)
UniqueEq is untried.

all s1, s2
  ( UniqueMsg(s1) iss UniqueMsg(s2) and s1 subseq s2
    and Bounded(Nchanges(Seqnums(s1)), Nchanges(Seqnums(s2)), 1)
    and LastBit(Seqnums(s1)) = LastBit(Seqnums(s2))
    imp UniqueMsg(s1) = UniqueMsg(s2))

89 U: apply BoundedBy1;

some k1, k2
  ( Bounded(k1, k2, 1)
    eqv (k1=k2) or (k1+1 = k2))
Automatically search for instantiation? yes [confirm]

1/1: (k2 = Nchanges(Seqnums(s2))) and (k1 = Nchanges(Seqnums(s1)))
Unsuccessful.
90 U: choose 1;

1/1: \((k_2 = \text{nchanges}(\text{Seqnums}(s_2))) \land (k_1 = \text{nchanges}(\text{Seqnums}(s_1)))\)

all \(s_1, s_2\)

\(\{\)

\(\text{nchanges}(\text{Seqnums}(s_1)) = \text{nchanges}(\text{Seqnums}(s_2))\)

or \(\text{nchanges}(\text{Seqnums}(s_1)) < \text{nchanges}(\text{Seqnums}(s_2))\)

and \(\text{nchanges}(\text{Seqnums}(s_1)) + 1 = \text{nchanges}(\text{Seqnums}(s_2))\)

and \(\text{Bounded}(\text{nchanges}(\text{Seqnums}(s_1)), \text{nchanges}(\text{Seqnums}(s_2)), 1)\)

and \(\text{UniqueMsg}(s_1) \text{ iss UniqueMsg}(s_2)\)

and \(s_1 \text{ subseq } s_2\)

and \(\text{LastBit}(\text{Seqnums}(s_1)) = \text{LastBit}(\text{Seqnums}(s_2))\)

imp \(\text{UniqueMsg}(s_1) = \text{UniqueMsg}(s_2)\)

\(\}\)

91 U: split;

(first:)

all \(s_1, s_2\)

\(\{\)

\(\text{nchanges}(\text{Seqnums}(s_1)) = \text{nchanges}(\text{Seqnums}(s_2))\)

and \(\text{Bounded}(\text{nchanges}(\text{Seqnums}(s_1)), \text{nchanges}(\text{Seqnums}(s_2)), 1)\)

and \(\text{UniqueMsg}(s_1) \text{ iss UniqueMsg}(s_2)\)

and \(s_1 \text{ subseq } s_2\)

and \(\text{LastBit}(\text{Seqnums}(s_1)) = \text{LastBit}(\text{Seqnums}(s_2))\)

imp \(\text{UniqueMsg}(s_1) = \text{UniqueMsg}(s_2)\)

\(\}\)

92 U: apply ncunique;

(ncunique => ncUnique)

some \(s\) \((\text{nchanges}(\text{Seqnums}(s)) = \text{Length}(\text{UniqueMsg}(s)))\)

Automatically search for instantiation? no [confirm]

93 U: apply ncunique;

(ncunique => ncUnique)

some \(s'\) \((\text{nchanges}(\text{Seqnums}(s')) = \text{Length}(\text{UniqueMsg}(s'))\))

Automatically search for instantiation? no [confirm]

94 U: put \(s=s_1, s'=s_2\);

all \(s_1, s_2\)

\(\{\)

\(\text{nchanges}(\text{Seqnums}(s_2)) = \text{Length}(\text{UniqueMsg}(s_2))\)

and \(\text{nchanges}(\text{Seqnums}(s_1)) = \text{Length}(\text{UniqueMsg}(s_1))\)

and \(\text{nchanges}(\text{Seqnums}(s_1)) = \text{nchanges}(\text{Seqnums}(s_2))\)

and \(\text{Bounded}(\text{nchanges}(\text{Seqnums}(s_1)), \text{nchanges}(\text{Seqnums}(s_2)), 1)\)

and \(\text{UniqueMsg}(s_1) \text{ iss UniqueMsg}(s_2)\)

and \(s_1 \text{ subseq } s_2\)

and \(\text{LastBit}(\text{Seqnums}(s_1)) = \text{LastBit}(\text{Seqnums}(s_2))\)

imp \(\text{UniqueMsg}(s_1) = \text{UniqueMsg}(s_2)\)

\(\}\)

95 U: replace;

all \(s_1, s_2\)

\(\{\)

\(\text{nchanges}(\text{Seqnums}(s_2)) = \text{Length}(\text{UniqueMsg}(s_2))\)

and \(\text{nchanges}(\text{Seqnums}(s_1)) = \text{Length}(\text{UniqueMsg}(s_2))\)
and Length(UniqueMsg(s1)) = Length(UniqueMsg(s2))
and UniqueMsg(s1) iss UniqueMsg(s2)
and s1 subseq s2
and LastBit(Seqnums(s1)) = LastBit(Seqnums(s2))
imp UniqueMsg(s1) = UniqueMsg(s2))

96 U: apply IssLenEq;

some m1, m2 (m1 iss m2 and (Length(m1) = Length(m2)) eqv m1=m2)
Automatically search for instantiation? yes [confirm]

1/1: (m2 = UniqueMsg(s2)) and (m1 = UniqueMsg(s1))
Proved by chaining and narrowing
using the substitution

(m2 = UniqueMsg(s2)) and (m1 = UniqueMsg(s1))

TRUE

97 U: next;
Going to leaf second:

all s1, s2
( 
  Nchanges(Seqnums(s1)) = Nchanges(Seqnums(s2))
  and Nchanges(Seqnums(s1)) < Nchanges(Seqnums(s2))
  and Nchanges(Seqnums(s1)) + 1 = Nchanges(Seqnums(s2))
  and Bounded(Nchanges(Seqnums(s1)), Nchanges(Seqnums(s2)), 1)
  and UniqueMsg(s1) iss UniqueMsg(s2)
  and s1 subseq s2
  and LastBit(Seqnums(s1)) = LastBit(Seqnums(s2))
  imp UniqueMsg(s1) = UniqueMsg(s2))

98 U: apply SubSeqnum;

some s1", s2" (s1" subseq s2" imp Seqnums(s1") subseq Seqnums(s2"))
Automatically search for instantiation? no [confirm]

99 U: put s1"=s1, s2"=s2;

all s1, s2
( 
  s1 subseq s2 and Seqnums(s1) subseq Seqnums(s2)
  and Nchanges(Seqnums(s1)) = Nchanges(Seqnums(s2))
  and Nchanges(Seqnums(s1)) < Nchanges(Seqnums(s2))
  and Nchanges(Seqnums(s1)) + 1 = Nchanges(Seqnums(s2))
  and Bounded(Nchanges(Seqnums(s1)), Nchanges(Seqnums(s2)), 1)
  and UniqueMsg(s1) iss UniqueMsg(s2)
  and LastBit(Seqnums(s1)) = LastBit(Seqnums(s2))
  imp UniqueMsg(s1) = UniqueMsg(s2))

100 U: apply LastEq;

some n1, n2
AFFIRM Type Specifications and Proofs

( n1 subseteq n2 and Bounded(Nchanges(n1), Nchanges(n2), 1)  
  \Rightarrow \text{LastBit}(n1) = \text{LastBit}(n2) \equiv Nchanges(n1) = Nchanges(n2) )

Automatically search for instantiation? yes [confirm]

1/1: (n2 = Seqnum(s2)) and (n1 = Seqnum(s1)).
Proved by chaining and narrowing using the substitution
(n2 = Seqnum(s2)) and (n1 = Seqnum(s1))

TRUE
UniqueEn is awaiting the proof of lemmas BoundedEq1, NcUnique, IssUniqueEq
and LastEq.

101 U: note

LastEq continues the "hard line" of lemmas. ;

102 U: tyr LastEq;
what? Please correct tyr using the list: try or type.
Please correct tyr: try [confirm]

LastEq;
LastEq is untried.

all n1, n2
( n1 subseteq n2 and Bounded(Nchanges(n1), Nchanges(n2), 1)
  \Rightarrow \text{LastBit}(n1) = \text{LastBit}(n2) \equiv Nchanges(n1) = Nchanges(n2) )

103 U: annotate

Stating the right form of this lemma was one of
the most crucial parts of the overall proof.;

104 U: employ Induction(n2);
Case NewSequenceOfBit: Prop(NewSequenceOfBit) remains to be shown.
Case apr: all ss, ii (IH(ss) \Rightarrow Prop(ss apr ii)) remains to be shown.

(NewSequenceOfBit):
all n1
( n1 = NewSequenceOfBit ) and Bounded(Nchanges(n1), 0, 1)
  \Rightarrow \text{LastBit}(n1) = \text{zero} \equiv Nchanges(n1) = 0 )

105 U: replace;

TRUE
Going to leaf apr:

all ss$, ii$, n1
( IH(ss$, S \{ \text{LastEq} \})
  and (n1 = NewSequenceOfBit) or n1 subseteq ss$
  or LessLast(n1) subseteq ss and (Last(n1) = ii$)
  \Rightarrow if LastBit(ss$) = ii$
    then Bounded(Nchanges(n1), Nchanges(ss$), 1)
    \Rightarrow LastBit(n1) = ii$ \equiv Nchanges(n1) = Nchanges(ss$
  else Bounded(Nchanges(n1), Nchanges(ss$) + 1, 1)
    \Rightarrow LastBit(n1) = ii$
    \equiv Nchanges(n1) = Nchanges(ss$) + 1)
106 u: suppose lastbit(sss) = iis;
(yes:)
all sss, iis, n1
( (lastbit(sss) = iis) and ih(sss, 5 {lasteq})
and (n1 = newsequenceofbit) or n1 subseq sss
or lesslast(n1) subseq sss
and last(n1) = iis
and bounded(nchanges(n1), nchanges(sss), 1)
imp lastbit(n1) = iis eqv nchanges(n1) = nchanges(sss))

107 u: replace iis;
all sss, iis, n1
( (lastbit(sss) = iis) and ih(sss, 5 {lasteq})
and (n1 = newsequenceofbit) or n1 subseq sss
or lesslast(n1) subseq sss
and last(n1) = lastbit(sss)
and bounded(nchanges(n1), nchanges(sss), 1)
imp lastbit(n1) = lastbit(sss)
eqv nchanges(n1) = nchanges(sss))

108 u: suppose n1 = newsequenceofbit or n1 subseq sss;
(yes:)
all sss, iis, n1
( (n1 = newsequenceofbit) or n1 subseq sss
and lastbit(sss) = iis
and ih(sss, 5 {lasteq})
and bounded(nchanges(n1), nchanges(sss), 1)
imp lastbit(n1) = lastbit(sss)
eqv nchanges(n1) = nchanges(sss))

109 u: augment n1 subseq sss;
(main:)
all sss, iis, n1
( n1 subseq sss and (lastbit(sss) = iis)
and ih(sss, 5 {lasteq})
and bounded(nchanges(n1), nchanges(sss), 1)
imp lastbit(n1) = lastbit(sss)
eqv nchanges(n1) = nchanges(sss))

110 u: invoke ih;
Automatically search for instantiation? yes [confirm]

1/1: n1' = n1
Proved by chaining and narrowing
using the substitution

n1' = n1

TRUE
Going to leaf thesis:.
all sss, iis, n1
    (n1 = NewSequenceOfBit) and (LastBit(sss) = iis)
    and IH(sss, 5 (LastEq))
    and Bounded(\n changes(n1), \n changes(sss), 1)
    imp n1 subseq sss)

111 U: replace n1:

TRUE
Going to leaf no.:

all sss, iis, n1
    (n1 = NewSequenceOfBit) and ~(n1 subseq sss)
    and LastBit(sss) = iis
    and IH(sss, 5 (LastEq))
    and LessLast(n1) subseq sss
    and Last(n1) = LastBit(sss)
    and Bounded(\n changes(n1), \n changes(sss), 1)
    imp LastBit(n1) = LastBit(sss)
    eqv \n changes(n1) = \n changes(sss))

112 U: employ NormalForm(n1);
Case NewSequenceOfBit: Prop(NewSequenceOfBit) proven.
Case apr: all ss, ii (Prop(ss apr ii)) remains to be shown.
(apr:)
all sss, iis, sss", iis'
    (\((ss apriis) subseq sss")
    and LastBit(sss") = iis"
    and IH(sss", 5 (LastEq))
    and sss' subseq sss"
    and iis = LastBit(sss")
    imp if LastBit(sss) = iis
    then Bounded(\n changes(sss), \n changes(sss"), 1)
    imp \n changes(sss) = \n changes(sss")
    else Bounded(\n changes(sss) + 1, \n changes(sss"), 1)
    imp \n changes(sss) + 1 = \n changes(sss")

113 U: suppose;
(first:)
all sss, iis, sss", iis'
    (\((ss apriis) subseq sss")
    and LastBit(sss") = iis"
    and IH(sss", 5 (LastEq))
    and sss subseq sss"
    and iis = LastBit(sss")
    and LastBit(sss) = iis'
    and Bounded(\n changes(sss), \n changes(sss"), 1)
    imp \n changes(sss) = \n changes(sss")

114 U: replace;
all $s$, $i$, $ss$, $ss'$, $i'$

```
    "((ss apr i) subseq ss")
    and LastHit(ss) = $i$
    and IH(ss', 5 {lastEq})
    and ss subseq ss'
    and $i$ = LastHit(ss)
    and LastHit(ss') = LastHit(ss)
    and Bounded(nchanges(ss), nchanges(ss'), 1)
    imp nchanges(ss) = nchanges(ss')
```

115 U: undo;
replace undone.

116 U: replace $i$;

all $s$, $i$, $ss$, $ss'$, $i'$

```
    "((ss apr LastHit(ss')) subseq ss")
    and LastHit(ss') = $i$
    and IH(ss', 5 {lastEq})
    and ss subseq ss'
    and $i$ = LastHit(ss)
    and LastHit(ss) = LastHit(ss')
    and Bounded(nchanges(ss), nchanges(ss'), 1)
    imp nchanges(ss) = nchanges(ss')
```

117 U: invoke I';
Automatically search for instantiation? yes [confirm]

1/2: $n_1$ = $ss$ apr LastHit($ss'$)
2/2: $n_1$ = $ss$

Proved by chaining and narrowing
using the substitution

$n_1$ = $ss$

TRUE
Going to leat second;

all $s$, $i$, $ss$, $ss'$, $i'$

```
    "((ss apr i) subseq ss")
    and LastHit(ss') = $i$
    and IH(ss', 5 {lastEq})
    and ss subseq ss'
    and $i$ = LastHit(ss)
    and LastHit(ss') = $i$
    and Bounded(nchanges(ss) + 1, nchanges(ss'), 1)
    imp nchanges(ss) + 1 = nchanges(ss')
```

118 U: replace $i$;

all $s$, $i$, $ss$, $ss'$, $i'$

```
    "((ss apr LastHit(ss')) subseq ss")
```
and LastBit(sss) = iis
and IH(sss, 5 (LastEq))
and sss subseq sss
and iis = LastBit(sss)
and LastBit(sss') = LastBit(sss)
and Bounded(Nchanges(sss) + 1, Nchanges(sss'), 1)
implies Nchanges(sss) + 1 = Nchanges(sss'))

119 U: apply SameLastBit;

some n1, n2, b
( n1 subseq n2 and (b = LastBit(n2))
implies (n1 apr b) subseq n2 or (LastBit(n1) = LastBit(n2)))
Automatically search for instantiation? Yes [confirm]

1/6: (n2 = sss') and (n1 = sss apr LastBit(sss'))
2/6: (n2 = sss') and (n1 = sss)
 1/3: b = iis
 2/3: b = LastBit(sss)
 3/3: b = LastBit(sss')

Proved by chaining and narrowing
using the substitution

(n2 = sss') and (n1 = sss) and (b = LastBit(sss'))

TRUE

120 U: next;
Going to leaf no.: 121

all sss, iis, n1
( LastBit(sss) = iis and IH(sss, 5 (LastEq))
and (n1 = NewSequenceOfBit) or n1 subseq sss
or LessLast(n1) subseq sss
and Last(n1) = iis
and Bounded(Nchanges(n1), Nchanges(sss) + 1, 1)
implies LastBit(n1) = iis
equiv Nchanges(n1) = Nchanges(sss) + 1

121 U: suppose n1 = NewSequenceOfBit or n1 subseq sss;

(... suppose aborted; effects undone)

122 U: suppose n1 = NewSequenceOfBit or n1 subseq sss;
(yes)
all sss, iis, n1
( n1 = NewSequenceOfBit or n1 subseq sss
and LastBit(sss) = iis
and IH(sss, 5 (LastEq))
and Bounded(Nchanges(n1), Nchanges(sss) + 1, 1)
implies LastBit(n1) = iis
equiv Nchanges(n1) = Nchanges(sss) + 1)
123 U: augment n1 subseq sss;
(main:
all sss, iis, n1
  ( n1 subseq sss and (LastBit(sss) "= iis"
            and IH(sss, 5) (LastEq))
    and Bounded(Nchanges(n1), Nchanges(sss) + 1, 1)
    imp LastBit(n1) = iis
    eqv Nchanges(n1) = Nchanges(sss) + 1)

124 U: apply BitSubRound;

some n1', n2
  ( n1" subseq n2 and Bounded(Nchanges(n1"), Nchanges(n2) + 1, 1)
    imp Bounded(Nchanges(n1"), "changes(n2), 1) and (Nchanges(n1") = Nchanges(n2))
    Automatically search for instantiation? no [confirm]

125 U: put n1"=n1, n2=sss;

all sss, iis, n1
  ( n1 subseq sss
    and Bounded(Nchanges(n1), Nchanges(sss) + 1, 1)
    and Bounded(Nchanges(n1), Nchanges(sss), 1)
    and Nchanges(n1) = Nchanges(sss)
    and LastBit(sss) "= iis
    and IH(sss, 5) (LastEq))
    imp LastBit(n1) "= iis

126 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/1: n1" = n1
Unsuccessful.

all sss, iis, n1 (some n1'
  ( n1 subseq sss
    and Bounded(Nchanges(n1), Nchanges(sss) + 1, 1)
    and Bounded(Nchanges(n1), Nchanges(sss), 1)
    and Nchanges(n1) = Nchanges(sss)
    imp LastBit(sss) = iis
    or if n1' subseq sss
      and Bounded(Nchanges(n1"), Nchanges(sss), 1)
      then if Nchanges(n1") = Nchanges(sss)
        then LastBit(n1") = LastBit(sss)
        imp LastBit(n1") = iis
        else LastBit(n1") = LastBit(sss)
      else LastBit(n1") = iis
    )

127 U: put n1"=n1;
AFFIRM Type Specifications and Proofs

all sss, iis, n1
  ( n1 subset ss
    and Bounded(\text{Nchanges}(n1), \text{Nchanges}(ss) + 1, 1)
    and Bounded(\text{Nchanges}(n1), \text{Nchanges}(ss), 1)
    and \text{Nchanges}(n1) = \text{Nchanges}(ss)
    and \text{LastBit}(ss) \text{ } \text{= iis}
    and \text{LastBit}(n1) = \text{LastBit}(ss)
  )
imp \text{LastBit}(n1) \text{ } \text{= iis}

128 U: replace;

TRUE
Going to leaf thesis:

all sss, iis, n1
  ( n1 = \text{NewSequenceOfBit} \text{ and (LastBit(ss) } \text{= iis})
    and IH(sss, 5 {LastEq})
    and Bounded(\text{Nchanges}(n1), \text{Nchanges}(ss) + 1, 1)
  )
imp n1 subset ss

129 U: replace n1;

TRUE
Going to leaf no:.

all sss, iis, n1
  ( n1 \text{ = NewSequenceOfBit} \text{ and (n1 subset ss)}
    and \text{LastBit(ss)} \text{ = iis}
    and IH(sss, 5 {LastEq})
    and \text{LessLast}(n1) subset ss
    and Last(n1) = iis
    and Bounded(\text{Nchanges}(n1), \text{Nchanges}(ss) + 1, 1)
  )
imp \text{LastBit}(n1) = iis
eqv \text{Nchanges}(n1) = \text{Nchanges}(ss) + 1

130 U: employ NormalForm(n1);
Case \text{NewSequenceOfBit}: Prop(\text{NewSequenceOfBit}) proven.
Case apr: all ss, ii (Prop(ss apr ii)) remains to be shown.

(aprt)
all sss, iis, sss', iis'
  ( ((ssss apr iiss) subset ss)
    and \text{LastBit(sss')} \text{ = iis'}
    and IH(sss', 5 {LastEq})
    and ss subset ss'
    and iis = iis'
  )
imp if \text{LastBit}(sss) = iis
then Bounded(\text{Nchanges}(sss), \text{Nchanges}(ssss') + 1, 1)
  imp \text{Nchanges}(ssss') = \text{Nchanges}(ssss') + 1
else Bounded(\text{Nchanges}(sss), \text{Nchanges}(ssss'), 1)
  imp \text{Nchanges}(ssss') + 1 = \text{Nchanges}(ssss') + 1

131 U: replace iis'
all sss, iis, sss', iis'
  (   ((ssss apr iis) subseq sss')
      and LastBit(sss') = iis
      and IH(sss', 5 {LastEq})
      and sss' subseq sss
      and iis = iis'
  )
imp if LastBit(sss) = iis
    then Bounded(nchanges(sss),
      nchanges(sss') + 1, 1)
    imp nchanges(sss) = nchanges(sss') + 1
  else Bounded(nchanges(sss), nchanges(sss') + 1, 1)
    imp nchanges(sss) + 1 = nchanges(sss') + 1

132 U: suppose;
  (first:)
all sss, iis, sss', iis'
  (   ((ssss apr iis) subseq sss')
      and LastBit(sss') = iis
      and IH(sss', 5 {LastEq})
      and sss' subseq sss
      and iis = iis'
      and LastBit(sss) = iis
      and Bounded(nchanges(sss), nchanges(sss') + 1, 1)
  )
imp nchanges(sss) = nchanges(sss') + 1)

133 U: swap (all:);
all sss, iis, sss', iis'
  (   ((ssss apr iis) subseq sss')
      and iis' = LastBit(sss')
      and IH(sss', 5 {LastEq})
      and sss' subseq sss
      and iis = iis'
      and iis = LastBit(sss)
      and Bounded(nchanges(sss), nchanges(sss') + 1, 1)
  )
imp nchanges(sss') + 1 = nchanges(sss)

134 U: replace;
all sss, iis, sss', iis'
  (   ((ssss apr LastBit(sss)) subseq sss')
      and LastBit(sss) = LastBit(sss')
      and IH(sss', 5 {LastEq})
      and sss' subseq sss
      and iis' = LastBit(sss')
      and iis = LastBit(sss)
      and Bounded(nchanges(sss), nchanges(sss') + 1, 1)
  )
imp nchanges(sss') + 1 = nchanges(sss)

135 U: apply BitSubRound;
some n1, n2
    ( n1 < n2 and Bounded(nchanges(n1), nchanges(n2) + 1, 1)
        imp Bounded(nchanges(n1), nchanges(n2), 1) and (nchanges(n1) = nchanges(n2))
    )
Automatically search for instantiation? no [confirm]

136 U: put n1=sss, n2=sss"

all sss, iis, sss", iis"
    ( sss subseq sss"
        and Bounded("nchanges(sss), nchanges(sss") + 1, 1)
        and Bounded("nchanges(sss), nchanges(sss")", 1)
        and "nchanges(sss) = nchanges(sss")
        and "((sss and LastBit(sss)) subseq sss")
        and LastBit(sss) = LastBit(sss")
        and IH(sss", 5 (LastEq))
        and iis" = LastBit(sss)
        imp iis" = LastBit(sss")
    )

137 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/2: n1 = sss
Proved by chaining and narrowing
using the substitution
n1 = sss
TRUE.
Going to leaf second:

all sss, iis, sss", iis"
    ( "((sss and iis) subseq sss")
        and LastBit(sss") = iis
        and IH(sss", 5 (LastEq))
        and sss subseq sss"
        and iis = iis"
        and LastBit(sss) = iis
        and Bounded("nchanges(sss), nchanges(sss")", 1)
        imp nchanges(sss) + 1 = nchanges(sss") + 1)

138 U: apply bivalued;
(bivalued => Bivalued)

some b, b1, b2 ((b=b1) or (b=b2) or (b1=b2))
Automatically search for instantiation? no [confirm]

139 U: put b=iis, b1=LastBit(sss), b2=LastBit(sss");

all sss, iis, sss", iis"
    ( iis" = LastBit(sss)
        and iis" = LastBit(sss")
    )
and LastBit(sss) = LastBit(sss')
and "((sss apr iis) subseq sss")
and IH(sss", 5 (LastKo))
and sss subseq sss"
and iis = iis"
and Bounded(Nchanges(sss), Nchanges(sss"), 1)
imp Nchanges(sss) + 1 = Nchanges(sss") + 1)

140 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/2: n1 = sss
2/2: n1 = sss apr iis
Unsuccessful.

all sss, iis, sss", iis" (some n1
 (    iis" = LastBit(sss)
    and iis" = LastBit(sss")
    and LastBit(sss) = LastBit(sss")
    imp "((sss apr iis) subseq sss"
    or if n1 subseq sss"
    and Bounded(Nchanges(n1), Nchanges(sss"), 1)
    then if "Nchanges(n1) = Nchanges(sss")
        then LastBit(n1) = LastBit(sss")
        and sss subseq sss"
        and iis = iis"
        and Bounded(Nchanges(sss), Nchanges(sss"), 1)
        imp Nchanges(sss) + 1
            = Nchanges(sss") + 1
    else LastBit(n1) "= LastBit(sss")
        and sss subseq sss"
        and iis = iis"
        and Bounded(Nchanges(sss), Nchanges(sss"), 1)
        imp Nchanges(sss) + 1
            = Nchanges(sss") + 1
    else sss" subseq sss"
        and iis = iis"
        and Bounded(Nchanges(sss), Nchanges(sss"), 1)
        imp Nchanges(sss) + 1
            = Nchanges(sss") + 1)

141 U: choose 1;

1/2: n1 = sss
LastUg is awaiting the proof of lemmas BitSubBound and A1Value.

TRUE
Going to lemma BitSubBound.
BitSubBound is untried.
all n1, n2
  ( n1 <= n2 and \text{Bounded}(\text{Nchanges(n1), Nchanges(n2) + 1, 1})
    \implies \text{Bounded}(\text{Nchanges(n1), Nchanges(n2), 1})
    \text{and (Nchanges(n1) = Nchanges(n2)})
)

142 U: note
  The bump has been passed and the rest should be clear sailing.

143 U: apply Subless:

some n1', n2' (n1' <= n2' \implies \text{Nchanges(n1')} <= \text{Nchanges(n2')})
Automatically search for instantiation? no [confirm]

144 U: put n1'=n1, n2'=n2;

all n1, n2
  ( n1 <= n2 and \text{Nchanges(n1)} <= \text{Nchanges(n2)})
    \implies \text{Bounded}(\text{Nchanges(n1), Nchanges(n2) + 1, 1})
    \text{and (Nchanges(n1) = Nchanges(n2)})

145 U: invoke Bounded 1all1;
BitSubBound proved.

TRUE
Going to unproven ancestor BiValued,
BiValued is untried.

all b, b1, b2 ((b=b1) or (b=b2) or (b1=b2))

146 U: employ NormalForm(b);
Case zero: Prop(zero) remains to be shown.
Case one: Prop(one) remains to be shown.
  (zero): all b1, b2 ((zero = b1) or (zero = b2) or (b1=b2))

147 U: employ NormalForm(b1);
Case zero: Prop(zero) proven.
Case one: Prop(one) remains to be shown.
  (one): all b2 ((zero = b2) or (one = b2))

148 U: employ NormalForm(b2);
Case zero: Prop(zero) proven.
Case one: Prop(one) proven.

TRUE
Going to leaf one:

all b1, b2 ((one = b1) or (one = b2) or (b1=b2))
149 U: employ NormalForm(b1);
Case zero: Prop(zero) remains to be shown.
Case one: Prop(one) proven.
(zero;)
all b2 ((one = b2) or (zero = b2))

150 U: employ NormalForm(b2);
Case zero: Prop(zero) proven.
Case one: Prop(one) proven.
BValued proved.
LastEq proved.

TRUE
There's more than one unproven ancestor. You may pick one of NoUnique, BoundedBy1, IssLenEq, SameLastBit2, or LastEq.

151 U: try IssLenEq;
IssLenEq is untried.

all m1, m2 (m1 iss m2 and (Length(m1) = Length(m2)) eqv m1=m2)

152 U: employ Induction(m2);
Case NewSequenceOfElemType: Prop(NewSequenceOfElemType) remains to be shown.
Case apr: all ss, ii (IH(ss) imp Prop(ss apr ii)) remains to be shown.
(NewSequenceOfElemType;
all m1 (m1 = NewSequenceOfElemType imp Length(m1) = 0)

153 U: replace;

TRUE
Going to leaf apr:

all sss$, iiss$, m1
(IH(sss$, 14 {IssLenEq})
 imprint (m1 = NewSequenceOfElemType) or m1 iss sss$
 or (LessLast(m1) = sss$ and (Last(m1) = iiss$)
 and Length(m1) = Length(sss$) + 1
 eqv m1 = sss$ apr iiss$)

154 U: replace:

all sss$, iiss$, m1
(IH(sss$, 14 {IssLenEq})
 and m1 ~= sss$ apr iiss$
 and (m1 = NewSequenceOfElemType) or m1 iss sss$
 or LessLast(m1) = sss$
 and Last(m1) = iiss$
 imprint Length(m1) ~= Length(sss$) + 1)

155 U: split;
(first:)
all sss$, iiss$, m1
AFFIRM Type Specifications and Proofs

(  IH(ssss, 14 (IssLenEq))
    and  m1 ~= ssss aor iiss
    and  m1 = NewSequenceOfElementType
    imp Length(m1) ~= Length(ssss) + 1)

156 U: replace:

all ssss, iiss, m1
  (  IH(ssss, 14 (IssLenEq)) and (m1 = NewSequenceOfElementType)
    imp 0 ~= Length(ssss) + 1)

157 U: apply LenNonneg, Length(m) ~> 0;

some m (0 <= Length(m))
Automatically search for instantiation? yes [confirm]
Unsuccessful.

158 U: put m=ssss;

TRUE
Going to leaf second:

all ssss, iiss, m1
  (  IH(ssss, 14 (IssLenEq))
    and  m1 ~= ssss aor iiss
    and  m1 = NewSequenceOfElementType
    and  m1 iss ssss
    or  LessLast(m1) = ssss
        and  Last(m1) = iiss
    imp Length(m1) ~= Length(ssss) + 1)

159 U: split;
(first:)

all ssss, iiss, m1
  (  IH(ssss, 14 (IssLenEq))
    and  m1 ~= ssss aor iiss
    and  m1 = NewSequenceOfElementType
    and  m1 iss ssss
    imp Length(m1) ~= Length(ssss) + 1)

160 U: apply IssLenLe;

some m1", m2 (m1" iss m2 imp Length(m1") <= Length(m2))
Automatically search for instantiation? no [confirm]

161 U: put m1"=m1, m2=ssss;

all ssss, iiss, m1
  (  m1 iss ssss and (Length(m1) <= Length(ssss))
    and IH(ssss, 14 (IssLenEq))
    imp  m1 = ssss aor iiss
    or  m1 = NewSequenceOfElementType


or Length(m1) \(=\) Length(ssss) + 1)

162 U: apply IntFact1, k1 \(\leq\) k2 imp k1 \(=\) k2 + 1;

some k1, k2 (k1 \(\leq\) k2 imp k1 \(=\) k2 + 1)
Automatically search for instantiation? yes [confirm]

1/1: (k2 = Length(ssss)) and (k1 = Length(m1))
Proved by chaining and narrowing
using the substitution

(k2 = Length(ssss)) and (k1 = Length(m1))

TRUE

163 U: next;
Going to leaf second:

all ssss, iiss, m1
(    TH(ssss, 14 {IssLenEq})
    and m1 \(=\) ssss apr iiss
    and m1 \(=\) NewSequenceOfElemType
    and \(\neg\)(m1 iss ssss)
    and LessLast(m1) = ssss
    and Last(m1) = iiss
    imp Length(m1) \(=\) Length(ssss) + 1)

164 U: employ NormalForm(m1);
Case NewSequenceOfElemType: Prop(NewSequenceOfElemType) proven.
Case apr: all ss, ii (Prop(ss apr ii)) proven.
IssLenEq is awaiting the proof of lemmas LenNonneg, IssLenLe, and IntFact1.

TRUE
Going to lemma IssLenLe.
IssLenLe is untried.

all m1, m2 (m1 iss m2 imp Length(m1) \(\leq\) Length(m2))

165 U: employ Induction(m2);
Case NewSequenceOfElemType: Prop(NewSequenceOfElemType) remains to be shown.
Case apr: all ss, ii (IH(ss) imp Prop(ss apr ii)) remains to be shown.
  (NewSequenceOfElemType)
all m1 (m1 = NewSequenceOfElemType imp Length(m1) \(\leq\) 0)

166 U: replace;
TRUE
Going to leaf apr:.

all ssss, iiss, m1
(    TH(ssss, 15 {IssLenLe})
    and \(\neg\)(m1 = NewSequenceOfElemType) or m1 iss ssss
or (LessLast(m1) = sss$) and (Last(m1) = iiss$)
imp Length(m1) <= Length(ssss) + 1)

167 U: suppose m1 = NewSequenceOfElemType or m1 iss sss$;
(yes:)
all sss$, iiss, m1
( (m1 = NewSequenceOfElemType) or m1 iss sss$
    and IH(ssss, 15 (IssLenLe))
    imp Length(m1) <= Length(ssss) + 1)

168 U: augment m1 iss sss$;
(main:)
all sss$, iiss, m1
( m1 iss sss$ and IH(ssss, 15 (IssLenLe))
    imp Length(m1) <= Length(ssss) + 1)

169 U: invoke IH;
Automatically search for instantiation? yes [confirm]

1/1: m1" = m1
Unsuccessful.
all sss$, iiss, m1 (some m1"
( m1 iss sss$
    and m1" iss sss$ imp Length(m1") <= Length(ssss)
    imp Length(m1) <= Length(ssss) + 1))

170 U: put m1" = m1;

all sss$, iiss, m1
( m1 iss sss$ and (Length(m1) <= Length(ssss))
    imp Length(m1) <= Length(ssss) + 1)

171 U: apply IntFact2, k1 <= k2 imp k1 <= k2 + 1;
some k1, k2 (k1 <= k2 imp k1 <= k2+1)
Automatically search for instantiation? yes [confirm]

1/2: (k2 = Length(ssss)) and (k1 = Length(m1))
Proved by chaining and narrowing
using the substitution
(k2 = Length(ssss)) and (k1 = Length(m1))
TRUE

172 U: next;
Going to leaf thesis:
all sss$, iiss, m1
( (m1 = NewSequenceOfElemType) and IH(ssss, 15 (IssLenLe))
    imp m1 iss sss$)
173 U: replace;

TRUE
Going to leaf no:.

all sss$, iiss$, m1 (m1 $= NewSequenceOfElemType$) and "(m1 iss sss$) and IH(ssss$, 15 (IssLenLe}) and LessLast(m1) = sss$ and Last(m1) = iiss$
imp Length(m1) $\leq$ Length(ssss$) + 1)

174 U: employ NormalForm(m1);
Case NewSequenceOfElemType: Prop(NewSequenceOfElemType) proven.
Case apr: all ss, ii (Prop(ss apr ii)) remains to be shown.
(apri)
all sss$, iiss$, sss", iiss"
("
"(ssss apr iiss$) iss sss"") and IH(ssss", 15 (IssLenLe}) and sss$ = sss$
and iiss$ = iiss$
imp Length(ssss) $\leq$ Length(ssss")

175 U: replace;
what? Please correct replace using the list: read or replace. Please correct replace: replace [confirm].
IssLenLe is awaiting the proof of lemma IntFact2.

TRUE
Going to lemma IntFact2.
IntFact2 is untried.
IntFact2 proved.
IssLenLe proved.

TRUE
There's more than one unproven ancestor. You may pick one of LenNonneg or IntFact1.

176 U: try IntFact1;
IntFact1 is untried.
IntFact1 proved.

TRUE
Going to unproven ancestor LenNonneg.
LenNonneg is untried.

all m (0 $\leq$ Length(m))

177 U: annotate Proved in Affirm type library. ;

178 U: assume LenNonneg;
ISSLENQ proved.

179 U: next;
There's more than one unproven ancestor. You may pick one of NcUnique or BoundedBy1.

180 U: try NcUnique;
(NcUnique => NcUnique)
NcUnique is untried.

all s (NcChanges(Segnums(s)) = Length(UniqueMsg(s)))

181 U: employ Induction(s);
Case NewSequenceIfPacket: Prop(NewSequenceIfPacket) proven.
Case aor: all ss, ii (IH(ss) imp Prop(ss aor ii)) remains to be shown.
(aor:)
all ss", ii"
( IH(ss", 1) (NcUnique))
  imp if LastBit(Segnums(ss")) = Seqno(ii"
    then NcChanges(Segnums(ss")) = Length(UniqueMsg(ss"))
    else NcChanges(Segnums(ss")) + 1 = Length(UniqueMsg(ss")) + 1)

182 U: invoke IH;
NcUnique proved.

TRUE
Going to unproven ancestor BoundedBy1.
BoundedBy1 is untried.

all k1, k2
( Bounded(k1, k2, 1)
  equiv (k1=k2) or (k1<k2) and (k1+1 = k2))

183 U: invoke Bounded;
BoundedBy1 proved.
UniqueEq proved.

TRUE
There's more than one unproven ancestor. You may pick one of SameLastBit2 or LastSeq.

184 U: try SameLastBit2;
SameLastBit2 is untried.

all s1, s2, o
( s1 subseq s2 and (o = Last(s2))
  imp (s1 aor o) subseq s2 or (LastBit(Segnums(s1)) = LastBit(Segnums(s2)))

185 U: employ NormalForm(s2);
Case NewSequenceIfPacket: Prop(NewSequenceIfPacket) remains to be shown.
Case aor: all ss, ii (Prop(ss aor ii)) remains to be shown.
(NewSequenceIfPacket:)


all s1, p
   (s1 = NewSequenceOfPacket) and (o = Last(NewSequenceOfPacket))
   imp LastBit(Seqnums(s1)) = zero

186 U: replace;

TRUE
Going to leaf apr:

all ss", ii", s1, p
   (if s1 = NewSequenceOfPacket
      then p=ii"
         imp (s1 apr p) subseq ss" or s1 subseq ss"
         or LastBit(Seqnums(s1)) = seqno(ii")
      else "(s1 subseq ss") and LessLast(s1) subseq ss"
         and Last(s1) = ii"
         and p=ii"
         imp (s1 apr p) subseq ss" or (LastBit(Seqnums(s1)) = seqno(ii"))

187 U: employ NormalForm(s1);
Case NewSequenceOfPacket: Prop(NewSequenceOfPacket) proven.
Case apr: all ss, ii (Prop(ss apr ii)) remains to be shown.
   (apr:)
all ss", ii", ss, ii, p
   ("((ss" apr ii") subseq ss) and ss" subseq ss
      and ii"=ii
      and p=ii"
   imp ((ss" apr ii") apr p) subseq ss or (seqno(ii") = seqno(ii"))

188 U: replace;
SameLastBit2 proved.

TRUE
Going to unproven ancestor LastSeq.
LastSeq is untried.

all s ((s = NewSequenceOfPacket) or (seqno(Last(s)) = LastBit(Seqnums(s)))))

189 U: employ NormalForm(s);
(employ => employ)
Case NewSequenceOfPacket: Prop(NewSequenceOfPacket) proven.
Case apr: all ss, ii (Prop(ss apr ii)) proven.
LastSeq proved.
SubToSeq proved.

TRUE
The proof of this part is finished.

190 U: print status;
No theorems are untried.
No theorems are tried.
The assumed theorem is LemMonSeq.
AFFIRM Type Specifications and Proofs

No theorems are awaiting lemma proof.
The proved theorems are BitSubBound, BiValued, BoundedBy1, IntFact1, IntFact2, IssLenEq, IssLenLe, LastEq, LastSeq, McNonneg, McUnique, PktSubBound, SameLastBit, SameLastBit2, SubLess, SubSeqNums, SubTotag, and UniqueEq.

191 U: quit:
Automatically summarize the proof attempts? yes (confirm)

```plaintext
 theorem LastSeq,
   s `=` NewSequenceOfPacket
   imp seqno(Last(s)) `=` LastBit(Seqnums(s));
```

```plaintext
proof tree:
189:: LastSeq
  employ NormalForm(s) {proved by Divito using AFFIRM 120 on 12-Feb-81 in transcript
  <DIVITO>AFFLOG.12-FEB-81.1}
  -> NewSequenceOfPacket:
    Immediate
  -> apr:
    Immediate
```

```plaintext
 theorem SameLastBit2, s1 subseq s2 and p `=` Last(s2)
 and `~`((s1 apr p) subseq s2)
 imp LastBit(Seqnums(s1)) `=` LastBit(Seqnums(s2));
```

```plaintext
proof tree:
185:: SameLastBit2
  employ NormalForm(s2) {proved by Divito using AFFIRM 120 on 12-Feb-81 in transcript
  <DIVITO>AFFLOG.12-FEB-81.1}
  186: NewSequenceOfPacket:
    209 replace
      (proven!)
  187: apr:
    210 employ NormalForm(s1)
    -> NewSequenceOfPacket:
      Immediate
  188: apr:
    211 replace
      (proven!)
```

```plaintext
 theorem BoundedBy1, Bounded(k1, k2, 1)
 eqv (k1 `=` k2) or (k1 + 1 `=` k2);
```

```plaintext
proof tree:
182:: BoundedBy1
```
normint (proved by Divito using AFFIRM 120 on 12-Feb-81 in transcript <DIVITO>AFFLOG,12-FEB-81.1)

183: 207 invoke Bounded
183: 208 normint

-> (proven!)

```text
theorem mcunique, nchanges(sequenc(s)) = length(uniquemsgs(s));
```

```text
proof tree:
191:1 mcunique
    employ Induction(s) (proved by Divito using AFFIRM 120 on 12-Feb-81 in transcript <DIVITO>AFFLOG,12-FEB-81.1)

-> NewSequenceOfPacket:
    Immediate
181: apr:
    204 cases
182: 205 invoke IH
182: 206 normint

-> (proven!)
```

```text
theorem lennonneg, length(m) >= 0;
```

```text
theorem intfact1, k1 <= k2 imp k1 = k2+1;
```

```text
proof tree:
176:1 intfact1
    normint (proved by Divito using AFFIRM 120 on 12-Feb-81 in transcript <DIVITO>AFFLOG,12-FEB-81.1)

-> (proven!)
```

```text
theorem intfact2, k1 <= k2 imp k1 <= k2+1;
```

```text
proof tree:
175:1 intfact2
    normint (proved by Divito using AFFIRM 120 on 12-Feb-81 in transcript <DIVITO>AFFLOG,12-FEB-81.1)

-> (proven!)
```
theorem IssLenLe, m1 iss m2 imp Length(m1) <= Length(m2);
IssLenLe uses IntFact2!.

proof tree:
165: ! IssLenLe
   employ Induction(m2) (proved by Divito using AFFIRM 120 on
   12-Feb-81 in transcript
   <DIVITO> AFFLOG, 12-FEB-81, 1)

166: NewSequenceOfElemType:
   192 replace
   (proven!)
167: apr:
   193 suppose m1 = NewSequenceOfElemType
   or m1 iss ssss
168: yes:
   194 augment m1 iss ssss
   main:
   196 invoke IH
   198 put m1' = m1
171: apply IntFact2
171: put k2 = Length(ssss)
   and k1 = Length(m1) {search}
171: (proven!)
173: thesis:{IssLenLe, apr:, yes:}
   197 replace
   (proven!)
174: no:{IssLenLe, apr:}
   195 employ NormalForm(m1)
   NewSequenceOfElemType:
   Immediate
175: apr:
   203 replace
   (proven!)

theorem IssLenEq, m1 iss m2 and (Length(m1) = Length(m2))
equiv m1=m2;
IssLenEq uses LenNonneg?, IssLenLe!, and IntFact1!.

proof tree:
152: ! IssLenEq
   employ Induction(m2) (proved by Divito using AFFIRM 120 on
   12-Feb-81 in transcript
   <DIVITO> AFFLOG, 12-FEB-81, 1)

153: NewSequenceOfElemType:
   176 replace
   (proven!)
154: apr:
177 replace
178 split
156:
  first:
  179 replace
157:
  181 apply LenNonneg
158:
  183 put m==sss
158:
  184 normint
  (proven!)
159:
  second:{IssLenEq, apr;}
180 split
160:
  first:
    185 apply IssLenLe
161:
    187 put (m1==m1) and (m2==sss)
162:
    188 apply IntFact1
162:
    190 put k2 = Length(sss)
    and k1 = Length(m1) {search}
162:
  (proven!)
164:
  second:{IssLenEq, apr; , second!}
186 employ NormalForm(m1)
  NewSequenceOfElemType:
    Immediate
  aor:
    Immediate

theorem BiValued, (a == b1) and (b == b2)
  implication b1==b2;

proof tree:
146: BiValued
    employ NormalForm(b) {proved by Divito using AFFIRM 120 on
    12-Feb-81 in transcript
    <DIVITO>AFFLOG, 12-FEB-81.1}
147: zero:
    172 employ NormalForm(b1)
    zero:
      Immediate
148: one:
    174 employ NormalForm(b2)
    zero:
      Immediate
    one:
      Immediate
149: one:{BiValued}
    173 employ NormalForm(b1)
150: zero:
    175 employ NormalForm(b2)
    zero:
      Immediate
    one:
AFFIRM Type Specifications and Proofs

Immediate

one:

Immediate

theorem BitSubBound,

\[ n1 \subseteq n2 \]

and \( \text{Bounded}(\text{Nchanges}(n1), \text{Nchanges}(n2) + 1, 1) \)

implies \( \text{Bounded}(\text{Nchanges}(n1), \text{Nchanges}(n2), 1) \)

and \( \text{Nchanges}(n1) = \text{Nchanges}(n2) \);

BitSubBound uses SubLess!.

proof tree:

143:1 BitSubBound

apply SubLess \( \text{proved by Divito using AFFIRM 120 on 12-Feb-81} \)

in transcript \(<\text{DIVITO}\text{AFFLOG.12-FEB-81.1}\>\)

144:169 put \( (n1' = n1) \) and \( (n2' = n2) \)

145:170 invoke Bounded 1 all 1

145:171 normint

\( \rightarrow \) (proven!)

theorem LastEq,

\[ n1 \subseteq n2 \]

and \( \text{Bounded}(\text{Nchanges}(n1), \text{Nchanges}(n2), 1) \)

implies \( \text{LastBit}(n1) = \text{LastBit}(n2) \) for \( \text{Nchanges}(n1) = \text{Nchanges}(n2) \);

LastEq uses SameLastBit!, BitSubBound!, and BiValued!.

proof tree:

104:1 LastEq

employ Induction(n2) \( \text{proved by Divito using AFFIRM 120 on} \)

12-Feb-81 \( \text{in transcript} \)

\(<\text{DIVITO}\text{AFFLOG.12-FEB-81.1}\>\)

105: NewSequenceOfHit:

122 replace

\( \rightarrow \) (proven!)

105: apr:

123 cases

106:124 suppose LastHit(sss) = iis

107: yes:

125 replace iis

127 suppose \( n1 = \text{NewSequenceOfHit} \)

or \( n1 \subseteq sss \)

108: yes:

128 augment \( n1 \subseteq sss \)

main:

130 invoke IH

132 put \( n1' = n1 \) (search)

110: (proven!)
thesis:
131 replace nl
(proven!)

->

no:{LastEq, apr:, yes:}
129 employ NormalForm(n1)

NewSequenceOfBit:
Immediate

apr:
134 cases
135 split
136 first:
136 replace iis
138 invoke IH
139 put n1 = sss {search}
(proven!)

second:{LastEq, apr:, yes:, no:, apr:}
137 replace iis
141 apply SameLastBit
142 put n2 = sss'
and n1 = sss
and b = LastBit(sss") {search}
(proven!)

no:{LastEq, apr:}
126 suppose n1 = NewSequenceOfBit
or n1 subseq sss
yes:
144 augment nl subseq sss

main:
146 apply BitSuperRound
148 put (n1"=n1) and (n2=sss)
149 normint
150 invoke IH
151 put nl="n1
152 replace
(proven!)

thesis:{LastEq, apr:, no:, yes:}
147 replace nl
(proven!)

->

no:{LastEq, apr:, no:}
145 employ NormalForm(n1)

NewSequenceOfBit:
Immediate

apr:
153 cases
154 replace iis'
155 split
156 swap I all I
158 replace
159 apply BitSuperRound
160 put (n1=sss) and (n2=sss")
161 normint
AFFIRM Type Specifications and Proofs

137: 162 invoke IH
137: 163 put n1 = sss {search}
137: (proven!)
138: second:{LastEq, apr1, not, nor, apr1}
157 apply BIVAlued
139: 165 put b1=iss
165 and b1 = LastBit(sss)
165 and b2 = LastBit(sss')
140: 166 invoke IH
141: 167 put n1 = sss {choose}
141: 168 normint
141: (proven!)

theorem UniqueEq,
UniqMsg(s1) iss UniqueMsg(s2)
and s1 subseq s2
and Bounded(Nchanges(Seqnums(s1)),
Nchanges(Seqnums(s2)), 1)
and LastBit(Seqnums(s1)) = LastBit(Seqnums(s2))
imp UniqueMsg(s1) = UniqueMsg(s2);
UniqueEq uses BoundedEq!, NcUnique!, IssLenEq!, SubSeqnum!, and LastEq!.

proof tree:
99: 1 UniqueEq
   apply BoundedEq! (proved by Divito using AFFIRM 120 on
   12-Feb-81 in transcript
   <DIVITO>AFFLOG,12-FEB-81.1)
90: 107 put k2 = Nchanges(Seqnums(s2))
90: 108 normint
91: 109 snlit
92: first:
   110 apply NcUnique
   112 apply NcUnique
94: 113 put (s=s1) and (s''=s2)
95: replace
96: 115 apply IssLenEq
96: 116 put (m2 = UniqueMsg(s2)) and (m1 = UniqueMsg(s1)) {search}
96: (proven!)
98: second:{UniqueEq}
   111 apply SubSeqnum
99: 118 put (s1''=s1) and (s2''=s2)
100: 119 apply LastEq
100: 120 put (n2 = Seqnums(s2)) and (n1 = Seqnums(s1)) {search}
100: (proven!)
theorem SubSequnum, s1 subset s2 imp Sequums(s1) subset Sequms(s2);

proof tree:
78:!! SubSequnum
    employ Induction(s2) (proved by Divito using AFFIRM 120 on 12-Feb-81 in transcript <DIVITO>AFFLOG,12-FEB-81.1)

79: NewSequenceOfPacket:
    95 replace
    (proven!)

80: apr:
    96 split

91: first:
    97 replace
    (proven!)

82: second:
    99 split

83: first:
    99 invoke IH

83: 101 put s1" = s1 (search)
83: (proven!)

84: second:
    100 employ NormalForm(s1)

85: NewSequenceOfPacket:
    Immediate

86: apr:
    103 replace

86: 104 invoke IH
86: 105 put s1 = ss" (search)
86: (proven!)

theorem NcNonneg, Nchanges(n) >= 0;

proof tree:
76:!! NcNonneg
    employ Induction(n) (proved by Divito using AFFIRM 120 on 12-Feb-81 in transcript <DIVITO>AFFLOG,12-FEB-81.1)

76: NewSequenceOfBit:
    Immediate

76: apr:
    92 cases

77: 93 invoke IH

77: 94 normint

77: (proven!)
theorem SameLastBit, n1 subseq n2 and b = LastBit(n2) and "((n1 apr b) subseq n2) imp LastBit(n1) = LastBit(n2);"

proof tree:
73: SameLastBit
   employ NormalForm(n2) {proved by Divito using AFFIRM 120 on 12-Feb-91 in transcript
   〈DIVITO〉AFFLOG.12-FEB-91.1}

74: NewSequenceOfBit:
   90 replace
   (proven!)
75: apr:
   91 employ NormalForm(n1)
   -> NewSequenceOfBit:
      Immediate
   -> apr:
      Immediate

theorem SubLess, n1 subseq n2 imp Nchanges(n1) <= Nchanges(n2);
SubLess uses MNonneg and SameLastBit!.

proof tree:
57: SubLess
   employ Induction(n2) {proved by Divito using AFFIRM 120 on 12-Feb-91 in transcript
   〈DIVITO〉AFFLOG.12-FEB-91.1}

58: NewSequenceOfBit:
   70 replace
   (proven!)
58: apr:
   71 cases
   72 split
   first:
   73 apply MNonneg
   75 put n=ss
   76 replace
   77 normint
   -> (proven!)
63: second: {SubLess, apr:}
   74 split
   first:
   78 invoke IH
   80 put n1'=n1
   81 normint
   -> (proven!)
66: second: {SubLess, apr:, second:}
   79 employ NormalForm(n1)
AFFIRM Type Specifications and Proofs

\[ \Rightarrow \]

NewSequenceOfBit:
  \text{Immediate}

\begin{align*}
66: & \quad \text{aor:} \\
67: & \quad \text{\& cases} \\
68: & \quad \text{\& invoke IH} \\
69: & \quad \text{\& put \texttt{n1}=\texttt{ss}ss} \\
70: & \quad \text{\& normint} \\
71: & \quad \text{\& replace} \\
72: & \quad \text{\& apply SameLastBit} \\
73: & \quad \text{\& put } n2 = \texttt{ss}ss' \\
74: & \quad \text{\& and } b = \texttt{11}ss' \\
75: & \quad \text{\& and } n1 = ss\texttt{ss} \quad \text{(choose)} \\
76: & \quad \text{\& replace} \\
\end{align*}
(proven!)

\text{theorem PktSubBound,}
\begin{align*}
\text{s1 subset s2} & \\
\text{and Bounded(\texttt{Nchanges(Seqnums(s1))},} & \\
\text{\texttt{Nchanges(Seqnums(s2))} + 1, 1)} & \\
\text{imp Bounded(\texttt{Nchanges(Seqnums(s1))},} & \\
\text{\texttt{Nchanges(Seqnums(s2))}, 1)} & \\
\text{and \texttt{Nchanges(Seqnums(s1))} = \texttt{Nchanges(Seqnums(s2))}} & \\
\text{PktSubBound uses SubSeqnum! and SubLess!}.
\end{align*}

\text{proof tree:}
52: \quad \text{PktSubBound} \\
53: \quad \text{\& apply SubSeqnum (proved by Divito using AFFIRM 120 on 12-Feb-81}} \\
54: \quad \text{\& in transcript <DIVITO>APFFLOG,12-FEB-81.1)} \\
55: \quad \text{\& put (s1'=s1) and (s2'=s2)} \\
56: \quad \text{\& invoke Bounded \& all \&} \\
57: \quad \text{\& normint} \\
58: \quad \text{\& apply SubLess} \\
59: \quad \text{\& put (n2 = Seqnums(s2)) and (n1 = Seqnums(s1)) \& (search)} \\
60: \quad \text{\& (proven!)}

\text{theorem SubToLag,}
\begin{align*}
\text{s1 subset s2} & \\
\text{and Repeats(s2)} & \\
\text{and Bounded(\texttt{Nchanges(Seqnums(s1))},} & \\
\text{\texttt{Nchanges(Seqnums(s2))}, 1)} & \\
\text{imp UniqueSeq(s1) iss UniqueSeq(s2)} & \\
\text{SubToLag uses LastSeq!, SameLastBit?, PktSubBound!, HiValued!, and UniqueEq!}.
\end{align*}

\text{proof tree:}
12: \quad \text{SubToLag} \\
13: \quad \text{\& employ Induction(s2) (proved by Divito using AFFIRM 120 on}} \\
14: \quad \text{\& 12-Feb-81 in transcript} \\
15: \quad \text{\& <DIVITO>APFFLOG,12-FEB-81.1)}
AFFIRM Type Specifications and Proofs

13: \textbf{NewSequenceOfPacket:}
14: \hspace{1em} 17 \textbf{replace}
15: \hspace{2em} (proven!)
16: \textbf{apr:}
17: \hspace{1em} 18 \textbf{cases}
18: \hspace{2em} 19 \textbf{split}
19: \hspace{3em} \textbf{first:}
20: \hspace{4em} 20 \textbf{replace}
21: \hspace{5em} (proven!)
22: \hspace{3em} \textbf{second:}
23: \hspace{4em} 21 \textbf{suppose} \texttt{LastBit}(\texttt{Segnums}(ss')) = \texttt{segno}(ii')
24: \hspace{5em} \textbf{yes:}
25: \hspace{6em} 22 \textbf{split}
26: \hspace{7em} 24 \textbf{invoke} \texttt{IH}
27: \hspace{8em} 26 \textbf{put} \texttt{s1'} = \texttt{s1} \textbf{(search)}
28: \hspace{9em} (proven!)
29: \hspace{2em} \textbf{second:}
30: \hspace{3em} 25 \textbf{employ} \texttt{NormalForm}(\texttt{s1})
31: \hspace{4em} \textbf{NewSequenceOfPacket:}
32: \hspace{5em} \texttt{Immediate}
33: \hspace{4em} \textbf{apr:}
34: \hspace{5em} 28 \textbf{cases}
35: \hspace{6em} 29 \textbf{suppose} \texttt{LastBit}(\texttt{Segnums}(ss'))
36: \hspace{7em} = \texttt{segno}(ii')
37: \hspace{5em} \textbf{yes:}
38: \hspace{6em} 30 \textbf{invoke} \texttt{IH}
39: \hspace{7em} 32 \textbf{put} \texttt{s1} = \texttt{ss'} \textbf{(search)}
40: \hspace{8em} (proven!)
41: \hspace{5em} \textbf{no:}
42: \hspace{6em} 31 \textbf{replace} \texttt{ii'}
43: \hspace{7em} 34 \textbf{apply} \texttt{LastSeq}
44: \hspace{8em} 35 \textbf{put} \texttt{s} = \texttt{ss}
45: \hspace{9em} 36 \textbf{split}
46: \hspace{5em} \textbf{first:}
47: \hspace{6em} 37 \textbf{replace} \texttt{ss}
48: \hspace{7em} 39 \textbf{replace} \texttt{ss'}
49: \hspace{8em} (proven!)
50: \hspace{5em} \textbf{second:}
51: \hspace{6em} 38 \textbf{replace} \texttt{segno}(\texttt{ii})
52: \hspace{7em} 40 \textbf{apply} \texttt{SameLastBit2}
53: \hspace{8em} 41 \textbf{put} \texttt{s2} = \texttt{ss}
54: \hspace{9em} \texttt{and} \texttt{s1} = \texttt{ss'}
55: \hspace{10em} \texttt{and} \texttt{p} = \texttt{ii} \textbf{(search)}
56: \hspace{5em} (proven!)
57: \hspace{4em} \textbf{no: (SubToLag, apr, second:)}
58: \hspace{5em} 23 \textbf{split}
59: \hspace{6em} \textbf{first:}
60: \hspace{7em} 43 \textbf{apply} \texttt{PktSubBound}
61: \hspace{8em} 45 \textbf{put} (\texttt{s1'} = \texttt{s1}) \textbf{and} (\texttt{s2} = \texttt{ss'})
62: \hspace{9em} 46 \textbf{invoke} \texttt{IH}
63: \hspace{10em} 47 \textbf{put} \texttt{s1'} = \texttt{s1} \textbf{(search)}
64: \hspace{11em} (proven!)
AFFIRM Type Specifications and Proofs

37: second:(SubToLoq, apr:, second:, not) employ NormalForm(s1)
   NewSequenceOfPacket:
   Immediate
37: apr:
38: cases 50 suppose LastBit(Seqnums(ss'))
   = seqno(ii)
39: yes:
40: apply PktSubBound
41: put (s1=ss') and (s2=ss)
42: invoke IH (proven!)
42: not:(SubToLoq, apr:, second:, not, second:, apr:)
43: replace
44: apply Bivalue
45: put b = seqno(ii)
46: and b1 = LastBit(Seqnums(ss'))
47: and b2 = LastBit(Seqnums(ss))
48: put s1 = ss' (search)
49: (proven!)

Type CONTINUE to return to AFFIRM.

192 U: print uses:

Theorem  Uses: Used by:
LastSeq, proved - SubToLoq SubToLoq
SameLastBit2, proved - UniqueEq UniqueEq
BoundedBy1, proved - UnEq UniqueEq
NCUnique, proved - IssLenEq IssLenEq
LenNonneg, assumed - IssLenEq IssLenEq
IntFact1, proved - IssLenEq IssLenEq
IntFact2, proved - IssLenEq IssLenEq
IssLenEq, proved IntFact2 IssLenEq
IssLenEq, proved LenNonneg UniqueEq
          IssLenEq IssLenEq
          IntFact1

Bivalue, proved - LastEq SubToLoq SubToLoq
BitSubBound, proved - UniqueEq UniqueEq
LastEq, proved SubLess SameLastBit
BitSubBound Bivalue
LastEq, proved BoundedBy1
NCUnique
IssLenEq
SubSeqnum
LastEq
SubSeqnum, proved - UniqueEq
<table>
<thead>
<tr>
<th>Type Spec</th>
<th>Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>NonNoneq, proved</td>
<td>-</td>
</tr>
<tr>
<td>SameLastBit, proved</td>
<td>-</td>
</tr>
<tr>
<td>Subless, proved</td>
<td>NonNoneq, SameLastBit</td>
</tr>
<tr>
<td>PktSubBound, proved</td>
<td>SubSeqnum, Subless</td>
</tr>
<tr>
<td>SubToLag, proved</td>
<td>LastSeq, SameLastBit</td>
</tr>
</tbody>
</table>

PktSubBound, Subless, LastEq, SubLag, HitSubBound, PktSubBound, SubToLag, -

193 U: print assumptions;
LenNonneg is used by IsSizeEq.
assume LenNonneg, Length(m) >= 0;

194 U: quit;
Automatically summarize the proof attempts? no [confirm]
Type CONTINUE to return to AFFIRM.