A

LOTOS definitions

notation	meaning
L	set of observable actions
$\mathcal{L}$	$L \cup \{\tau = \text{the internal action}\}$
L^*	set of strings of observable actions, also called a *trace*
ϵ	the empty string: $P \stackrel{\varepsilon}{\Rightarrow} Q \text{ifandonlyif} P \stackrel{\tau^k}{\rightarrow} \text{forsome} k \geq 0$
$\mathcal{A}ct(X)$	the set of actions belonging to a process X
τ^k	a sequence of $k (k > 0)$ **i**-actions
s	a string of observable actions
$B \stackrel{s}{\rightarrow} B'$	$\exists B_i, 0 \leq i \leq n : B = B_0 \stackrel{\mu_1}{\rightarrow} B_1, B_1 \stackrel{\mu_2}{\rightarrow} B_2, ..., B_{n-1} \stackrel{\mu_n}{\rightarrow} B_n = B'$
	$(\text{shorthand}) B \stackrel{\mu_1}{\rightarrow}\stackrel{\mu_2}{\rightarrow} ... \stackrel{\mu_n}{\rightarrow} B'$
$B \stackrel{s}{\Rightarrow} B'$	$\exists s = \mu_1\mu_2...\mu_n \mid B \stackrel{\tau^{k_0}}{\rightarrow}\stackrel{\mu_1}{\rightarrow}\stackrel{\tau^{k_1}}{\rightarrow}\stackrel{\mu_2}{\rightarrow} \cdots \stackrel{\mu_n}{\rightarrow}\stackrel{\tau^{k_n}}{\rightarrow} B'$
$B \stackrel{s}{\Rightarrow}$	$\exists B' : B \stackrel{s}{\Rightarrow} B'$
$out(B)$	$\{a \in L \mid B \stackrel{a}{\Rightarrow}\}$
$out_\sigma(B)$	$\bigcup_{B \stackrel{\sigma}{\Rightarrow} B'} out(B')$ where $\sigma \in Tr(B)$

Figure A.1: Notation for LOTOS and its LTS

name	syntax	axioms/inference rules
inaction	`stop`	
action prefix		
$-$*observable*	$a; B$	$a; B \stackrel{a}{\rightarrow} B$
$-$*unobservable*(*internal*)	$\mathbf{i}; B$	$\mathbf{i}; B \stackrel{\tau}{\rightarrow} B$
choice	$B_1 [] B_2$	$\frac{B_1 \stackrel{a}{\rightarrow} B_1'}{B_1 [] B_2 \stackrel{a}{\rightarrow} B_1'}, \frac{B_1 \stackrel{a}{\rightarrow} B_1'}{B_1 [] B_2 \stackrel{a}{\rightarrow} B_2'}$
generalised choice	$\Sigma\mathcal{B}$	$\frac{B \stackrel{a}{\rightarrow} B', B \in \mathcal{B}}{\Sigma\mathcal{B} \stackrel{a}{\rightarrow} B'}$
composition	$B_1 \| B_2$	$\frac{B_1 \stackrel{\tau}{\rightarrow} B_1'}{B_1 \| B_2 \stackrel{\tau}{\rightarrow} B_1' \| B_2}, \frac{B_2 \stackrel{\tau}{\rightarrow} B_2'}{B_1 \| B_2 \stackrel{\tau}{\rightarrow} B_1 \| B_2'}, \frac{B_1 \stackrel{a}{\rightarrow} B_1', B_2 \stackrel{a}{\rightarrow} B_2', a \neq \tau}{B_1 \| B_2 \stackrel{a}{\rightarrow} B_1' \| B_2'}$

Figure A.2: LOTOS Transition Rules

B

Guide Words for Flexport

PROVISION

- OMISSON
  - Total
  - Partial
- COMMISSION (extra data)
- CORRUPTION

TIMING

- RELATIVE
  - Early
  - Late
- ABSOLUTE
  - Deadline missed
  - Timeout
- RATE
  - Too Slow
  - Too Fast
- DURATION

- Too Short
- Too Long

C

Tool Summary

As part of the CM Plan, we detail the platform plus tools and their versions which were used for the development.

```
HARDWARE: SUN 670MP
O/S     : UNIX, SunOS Release 4.1.3.,
          Xwindows X11/R5

SPECIFICATION TOOLS:

LITE3.0
 - TOPO :    3R6
 - SMILE:    2.2
 - LOLA :

CADP  BETA-VERSION Z-r (Wed Jun 12 16:29:49 MET DST 1996)

%src: /research/fm/cadp/VERSION

aldebaran       :       6.0
caesar          :       5.0
caesar.adt      :       4.4
open/caesar     :       1.8
bcg             :       1.0
```

```
CWB             :       7.0

CM TOOLS:

RCS         Revision Number: 5.6
Gnu Make    version          3.74
m4    (see O/S)
```

D

LOTOS specifications

D.1 Flexport Intensional Specification

The following is a listing of the Flexport Intensional specification with Baud Rate Hunting included.

```
(*********************************************)
(*                                           *)
(*  LOTOS intensional specification for      *)
(*  Flexport Protocol Lower Level Link Layer *)
(*                                           *)
(*********************************************)

(* $History$ *)
(* $Log: flexint_t.lot,v $
# Revision 2.1  1997/04/23  11:56:31  cs_s447
# This version supports Baud rate hunting
# with two extra gates 'baud' and 'tick'
#
# Revision 1.2  1997/04/23  11:53:40  cs_s447
# library call for data types omitted
#
# Revision 1.1  1997/04/23  11:51:34  cs_s447
# Initial revision
#   *)

specification Flexport [h1,h2,l1,l2,baud,tick] : noexit

(*****************************************)
(** Data Types Definition for Flexport     **)
(*****************************************)

(* $History$ *)
(* $Log: flexdata.lotdat,v $
# Revision 2.1  1997/04/16  10:18:05  cs_s447
# Data types added to support Baud Rate Hunting
```

```
#
# Revision 1.3  1997/04/09  13:55:57  cs_s447
# Queue Data type added to support n-place buffer
#
# Revision 1.2  1997/04/08  15:01:23  cs_s447
# Compatibility for CADP - a replacement file which henceforth
# is to contain all the data type definitions...
#
# 1. Boolean, Natural    Data Types added
#
# 2. Comments included to be read by caesar.adt
# 3  TYPE DataItem replaced to compensate for C's lack of
# support for overloading ..... New sorts: PDU_data and Packet_Data
# defined.
#
# Revision 1.1  1997/04/04  14:28:57  cs_s447
# Initial revision
#     *)

type Boolean is
   sorts
      Bool (*! implementedby ADT_BOOL comparedby ADT_CMP_BOOL
               enumeratedby ADT_ENUM_BOOL printedby ADT_PRINT_BOOL *)
   opns
      false     (*! implementedby ADT_FALSE constructor *),
      true      (*! implementedby ADT_TRUE constructor *) : -> Bool
      not       (*! implementedby ADT_NOT *) : Bool -> Bool
      _and_     (*! implementedby ADT_AND *),
      _or_      (*! implementedby ADT_OR *),
      _xor_     (*! implementedby ADT_XOR *),
      _implies_ (*! implementedby ADT_IMPLIES *),
      _iff_     (*! implementedby ADT_IFF *),
      _eq_      (*! implementedby ADT_EQ_BOOL *),
      _ne_      (*! implementedby ADT_NE_BOOL *) : Bool, Bool  -> Bool
   eqns
      forall x, y : Bool
      ofsort Bool
         not (true)   =  false;
         not (false)  =  true;
         x and true   =  x;
         x and false  =  false;
         x or true    =  true;
         x or false   =  x;
         x xor y      =  (x and not (y)) or (y and not (x));
         x implies y  =  y or not (x);
         x iff y      =  (x implies y) and (y implies x);
         x eq y       =  x iff y;
         x ne y       =  x xor y;
```

```
endtype (* Boolean *)


type Natural is Boolean
   sorts Nat (*! implementedby ADT_NAT comparedby ADT_CMP_NAT
                 enumeratedby ADT_ENUM_NAT printedby ADT_PRINT_NAT *)
   opns 0 (*! implementedby ADT_N0 constructor *),
        1 (*! implementedby ADT_N1 *),
        2 (*! implementedby ADT_N2 *),
        3 (*! implementedby ADT_N3 *),
        4 (*! implementedby ADT_N4 *),
        5 (*! implementedby ADT_N5 *),
        6 (*! implementedby ADT_N6 *),
        7 (*! implementedby ADT_N7 *),
        8 (*! implementedby ADT_N8 *),
        9 (*! implementedby ADT_N9 *) : -> Nat
        Succ (*! implementedby ADT_SUCC constructor *) : Nat -> Nat
        _+_  (*! implementedby ADT_PLUS *),
        _*_  (*! implementedby ADT_MULT *),
        _**_ (*! implementedby ADT_POWER *),
        _-_  (*! implementedby ADT_MINUS *),
        _div_ (*! implementedby ADT_DIV *),
_mod_ (*! implementedby ADT_MOD *) : Nat, Nat -> Nat
        _eq_ (*! implementedby ADT_EQ_NAT *),
        _ne_ (*! implementedby ADT_NE_NAT *),
        _lt_ (*! implementedby ADT_LT_NAT *),
        _le_ (*! implementedby ADT_LE_NAT *),
        _gt_ (*! implementedby ADT_GT_NAT *),
        _ge_ (*! implementedby ADT_GE_NAT *),
_==_ (*! implementedby ADT_EQ_BIS_NAT *),
        _<>_ (*! implementedby ADT_NE_BIS_NAT *),
_<_ (*! implementedby ADT_LT_BIS_NAT *),
_<=_ (*! implementedby ADT_LE_BIS_NAT *),
_>_ (*! implementedby ADT_GT_BIS_NAT *),
_>=_ (*! implementedby ADT_GE_BIS_NAT *) : Nat, Nat -> Bool
        min (*! implementedby ADT_MIN *),
        max (*! implementedby ADT_MAX *),
        gcd (*! implementedby ADT_GCD *),
        scm (*! implementedby ADT_SCM *) : Nat, Nat -> Nat
   eqns
      forall m, n : Nat
      ofsort Nat
         1 = Succ (0);
         2 = Succ (1);
         3 = Succ (2);
         4 = Succ (3);
         5 = Succ (4);
         6 = Succ (5);
```

```
   7 = Succ (6);
   8 = Succ (7);
   9 = Succ (8);
ofsort Nat
   m + 0 = m;
   m + Succ(n) = Succ(m) + n;
ofsort Nat
   m * 0 = 0;
   m * Succ(n) = m + (m * n);
ofsort Nat
   m ** 0 = Succ(0);
   m ** Succ(n) = m * (m ** n)
ofsort Nat
   m - 0 = m;
   Succ (m) - Succ (n) = m - n;
ofsort Nat
   n ne 0, m lt n => m div n = 0;
   n ne 0, m ge n => m div n = 1 + ((m - n) div n);
ofsort Nat
   n ne 0, m lt n => m mod n = m;
   n ne 0, m ge n => m mod n = ((m - n) mod n);
ofsort Bool
   0 eq 0                   = true;
   0 eq Succ (n)            = false;
   Succ (m) eq 0            = false;
   Succ (m) eq Succ (n)     = m eq n;
ofsort Bool
   m ne n                   = not (m eq n);
ofsort Bool
   0 lt 0                   = false;
   0 lt Succ (n)            = true;
   Succ (n) lt 0            = false;
   Succ (m) lt Succ (n)     = m lt n;
ofsort Bool
   m le n                   = (m lt n) or (m eq n);
ofsort Bool
   m ge n                   = not (m lt n);
ofsort Bool
   m gt n                   = not (m le n);
ofsort Bool
   m == n                    = m eq n;
   m <> n                    = m ne n;
   m <= n                    = m le n;
   m < n                     = m lt n;
   m > n                     = m gt n;
   m >= n                    = m ge n;
ofsort Nat
   m le n => min (m, n) = m;
```

```
      m gt n => min (m, n) = n;
    ofsort Nat
      m ge n => max (m, n) = m;
      m lt n => max (m, n) = n;
    ofsort Nat
      m eq n, m ne 0 => gcd (m, n) = m;
      m lt n, m ne 0 => gcd (m, n) = gcd (m, n - m);
      m gt n, n ne 0 => gcd (m, n) = gcd (m - n, n);
    ofsort Nat
      scm (m, n) = (m * n) div gcd (m, n);

endtype (* NATURAL *)


type PDUnits is BOOLEAN, NATURAL

  sorts PDU (*! implementedby PDU comparedby CMP_PDU
              enumeratedby ENUM_PDU printedby PRINT_PDU *),
       pdu_data (*! implementedby PDU_DATA comparedby CMP_PDU_DATA
              enumeratedby ENUM_PDU_DATA printedby PRINT_PDU_DATA *)

  opns

    (* 'Raw' Protocol Data Units primitives *)

       ACK       (*! implementedby PDU_ACK constructor   *),
        CR       (*! implementedby PDU_CR constructor    *),
       DLE       (*! implementedby PDU_DLE constructor   *),
       ENQ       (*! implementedby PDU_ENQ constructor   *),
       NAK       (*! implementedby PDU_NAK constructor   *),
      X_ON       (*! implementedby PDU_X_ON constructor  *),
     X_OFF       (*! implementedby PDU_X_OFF constructor *),

    (* 'Refined' PDUs *)
      DAT         (*! implementedby PDU_DAT constructor  *),
      LC_P        (*! implementedby PDU_LC_P constructor *)

                  : -> PDU

     _ eq _     (*! implementedby PDU_EQ *),
     _ ne _     (*! implementedby PDU_NE *)

                  :  PDU,PDU -> Bool

      NumPDU   (*! implementedby  PDU_NUM *):   PDU  -> Nat
      send     (*! implementedby PDU_DATA_SEND constructor  *),
      receive  (*! implementedby PDU_DATA_RECEIVE constructor *)
```

```
                  :     PDU -> PDU_Data


      eqns
        forall x,y: PDU
          ofsort Nat
        NumPDU (ACK)   = 0;
        NumPDU (CR)    = Succ(NumPDU (ACK));
        NumPDU (DLE)   = Succ(NumPDU (CR));
        NumPDU (ENQ)   = Succ(NumPDU (DLE));
        NumPDU (NAK)   = Succ(NumPDU (ENQ));
        NumPDU (X_ON)  = Succ(NumPDU (NAK));
        NumPDU (X_OFF) = Succ(NumPDU (X_ON));

          ofsort Bool
             x eq y    = NumPDU (x) eq NumPDU (y);
             x ne y    = NumPDU (x) ne NumPDU (y);


endtype

type Packet is Boolean, Natural

  sorts pkt (*! implementedby PKT comparedby CMP_PKT
               enumeratedby ENUM_PKT printedby PRINT_PKT *),
       pkt_data (*! implementedby PKT_DATA comparedby CMP_PKT_DATA
               enumeratedby ENUM_PKT_DATA printedby PRINT_PKT_DATA *)

  opns

     LC_packet (*! implementedby PKT_LC_PACKET constructor *):  -> Pkt
      _ eq _     (*! implementedby PKT_EQ *),
      _ ne _     (*! implementedby PKT_NE *) : PKT,PKT -> Bool

       NumPKT   (*! implementedby  PKT_NUM *):   PKT  -> Nat
     send       (*! implementedby PKT_DATA_SEND constructor   *),
     receive    (*! implementedby PKT_DATA_RECEIVE constructor *)

                   :  Pkt -> pkt_Data


  eqns
   forall x,y: pkt
    ofsort Nat
  NumPKT (LC_Packet) = 0;

    ofsort Bool
             x eq y    = NumPKT (x) eq NumPKT (y);
```

```
            x ne y    = NumPKT (x) ne NumPKT (y);

endtype (* Packet *)

type Queue is Boolean, PDUnits


  Sorts Queue (*! implementedby QUEUE comparedby CMP_QUEUE
               enumeratedby ENUM_QUEUE printedby PRINT_QUEUE *)

  OPNS
    head (*! implementedby QUEUE_HEAD             *):    Queue -> PDU
    tail (*! implementedby QUEUE_TAIL             *):    Queue -> Queue
    _ + _  (*! implementedby QUEUE_ADD constructor *)
                                                  : Queue, PDU -> Queue
    empty  (*! implementedby QUEUE_ISEMPTY constructor *):     -> Queue
    length (*! implementedby QUEUE_LENGTH             *):   Queue -> Nat
    maxlength (* implementedby QUEUE_MAXLENGTH constructor*):     -> Nat

  EQNS

  forall s: Queue, x,y: PDU

  OFSORT PDU

   head (empty + x) = x;
   head (s+ x+ y) = head(s+x)

  OFSORT Queue

   tail (empty + x) = empty;
   tail (s+x+y) = tail(s+x) + y;

  OFSORT Nat

   maxlength                 = succ(0);
  (* constant for maximum capacity of channel *)

   length (empty) = 0;
   length (s + x) = succ(length (s))

endtype (* Queue *)


type BaudDataRate is Boolean, Natural

  sorts BaudRate (*! implementedby BAUDRATE comparedby CMP_BAUDRATE
               enumeratedby ENUM_BAUDRATE printedby PRINT_BAUDRATE *)
```

```
  opns

   1200  (*! implementedby BAUDRATE_1200 constructor *) :   -> BaudRate
   2400  (*! implementedby BAUDRATE_2400 constructor *) :   -> BaudRate
   4800  (*! implementedby BAUDRATE_4800 constructor *) :   -> BaudRate
   9600  (*! implementedby BAUDRATE_9600 constructor *) :   -> BaudRate
   next          (*! implementedby BAUDRATE_NEXT     *)
                                                 : BaudRate -> BaudRate
   _ eqbaud _    (*! implementedby BAUDRATE_EQBAUD   *)
                                        : BaudRate,BaudRate -> Bool
   _ neqbaud _   (*! implementedby BAUDRATE_NEQBAUD  *)
                                        : BaudRate,BaudRate -> Bool
   BaudNum (*! implementedby BAUDRATE_BAUDNUM        *)
                                                 : BaudRate -> Nat

  eqns

   forall x,y: BaudRate

     ofsort BaudRate

   next(1200)  = 9600;
   next(2400)  = 1200;
   next(4800)  = 2400;
   next(9600)  = 4800;

     ofsort Nat

   BaudNum(1200) = 0;
   BaudNum(2400) = succ(BaudNum(1200));
   BaudNum(4800) = succ(BaudNum(2400));
   BaudNum(9600) = succ(BaudNum(4800))

    ofsort Bool

     x EqBaud y  = BaudNum(x) eq BaudNum(y);
     x NeqBaud y = BaudNum(x) ne BaudNum(y)

endtype (* BaudDataRate *)
```

```
behaviour

 (
  (  Station1[h1,l1,baud,tick]  |||  Station2[h2,l2] )
   |[h1,h2]|
  (  Duplex_chan[h1,h2] )
 )

where

(*******************************************)
(*  Duplex Channel Specification           *)
(*******************************************)

(* $History$ *)
(* $Log: buffer.lotsrc,v $
# Revision 2.1  1997/04/09  13:47:07  cs_s447
# buffer generalised to n-place, where
# n is given by constant type 'maxlength'
#
# Revision 1.1  1997/04/04  14:25:24  cs_s447
# Initial revision
# *)

process Duplex_chan [h1, h2]: noexit :=

  Simplex[h1,h2] ||| Simplex[h2,h1]

  where

  process Simplex[a,b] : noexit :=
    Buff0[a,b](maxlength)

(* set this constant in the Queue ADT in FLEXPORT.lib *)

  where

    process Buff0[get,put] (max: Nat) : noexit :=

    (* caters for parametrised sends/receives ... *)

    choice x:PDU []
      get!send(x); Buff[get,put](empty + x, succ(0), max)

    endproc (* Buff0 *)

    process Buff[get,put] (q: Queue, leng, max:Nat) : noexit :=
```

```
      (put!receive(head(q)) [leng eq succ(0)]; Buff0[get,put](max)
      )
     []
      (put!receive(head(q)) [leng gt succ(0)]; Buff[get,put]
                                      (tail(q), length(tail(q)),max)
      )
     []
      (choice x:PDU []
       get!send(x) [leng lt max]; Buff[get,put]
                                     ((q + x), succ(length(q)), max)
      )

    endproc (* Buff *)

  endproc (* Simplex *)

endproc (* Duplex_chan *)


(*****************************************)
(*  Flexport Station 1                   *)
(*****************************************)

(* $History$ *)
(* $Log: station1.lotsrc,v $
# Revision 2.1  1997/04/23  14:06:10  cs_s447
# Baud Rate Hunting added
#
# Revision 1.3  1997/04/23  13:56:00  cs_s447
# now allows for repeated receipt of data until DLE is
#
# Revision 1.2  1997/04/23  13:48:04  cs_s447
#  restructuring of the sequences of processes:
# S1_connect is now simply S1_sendENQ and
# each of S1_sendENQ and S1_TestXON have their exits replaced
# a call to the next process
#     *)

process Station1[h1,l1,b,t] : noexit :=

      (S1_Connect[h1,l1,b,t] >> S1_DataTransmit[h1,l1])

where

  process S1_connect[h1,l1,b,t] : exit :=

    S1_sendENQ[h1,l1,b,t]
```

```
   where

    process S1_sendENQ[h1,l1,b,t] : exit :=

(* this is the initial process for station 1 *)

 S1_BaudRateHunting[h1,l1,b,t](9600,0)

    where

   process S1_BaudRateHunting[h1,l1,b,t](rate:BaudRate,count:Nat)
                                                          : exit :=

     b!rate;h1!send(ENQ); TestDLE[h1,l1,b,t](rate,count)

       where

     process TestDLE[h1,l1,b,t](rate:Baudrate,count:Nat) : exit :=

       (
        t (* 1 second elapses *);

        ([count eq succ(succ(0))] -> TryConnect[h1,l1,b,t](next(rate),0)
        []
         [count ne succ(succ(0))] -> TryConnect[h1,l1,b,t]
                                                 (rate,succ(count))
        )
       )
      []
       (
        choice tx:PDU []

          h1!receive(tx); ([tx eq DLE] -> h1!send(X_ON)
                                            ; S1_TestX_ON[h1,l1,b,t]
                          []
                           [tx ne DLE] -> TestDLE[h1,l1,b,t](rate,count)
                          )
       )
       endproc (* TestDLE *)

      process TryConnect[h1,l1,b,t](rate:Baudrate,count:Nat) : exit :=

       b!rate;
        (
         h1!send(ENQ); TestDLE[h1,l1,b,t](rate,count)
        []
         (choice x3:PDU []
          h1!receive(x3); h1!send(ENQ); TestDLE[h1,l1,b,t](rate,count))
```

```
      )

    endproc(* TryConnect *)

  endproc (* S1_BaudRateHunting *)

  endproc (* S1_sendENQ *)

  process S1_TestX_ON[h1,l1,b,t] : exit :=

  (* Now test for X_ON transmission from S2 *)
  (* using IF ... THEN .. ELSE construct    *)

    (choice x:PDU []
       h1!receive(x); ([x eq X_ON] -> S1_SendLCPack[h1,l1]
                       []
                       [x ne X_ON] -> S1_connect[h1,l1,b,t]
                      )
    )

  endproc (* S1_TestX_ON *)


  process S1_SendLCPack[h1,l1] : exit :=

  (* Send a link configuration packet *)

    l1!send(LC_packet);  (* Upper level sends Link Config. Packet *)
                         (*  - service request                    *)
    h1!send(LC_P);       (* Lower level sends corresponding PDUs  *)
    h1!receive(ACK);     (* Lower level receives acknowledgement  *)
    h1!receive(LC_P);     (* Lower level receives PDU sequence    *)
    l1!receive(LC_packet);(* .. which is recomposed into a Link   *)
                          (* configuration packet                 *)
    h1!send(ACK);         (* Acknowledgement sent by lower level  *)
                          (* link                                 *)
    exit                  (* Connection phase over for station 1  *)

  endproc (* S1_SendLCPack *)

endproc (* S1_connect *)

process S1_DataTransmit[h1,l1] : noexit :=

(* connection established - assume continuous transmission *)
(* modelled as an endless sequence of token PDUs           *)

   h1!send(DAT); S1_DataTransmit[h1,l1]
```

```
   []
     h1!receive(DAT); S1_DataTransmit[h1,l1]

  endproc (* S1_DataTransmit *)

endproc (* Station1 *)
(*****************************************)
(*  Flexport Station 2                   *)
(*****************************************)

(* $History$ *)
(* $Log: station2.lotsrc,v $
# Revision 1.4  1997/04/15  10:22:34  cs_s447
#  now allows for repeated receipt of data until ENQ is
# received; also allows for S1's X_ON to be received
# between sending its DLE and X_ON.
#
# Revision 1.3  1997/04/07  17:12:07  cs_s447
# simple bug fixes
#
# Revision 1.2  1997/04/07  16:37:42  cs_s447
# restructuring of the sequences of processes:
#
# S2_Connect is now simply S2_AwaitENQ and
# each of S2_AwaitENQ and S2_TestX_ON have their exits replaced
# by '>>' and a call to the next process
#
# REASON: caesar could not cope with recursive instantiation in
# a process to the left of '>>' operator
#
# Revision 1.1  1997/04/04  14:28:35  cs_s447
# Initial revision
#     *)

process Station2[h2,l2] : noexit :=

      (S2_Connect[h2,l2] >> S2_DataTransmit[h2,l2])

where

  process S2_connect[h2,l2] : exit :=

    S2_AwaitENQ[h2,l2]

(* this is the initial process to be instantiated *)

  where
```

```
process S2_AwaitENQ[h2,l2] : exit :=

   (choice y1:PDU []

      h2!receive(y1); ([y1 eq ENQ] -> h2!send(DLE)
                                        ; S2_TestX_ON[h2,l2]
                      []
                       [y1 ne ENQ] -> S2_AwaitENQ[h2,l2]
                      )
   )

endproc (* S2_AwaitENQ *)


process S2_TestX_ON [h2,l2] : exit :=

  (choice y2:PDU []
  h2!receive(y2); h2!send(X_ON) ;
                  (
                   [y2 eq X_ON] -> S2_SendLCPack[h2,l2]
                  []
                   [y2 ne X_ON] -> S2_Connect [h2,l2]
                   )
  )
[]
  h2!send(X_ON);
  (choice y3:PDU []
       h2!receive(y3);
                  (
                   [y3 eq X_ON] -> S2_SendLCPack[h2,l2]
                  []
                   [y3 ne X_ON] -> S2_Connect [h2,l2]
                   )
  )

endproc (* S2_TestX_ON *)


process S2_SendLCPack[h2,l2] : exit :=

  h2!receive(LC_P);
  l2!receive(LC_packet);
  h2!send(ACK);
  l2!send(LC_Packet);
  h2!send(LC_P);
  h2!receive(ACK);
  exit               (* Connection phase over for station 2  *)
```

```
    endproc (* S2_SendLCPack *)

  endproc (* S2_connect *)

  process S2_DataTransmit[h2,l2]: noexit :=

  (* connection established - assume continuous transmission *)
  (* modelled as an endless sequence of token PDUs           *)

    h2!send(DAT); S2_DataTransmit[h2,l2]
   []
    h2!receive(DAT);S2_DataTransmit[h2,l2]

  endproc (* S2_DataTransmit *)

endproc (* Station 2 *)


endspec
```

D.2 Unified Tester

The unified tester for the MSC-based specification is given below.

```
(*********************************************)
(*                                           *)
(*  LOTOS Tester for Reduction relation      *)
(*  with respect to Intensional spec for     *)
(* Flexport Protocol Lower Level Link Layer  *)
(*                                           *)
(*    Requires: v1.2 or higher               *)
(*              of Flexdata.lotdat           *)
(*********************************************)

(* $History$ *)
(* $Log: testmsc_t.lot,v $
# Revision 1.1  1997/04/24  14:09:06  cs_s447
# Initial revision
#     *)

specification FlexportTestMSC [h1,h2,l1,l2,successMSC,
                 data_failure,gate_failure,twofold_failure] : noexit

library

  Boolean, Set

endlib

(*******************************************)
(** Data Types Definition for Flexport     **)
(*******************************************)

(* $History$ *)
(* $Log: flexdata.lotdat,v $
# Revision 1.2.1.2  1997/04/24  14:24:16  cs_s447
# Queue Data Type added to support n-place buffer
#
# Revision 1.2.1.1  1997/04/24  14:04:12  cs_s447
# Variant for the unified tester:
#  * Boolean type omitted
#  * Sort 'Nat' renamed 'Natl' to avoid clashes in library
#    calls for other types
#  * New type 'DataItem' with complete equations for sends and
#    receives w.r.t to eq and ne operators
#  * New type 'Dataset' to support TestEvent processes
#
# Revision 1.2  1997/04/08  15:01:23  cs_s447
```

```
# Compatibility for CADP - a replacement file which henceforth
# is to contain all the data type definitions...
#
# 1. Boolean, Natural    Data Types added
#
# 2. Comments included to be read by caesar.adt
# 3  TYPE DataItem replaced to compensate for C's lack of
# support for overloading ..... New sorts: PDU_data and Packet_Data
# defined.
#
# Revision 1.1  1997/04/04  14:28:57  cs_s447
# Initial revision
#     *)


type Natural is Boolean
   sorts Natl (*! implementedby ADT_NAT comparedby ADT_CMP_NAT
                  enumeratedby ADT_ENUM_NAT printedby ADT_PRINT_NAT *)

< .... >

endtype (* Natural *)

type PDUnits is BOOLEAN, NATURAL

< .... >

endtype (* PDUnites *)

type Packet is Boolean, Natural

< .... >

endtype (* Packet *)

type DataItem is PDUnits, Packet

  sorts Data

  opns

  send, receive:   Pkt -> Data
  send, receive:   PDU -> Data
       _ eq _ ,
       _ ne _ ,
       _ lt _  :   Data,Data -> Bool

  eqns
```

```
  (* there should be 2^5 of these *)

   forall x,y: PDU, w,z: PKT
    ofsort Bool
  send(x) eq send(y)       = x eq y;
  receive(x) eq receive(y) = x eq y;
  send(x) ne send(y)       = x ne y;
  receive(x) ne receive(y) = x ne y;
  send(x) lt send(y)       = x lt y;
  receive(x) lt receive(y) = x lt y;

  send(w) eq send(z)       = w eq z;
  receive(w) eq receive(z) = w eq z;
  send(w) ne send(z)       = w ne z;
  receive(w) ne receive(z) = w ne z;
  send(w) lt send(z)       = w lt z;
  receive(w) lt receive(z) = w lt z;

(* Data Mismatches defined for testing purposes ... *)
(* 1. Comparisons of like param data types *)

  send(x) ne receive(y)   = true;
  send(x) eq receive(y)   = false;
(* undefined for 'lt' *)

  send(w) ne receive(z)   = true;
  send(w) eq receive(z)   = false;
(* undefined for 'lt' *)

  receive(x) ne send(y)   = true;
  receive(x) eq send(y)   = false;
(* undefined for 'lt' *)

  receive(w) ne send(z)   = true;
  receive(w) eq send(z)   = false;
(* undefined for 'lt' *)

(* 2. Comparisons with different param data types *)

  send(x) eq send(w)       = false;
  send(x) eq receive(w)    = false;
  receive(x) eq send(w)    = false;
  receive(x) eq receive(w) = false;

  send(w) eq send(x)       = false;
  send(w) eq receive(x)    = false;
  receive(w) eq send(x)    = false;
```

```
  receive(w) eq receive(x) = false;


  send(x) ne send(w)       = true;
  send(x) ne receive(w)    = true;
  receive(x) ne send(w)    = true;
  receive(x) ne receive(w) = true;

  send(w) ne send(x)       = true;
  send(w) ne receive(x)    = true;
  receive(w) ne send(x)    = true;
  receive(w) ne receive(x) = true

(* undefined for 'lt' *)

endtype (* DataItem *)

Type Dataset is Set actualizedby DataItem, Boolean using

(* note that the library type 'set' requires the definition *)
(* of operations 'eq','ne','lt'                             *)

sortnames

  Pset for Set
  Data for Element
  Bool for Fbool

endtype (* Dataset *)

type Queue is Boolean, PDUnits

  Sorts Queue

  OPNS

    head: (* extract first element of a queue *) Queue -> PDU
    tail: (* return all but head of a queue *)   Queue -> Queue
    _ + _       : Queue, PDU -> Queue
    empty :                  -> Queue
    length :      Queue      -> Natl

  EQNS

   forall s: Queue, x,y: PDU

  OFSORT PDU
```

```
   head (empty + x) = x;
   head (s+ x+ y) = head(s+x)

  OFSORT Queue

   tail (empty + x) = empty;
   tail (s+x+y) = tail(s+x) + y

  OFSORT Natl

   length (empty) = 0;
   length (s + x) = succ(length (s))

endtype (* Queue *)

behaviour

UnifiedTesterMSCBase[h1,h2,l1,l2,successMSC,
                     data_failure,gate_failure,twofold_failure]
|[h1,h2,l1,l2]|
Flexint[h1,h2,l1,l2]

where

  process Flexint[h1,h2,l1,l2] : noexit :=

   (
    (  Station1[h1,l1]  |||  Station2[h2,l2] )
     |[h1,h2]|
    (  Duplex_chan[h1,h2] )
   )

  where

  (*********************************************)
(*  Duplex Channel Specification              *)
(*********************************************)

(* $History$ *)
(* $Log: buffer.lotsrc,v $
# Revision 2.1.1.1  1997/04/24  15:00:06  cs_s447
# Variant for test purposes ..
#  doesn't use maxlength constant data type, but
# a magic number (5) instead.
#
# Revision 2.1  1997/04/09  13:47:07  cs_s447
# buffer generalised to n-place, where
# n is given by constant type 'maxlength'
```

```
#
# Revision 1.1  1997/04/04  14:25:24  cs_s447
# Initial revision
# *)

process Duplex_chan [h1, h2]: noexit :=

  Simplex[h1,h2] ||| Simplex[h2,h1]

  where

  process Simplex[a,b] : noexit :=
    Buff0[a,b](5)

(* change this constant to any number as required *)

  where

    process Buff0[get,put] (max: Natl) : noexit :=

    (* caters for parametrised sends/receives ... *)

    choice x:PDU []
      get!send(x); Buff[get,put](empty + x, 1, max)

    endproc (* Buff0 *)

    process Buff[get,put] (q: Queue, leng, max:Natl) : noexit :=

      (put!receive(head(q)) [leng eq 1]; Buff0[get,put](max)
      )
     []
      (put!receive(head(q)) [leng gt 1]; Buff[get,put]
                                    (tail(q), length(tail(q)),max)
      )
     []
      (choice x:PDU []
       get!send(x) [leng lt max]; Buff[get,put]
                                   ((q + x), succ(length(q)), max)
      )

    endproc (* Buff *)

  endproc (* Simplex *)

endproc (* Duplex_chan *)
```

```
  (*****************************************)
(*  Flexport Station 1                     *)
(*****************************************)

(* $History$ *)
(* $Log: station1.lotsrc,v $
# Revision 1.1  1997/04/23  13:43:14  cs_s447
# Initial revision
#     *)

process Station1[h1,l1] : noexit :=

< .... >

endproc (* Station1 *)

(*****************************************)
(*  Flexport Station 2                     *)
(*****************************************)

(* $History$ *)
(* $Log: station2.lotsrc,v $
# Revision 1.1  1997/04/07  16:12:53  cs_s447
# Initial revision
#
# Revision 1.1  1997/04/04  14:28:35  cs_s447
# Initial revision
#     *)

process Station2[h2,l2] : noexit :=

 < .... >

endproc (* Station 2 *)

endproc (* flexint *)

(*********************************************)
(*  LOTOS sources for MSCTestBase and the      *)
(*  manaully derived Unified Tester            *)
(*********************************************)

process TestMSC[h1,h2,l1,l2,success] : noexit :=

  h1!send(ENQ);

  h2!receive(ENQ);
```

```
 h2!send(DLE); h1!receive(DLE);

 h1!send(X_ON); h2!receive(X_ON);

 h2!send(X_ON); h1!receive(X_ON);

 l1!send(LC_Packet); h1!send(LC_P);

 h2!receive(LC_P); l2!receive(LC_Packet);

 h2!send(ACK);

 (i; h1!receive(ACK);

 l2!send(LC_Packet);  h2!send(LC_P);

 h1!receive(LC_P); l1!receive(LC_Packet);

 h1!send(ACK); h2!receive(ACK);

 success; stop

 []

  i; l2!send(LC_Packet);
   (
    i; h1!receive(ACK);  h2!send(LC_P);

     h1!receive(LC_P);  l1!receive(LC_Packet);

     h1!send(ACK); h2!receive(ACK);

     success;stop
    []
     i; h2! send (LC_P); h1! receive(ACK);

     h1!receive(LC_P);  l1!receive(LC_Packet);

     h1!send(ACK); h2!receive(ACK);

     success;stop
    )
 )

endproc (* TestMSC *)


process UnifiedTesterMSCbase[h1,h2,l1,l2,success,
```

```
                              fail_data,fail_gate,fail_both] : noexit :=

 TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](send(ENQ))
>>
 TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](receive(ENQ))
>>
 TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](send(DLE))
>>
 TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](receive(DLE))
>>
 TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](send(X_ON))
>>
 TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](receive(X_ON))
>>
 TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](send(X_ON))
>>
 TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](receive(X_ON))
>>
 TestEvent[l1,h2,h1,l2,fail_data,fail_gate,fail_both](send(LC_Packet))
>>
 TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](send(LC_P))
>>
 TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](receive(LC_P))
>>
 TestEvent[l2,h1,h2,l1,fail_data,fail_gate,fail_both]
                                                  (receive(LC_Packet))
>>
 TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](send(ACK))
>>
 i;(
  TestEventinChoice[h1,h2,l1,l1,fail_data,fail_gate,fail_both]
      (receive(ACK),Insert(send(LC_Packet), Insert_1(receive(ACK),{})))
  >>
  TestEvent[l2,h1,h2,l1,fail_data,fail_gate,fail_both](send(LC_Packet))
  >>
  TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](send(LC_P))
  >>
  TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](receive(LC_P))
  >>
  TestEvent[l1,h1,h2,l2,fail_data,fail_gate,fail_both]
                                                    (receive(LC_Packet))
  >>
  TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](send(ACK))
  >>
  TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](receive(ACK))
  >>
  success; stop
  []
```

```
  (* extra choice - actually redundant
  TestEventinChoice[h1,h2,l1,l1,fail_data,fail_gate,fail_both]
    (receive(X_ON),Insert(send(LC_Packet), Insert_1(receive(X_ON),{})))
  >>
   TestEvent[l2,h1,h2,l1,fail_data,fail_gate,fail_both](send(LC_Packet))
  >>
   TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](send(LC_P))
  >>
   TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](receive(LC_P))
  >>
   TestEvent[l1,h1,h2,l2,fail_data,fail_gate,fail_both]
                                                  (receive(LC_Packet))
  >>
   TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](send(ACK))
  >>
   TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](receive(ACK))
  >>
   success; stop
[]
          *)

  TestEventinChoice[l2,h2,h2,l1,fail_data,fail_gate,fail_both]
    (send(LC_Packet),Insert(receive(ACK), Insert_1(send(LC_Packet),{})))
  >>
   i;( TestEventinChoice[h1,h2,l1,l2,fail_data,fail_gate,fail_both]
          (receive(ACK),Insert(send(LC_P), Insert_1(receive(ACK),{})))
    >>
     TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](send(LC_P))
    >>
     TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](receive(LC_P))
    >>
     TestEvent[l1,h1,h2,l2,fail_data,fail_gate,fail_both]
                                                    (receive(LC_Packet))
    >>
     TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](send(ACK))
    >>
     TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](receive(ACK))
    >>
     success;stop

   []
     TestEventinChoice[h1,h2,l1,l2,fail_data,fail_gate,fail_both]
         (send(LC_P),Insert(receive(ACK), Insert_1(send(LC_P),{})))
    >>
     TestEvent[h1,h1,l1,l2,fail_data,fail_gate,fail_both](receive(ACK))
    >>
     TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](receive(LC_P))
```

```
    >>
     TestEvent[l1,h1,h2,l2,fail_data,fail_gate,fail_both]
                                                    (receive(LC_Packet))
    >>
     TestEvent[h1,h2,l1,l2,fail_data,fail_gate,fail_both](send(ACK))
    >>
     TestEvent[h2,h1,l1,l2,fail_data,fail_gate,fail_both](receive(ACK))
    >>
     success;stop
    )
 )
 endproc (* UnifiedTesterMSCbase *)

 process TestEvent [g,f1,f2,f3, fail_data, fail_gate, fail_both]
             (z: data) : exit :=

(* If the int spec offers more than one choice, then this    *)
(* process testevent will reject at least once choice        *)

  choice zz: Data []
   [zz eq z] ->  (* Correct Data *)
    (
     g!z; exit  (* valid input  *)
    []
     f1!zz; fail_gate; stop
    []
     f2!zz; fail_gate; stop
    []
     f3!zz; fail_gate; stop
    )
  []
   [zz ne z ] -> (* Incorrect Data *)
    (
     g!zz; fail_data; stop  (* but Correct Gate *)
    []
     f1!zz; fail_both; stop
    []
     f2!zz; fail_both; stop
    []
     f3!zz; fail_both; stop
    )

  endproc (* TestEvent *)

  process TestEventinChoice [g,f1,f2,f3, fail_data, fail_gate,fail_both]
              (e:Data, z: Pset) : exit :=

  (* this process copes with choices *)
```

```
  choice zz: Data []
   [zz eq e] ->  (* Correct Data *)
    (
     g!e; exit  (* valid input  *)
    []
     f1!zz; fail_gate; stop
    []
     f2!zz; fail_gate; stop
    []
     f3!zz; fail_gate; stop
    )
  []
   [zz NotIn z ] -> (* Incorrect Data *)
    (
     g!zz; fail_data; stop  (* but Correct Gate *)
    []
     f1!zz; fail_both; stop
    []
     f2!zz; fail_both; stop
    []
     f3!zz; fail_both; stop
    )

  endproc (* TestEventinChoice *)

endspec (* FlexportTestMSC *)
```

D.3 Outputs

The following is a path produced in the expansion of Flexport intensional specification under SMILE2.4. This path leads to a `stop` immediately after the last action shown below.

VERSIONS

```
flexint_t.lot  #Revision 2.1
flexdata.lotdat # Revision 2.1
buffer.lotsrc # Revision 2.1
station1.lotsrc,v # Revision 2.1
station2.lotsrc,v # Revision 1.4
```

PATH

```
baud !9600

h1 !send(ENQ)
  send(x_4) = send(ENQ)
h2 !receive(x_4)
  receive(y1_5) = receive(x_4)
h2 !send(DLE)
  send(x_8) = send(DLE);
  (y1_5 eq ENQ)
tick

baud !9600

h1 !send(ENQ)
  send(x_32) = send(ENQ)
h2 !receive(x_32)
  receive(y2_14) = receive(x_32)
h1 !receive(tx_25)
  receive(x_8) = receive(tx_25)
h2 !send(X_ON)
  send(x_78) = send(X_ON)
h1 !send(X_ON)
  send(x_103) = send(X_ON);
  (tx_25 eq DLE)
h1 !receive(x2_22)
  receive(x_78) = receive(x2_22)
h2 !receive(y1_80)
  receive(x_103) = receive(y1_80);
  (y2_14 ne X_ON)
l1 !send(LC_packet)
  (x2_22 eq X_ON)
h1 !send(LC_P)
```

```
  send(x_254) = send(LC_P)
h2 !receive(y1_265)
  receive(x_254) = receive(y1_265);
  (y1_80 ne ENQ)

------
```

E

Summary of Versions for Flexport

F

Flexport Definition : Ambiguities or other suspected errors

We list below some of the suspected errors in the Universal Flexport protocol. For each error, we specify the relevant page(s) in the document, the aspect concerned, the kind of error, followed by a description of the problem and suggested solution (if any).

1. **Page 1.3, Baud Rate Hunting**:

   Error Type: misprint?

   > "If no recognizable DLE is received, then station 1 drops to the next baud rate and tries sending *DLE* 3 times again."

   Suggestion: The *DLE* should be replaced by ENQ since the Link Connect Protocol and Link Establishment MSC do not mention station 1 transmitting DLE).

2. **Page 1.5, Packet Definition**

   Error type: inconsistency (misprint?)

   The diagram of a packet indicates the checksum at Byte 'length + 3'.

   Suggestion: This should be length + 2, to be consistent with the definition of length as the number of DATA bytes in the packet.

3. **Page 1-4, Error Recovery**

   Error type: incomplete information ?

   Is the error recovery a continuous automatic repeat request (ARQ)? A 'sliding window' is mentioned – which "can have at most 1 packet outstanding". Is this related to window size? (The sequence number is actually 8 bit, which would imply a window size up to 255 ...)

Some other miscellaneous queries arise (not necessarily errors): What about timeout between packet transmission? What if a packet is lost? What if an 'ACK' is damaged?

4. **page 1.4, 'Link alive period'**

   Error Type: ambiguity

   On this page, in the 'Link Alive' section, the 'Link alive period' is defined as equal to (link time-out - 1)

   ("for values greater than 1") ....

   There are two possible interpretations of "for values greater than 1":

   Either:

   (a) This refers to the link time-out. Then, referring to the definition of the Link Configuration Frame, the 'Link Alive' is undefined for

       Link Time-out = 0 (No time-out)

       *Link Time-out = 1 (Time-out time of 1 second)

       Or:

   (b) This refers to the link alive period then, referring to the definition of the Link Configuration Frame, the 'Link Alive' is additionally undefined for

       *Link Time-out = 2 (Time-out time of 2 seconds)

       In both cases, it may be the case that a link alive signal is not sent when there is a timeout of 1 second is defined in the link configuration frame.

G

Risk Management Example

Consider one fault in the FTA, viz: 'Data Corruption due to Upper Level Link Interface'. The Flexport document specifies that data to and from this level is sent in packets. This data may become corrupted, so an error recovery procedure is used as a method of control.

Baseline 4 Branch: 1.1							
Hazard No.	Baseline Entered	Hazard & Cause	Risk	Req'mt [Method of Control]	Req. Ref.	Verification & Validation	RFU
HA-4-1-001	0	Data corruption due to Upper Level Link	II	Error Recovery Procedure		1. <descriptn>. 2. <formulae> 3. <proof>	

Table G.1: Example RM Log for Flexport

where <descriptn> quotes the Flexport document:

1. "If a packet is received and the checksum test is passed, the receiving station sends an ACK followed by the sequence number of the packet. If a packet is received and the checksum test is failed, the receiving station sends a NAK followed by the sequence number of the packet."

2. "If the sending station still has a copy of the packet, it will re-send the requested packet up to one at a time. If the packet is no longer available, the sender sends a NAK followed by the sequence number."

and where <formulae> consists of corresponding formalisations:

1. <modal formulae $\phi_{1,1}, ...$ >

2. < modal formulae $\phi_{2,1}, ...$ >

and where <proofs> may read:

" a) Flexport textual specification `flexint.lotos` transformed by *CÆSAR* into a labelled transition system representation `flexint.aut` in *ALDEBARAN* format. This transformation preserves strong equivalence (LEMMA ...) and therefore temporal properties (LEMMA ...).

b) `flexint.aut` loaded as an automaton into *AUTO* and saved as `flexint.cw` in *Concurrency Workbench* format. Top level process label `flexintensional` identified as agent X also preserving strong equivalence (LEMMA ...) and therefore temporal properties (LEMMA ... as above).

c) `flexint.cw` loaded into *Concurrency Workbench*, whose modal logic verifier shows that formulae $\phi_{1,1}, ...$ hold for agent $X_{\Box}$ "

H

Questionnaire

Bibliography

[1-495] ITU Recommendations X.901-904 ISO/IEC 10746 1-4. *Open Distributed Processing – Reference Model – Parts 1–4*. ISO/ITU-T, 1995.

[ABe+90] Rudy Alderden, Ed Brinksma, Jorg Burmeister (ed.), Jan de Meer (ed.), Tomás Robles, Jan Tretmans, and Clazien Wezeman. A theoretical and methodological framework to conformance testing. Technical Report ESPRIT Ref: 2304; Lo/WP1/T1.3/N0006/V5, LOTOSphere Consortium, 1990.

[Abr87] Samson Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53:225–241, 1987.

[ACJ+96] M. Ardis, J. Chaves, L. Jagadeesan, P. Mataga, C. Puchol, M. Staskauskas, and J. Von Olnhausen. A framework for evaluating specification methods for reactive systems. *IEEE Transactions on Software Engineering*, 22(6):378–389, 1996.

[AH94] L. Aceto and M. Hennessy. Adding action refinement to a finite process algebra.ecember 1994. *Information and Computation*, 115(2):179–247, 1994.

[AHJJ93] M. Andreu, M. Haziza, Claude Jard, and Jean-Marc Jezequel. Analysing a space-protocol: from specification, simulation to experimentation. In *FORTE '92*, pages 187–198. Elsevier Science Publishers B.V. (N.Holland) IFIP, 1993.

[AL89a] Martín Abadi and Leslie Lamport. Composing specifications. In *Stepwise Refinement of Distributed Systems*, volume 430 of LNCS, pages 1–41. Springer-Verlag, 1989.

[AL89b] Pierre Azema and François Vernadat Jean-Christophe Lloret. Requirements analysis for communication protocols. In *Automatic Verification Methods for Finite State Systems and Grenoble and FRANCE*, volume 407 of LNCS, pages 286–293. Springer-Verlag [Ed. J.Sifakis], 1989.

[ASvS89] I. Ajubi, G. Scollo, and M. van Sinderen. Formal description of the osi session layer (4 papers). In *FORTE '88 – The Formal Description Technique LOTOS*, pages 89–210. Elsevier Science Publishers, P.O. Box 103, 1000 AC Amsterdam, The Netherlands, 1989.

[B. 89] B. A. Wichmann, Ed. Software in safety-related systems. Technical report, IEE and BCS, 1989.

[BA90] M. Ben-Ari. *Principles of Concurrent and Distributed Programming.* Prentice Hall, 1990.

[BA91] Glenn Bruns and Stuart Anderson. The formalization and analysis of a communications protocol. Technical Report LFCS Report Series ECS-LFCS-91-137, LFCS, Department of Computer Science, The University of Edinburgh, 1991.

[BA93] G. Bruns and S. Anderson. Validating safety models with fault trees. In *SAFECOMP'93*, pages 21–30. Springer -Verlag, 1993.

[BAL+89] Ed Brinksma, Rudie Alderden, Rom Langerak, Jaroen van de Lagemaat, and Jan Tretmans. A formal approach to conformance testing. In *2nd International Workshop on Protocol Test Systems.* IFIP, 1989.

[Bar95] Geoff Barrett. Model checking in practice: The T9000 virtual channel processor. *IEEE Transactions on Software Engineering*, 21(2):69–78, 1995.

[BB87] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1987.

[BB89] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. In *FORTE '88*, pages 23–73. Elsevier Science Publishers B.V., 1989.

[BBBC94] Howard Bowman, Gordon S. Blair, Lynee Blair, and Amanda G. Chetwynd. Time versus abstraction in formal description. In *FORTE '93*, pages 467–482. Elsevier Science B.V., Sara Bergerhartstraat 25, P.O. Box 211, 1000 AC Amsterdam, The Netherlands, 1994.

[BCG91] R. E. Bloomfield, J. H. Cheng, and J. Górksi. Towards a Common Safety Description Model. In *SAFECOMP'91*, pages 1–6. IFAC, 1991.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^20 states and beyond. *Information and Computation*, 98:142–170, 1992.

[Bea91] S. Bear. An overview of HP-SL. In *VDM '91, Formal Software Development Methods*, volume 551 of LNCS. Springer Verlag, 1991.

[BF93] Ricky W. Butler and George B. Finelli. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, 19(1):3–12, 1993.

[BGMW95] Howard Barringer, Graham Gough, Brian Monahan, and Alan Williams. The state evolution method for verifying hardware systems. Technical Report UMCS-95-7-1, Department of Computer Science, University of Manchester, 1995.

[BH95] Jonathan P. Bowen and Michael G. Hinchey. Seven more myths of formal methods. *IEEE Software*, 12(4):34–41, 1995.

[BHR84] S. D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the Association of Computing Machinery*, 31(3):560–599, 1984.

[Bib95] B.R.M. Bibb. A new standard for PEMS and its application to radiotherapy systems. *Computing and Control Engineering Journal*, 6(5):226–232, 1995.

[BK84] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60:109–137, 1984.

[BL92] T. Bolognesi and F. Lucidi. LOTOS-like process algebras with urgent or timed interactions. In *FORTE '91*, pages 249–264. Elsevier Science Publishers, P.O. Box 103, 1000 AC Amsterdam, The Netherlands, 1992.

[BL95] D. W. Bustard and P. J. Lundy. Enhancing soft systems analysis with formal modelling. Technical Report 10 (ISSN 0966-4963), School of Information and Software Engineering, University of Ulster, 1995.

[Boo91] Grady Booch. *Object Oriented Design with Applications*. Benjamin/Cummings, 1991.

[BR93] R. Bell and D. Reinert. Risk and system integrity concepts for safety-related control systems. *Microprocessors and Microsystems*, 17(1):3–15, 1993.

[Bri83] British Standards Institute, 389 Chiswick High Road, London W4 4AL. *BS5760, Part 1 - Guide to Reliability and Programme Management*, 1983.

[Bri87] Ed Brinksma. On the existence of canonical testers. Technical report, University of Twente, Department of Computer Science, 1987.

[Bri89] Ed Brinksma. A theory for the derivation of tests. In *PSTV VIII: IFIP WG 6.1*, pages 235–247. Elsevier Science Publishers, P.O. Box 103, 1000 AC Amsterdam, The Netherlands, 1989.

[Bri91] British Standards Institute, 389 Chiswick High Road, London W4 4AL. *BS5760, Part 5 - Reliability of Systems, equipment and components: Guide to failure modes, effects and criticality analysis*, 1991.

[Bri92] Ed Brinksma. What is the method in formal methods? In *FORTE '91*, pages 33–50. Elsevier Science Publishers, P.O. Box 103, 1000 AC Amsterdam, The Netherlands, 1992.

[Bru93] Glenn Bruns. A practical technique for process abstraction. In *CONCUR '93*, volume 715, pages 37–49. Springer-Verlag, 1993.

[Bru94] Glenn Bruns. Applying process refinement to a safety-relevant system. Technical Report LFCS Report Series ECS-LFCS-94-287, LFCS, Department of Computer Science, The University of Edinburgh, 1994.

[Bru95] Glenn Bruns. Refinement and dependable systems. In *10th Annual IEEE Conference on COMPuter ASSurance (COMPASS 95)*, 1995.

[Bry95] Randal E. Bryant. Binary decision diagrams and beyond: Enabling technologies for formal verification. In *International Conference on Computer Aided Design (ICCAD '95)*, 1995.

[BS93a] Jonathan Bowen and Victoria Stavridou. The industrial take-up of formal methods in safety-critical and other areas: A perspective. In *FME' 93: First International Symposium on Formal Methods Europe, Odense, Denmark*, volume 670 of LNCS, pages 183–195. Springer-Verlag, 1993.

[BS93b] Jonathan Bowen and Victoria Stavridou. Safety-critical systems and formal methods and standards. *Software Engineering Journal*, 18(04):189–209, 1993.

[BSS86] Ed Brinksma, Giuseppe Scollo, and Chris Steenbergen. LOTOS specifications and their implementations and their tests. In *6th International Symposium on Protocol Specification and Testing and Verification (PSTV) VI*, pages 349–360. Elsevier Science Publishers B.V., The Netherlands, 1986.

[BvdLV95] Tommaso Bolognesi, Jeroen van de Lagemaat, and Chris Vissers, editors. *LOTOSphere: Software Development with LOTOS*. Kluwer Academic Publishers, 1995.

[BW90] J. C. Baeten and W. P. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[BW94] David W. Bustard and Adam C. Winstanley. Making changes to formal specifications. *IEEE Transactions on Software Engineering*, 20(8):562–568, 1994.

[Bye92] P. J. Byers. Definition of requirements for a safety-critical communications system. In *SETSS 92: Eighth International Conference on Software for Telecommunication Systems and*, volume 352, pages 98–101. IEE Conference Publications, London, 1992.

[CBH91] J. L. Cyrus, J.D. Bledsoe, and P.D. Harry. Formal specification and structured design in software development. *Hewlett-Packard Journal*, 6:51–58, 1991.

[CBM+95] Morris Chudleigh, Chris Berridge, Rod May, Jenny Butler, and Ian Poole. The SADLI project - safety critical software research in the medical diagnostic domain. *Computing and Control Engineering Journal*, 6(5):211–215, 1995.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CFH95] K. Chan, C. Fencott, and B. Hebbron. Formal support for the safety analysis of requirement models. In *SAFECOMP'95*, pages 75–89. Springer -Verlag, 1995.

[CG93] F. J. Carrasco and J. J. Gil. A method for specifying and validating communication protocols in LOTOS. In *FORTE '92*, pages 247–262. Elsevier Science Publishers, P.O. Box 103, 1000 AC Amsterdam, The Netherlands, 1993.

[CGR95] Dan Craigen, Susan Gerhart, and Ted Ralston. Formal methods reality check: Industrial usage. *IEEE Transactions on Software Engineering*, 21(2):90–98, 1995.

[CGW91] W. J. Cullyer, S. J. Goodenough, and B.A. Wichmann. The choice of computer languages for use in safety-critical systems. *Software Engineering Journal*, 6(2):51–58, 1991.

[CH89] Rance Cleaveland and Matthew Hennessy. Testing equivalence as a bisimulation equivalence. In *Automatic Verification Methods for Finite State Systems, Grenoble, FRANCE [Proceedings]*, volume 407 of LNCS, pages 011–023. Springer-Verlag [Ed. J.Sifakis], 1989.

[Che81] P.B. Checkland. *System Thinking, System Practice*. Wiley, 1981.

[Che87] Chemical Industry Associates Ltd. *A Guide to Hazard and Operability Stuides*, 1987.

[CN92] Phil Curran and Kathy Norrie. An approach to verifying concurrent systems - a medical information bus (MIB) case study. In *5th Annual IEEE Symposium: Computer Based Medical Systems*, pages 74–83. IEEE, 1992.

[CNS96] Phil Curran, Kate Norrie, and Mike Spicer. A rigorous approach to assessing medical device communication standards. In *Medical Informatics Europe '96 and Copenhagen*, 1996.

[CPS89] Rance Cleaveland, Joachim Parrow, and Bernard Steffen. The Concurrency Workbench. In *Automatic Verification Methods for Finite State Systems and Grenoble and FRANCE [Proceedings]*, volume 407 of LNCS, pages 24–37. Springer-Verlag [Ed. J.Sifakis], 1989.

[CS93] J.P. Courtiat and D.E. Saïdouni. Action refinement in LOTOS. In *13th International Symposium on Protocol Specification, Testing and Verification (PSTV)*, pages 341–354. Elsevier Science Publishers B.V., The Netherlands, 1993.

[DBBS96] John Derrick, Eerke Boiten, Howard Bowman, and Maarten Steen. Supporting ODP – translating LOTOS to Z. In *First IFIP International Workshop on Formal Methods for Open Object-based Distributed Systems, Paris*. Chapman & Hall., 1996.

[Def95] Defence Standards - UK Ministry of Defence, [Director of Standardization] Kentigern House 65 Brown Street GLASGOW G2 8EX. *A Guideline for HAZOP Studies on Systems which include a Programmable Electronic System*, 1995.

[DLG92] Hana De-Leon and Orna Grumberg. Modular abstractions for verifying real-time distributed systems. In *3rd International Workshop on Computer Aided Verification and CAV '92*, pages 2–15. Springer-Verlag, 1992.

[dLSA91] R. de Lemos, A. Saeed, and T. Anderson. Analysis of timeliness requirements in safety-critical systems. In *Formal Techniques in Real-Time and Fault-Tolerant Systems, 2nd International Symposium, Nijmegen, The Netherlands.*, volume 571 of LNCS, pages 171–192. Springer-Verlag, 1991.

[dLSA94] R. de Lemos, A. Saeed, and T. Anderson. Requirements analysis for safety-critical systems: a chemical batch processing example. Technical report, Centre for Software Reliability, University of Newcastle upon Tyne, 1994.

[dLSA95] Rogério de Lemos, Amer Saeed, and Tom Anderson. Analyzing safety requirements for process control systems. *IEEE Software*, pages 42–53, 1995.

[EFT91] Reinhard Enders, Thomas Filkorn, and Dirk Taubner. Generating BDDs for symbolic model checking in CCS. In *3rd International Workshop and CAV '91*. Springer-Verlag, 1991.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specifications*. Springer-Verlag, Berlin, 1985.

[EW93] Henk Eertink and Dietmar Wolz. Symbolic execution of LOTOS specifications. In *FORTE '92*, pages 295–310. IFIP, 1993.

[Fel79] Stuart I. Feldman. Make - a program for maintaining computer programs. *Software Practice and Experience*, 9:255–265, 1979.

[Fel95] Felix Redmill, Editor. Special feature: Medical safety systems. *Computing and Control Engineering Journal*, 6(5):202–240, 1995.

[Fer88] J.C. Fernandez. *Aldébaran, Un Système de véification par réduction de processus communicants*. PhD thesis, Université de Grenoble, 1988.

[FGM+92] Jean-Claude Fernández, Hubert Garavel, Laurent Mounier, Anne Rasse, Carlos Rodríguez, and Joseph Sifakis. A toolbox for the verification of LOTOS programs. In *14th International Conference on Software Engineering (ICSE 14)*, pages 246–259, 1992.

[FHMS93] Justin Forder, Chris Higgins, John McDermid, and Graham Storrs. Sam - a tool to support the construction, review and evolution of safety arguments. *Directions in Safety-critical systems: Proceedings of the Safety-critical Systems Symposium*, pages 195–216, 1993.

[FJMP94] P. Fenelon, J.A.McDermid, M.Nicholson, and D.J. Pumfrey. Towards integrated safety analysis and design. *ACM Computing Reviews*, 2(1):21–32, 1994?

[FKN+92] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *Journal of Software Engineering and Knowledge Engineering*, 2:31–57, 1992.

[FM91] J. Fernandez and L. Mounier. Verifying bisimulations of the fly. In *FORTE '90*, pages 91–107. Elsevier Science Publishers, P.O. Box 103, 1000 AC Amsterdam, The Netherlands, 1991.

[For94] Formal Methods and Tools Research Groups. VALIDE: Formal methods and tools for distributed system design. Technical report, Dept. of Tele-Informatics and Open Systems, University of Twente, 1994.

[FST92] Siegfried Fischer, Andreas Scholz, and Dirk Taubner. Verification in process algebra of the distributed control of track vehicles - a case study. In *3rd International Workshop on Computer Aided Verification and CAV '92*, pages 192–205. Springer-Verlag, 1992.

[G94] J. Górksi. Extending safety analysis techniques with formal semantics. In *Second Safety-critical Systems Symposium and Birmingham*, pages 147–163. Springer-Verlag and London, 1994.

[GD95] David Garlan and Norman Delisle. Formal specification of an architecture for a family of instrumentation systems. In *Applications of Formal Methods*, pages 55–72. Prentice Hall International, 1995.

[GG89] Stephen J. Garland and John V. Guttag. An overview of lp, the larch prover. in third international. In *3rd International Conference on Rewriting Techniques and Applications*, volume 355 of LNCS, pages 137–151. Springer Verlag, 1989.

[GH93] Patrice Godefroid and Gerard J. Holzmann. On the verification of temporal properties. In *13th International Symposium on Protocol Specification and Testing and Verification (PSTV XIII)*, pages 109–124. Elsevier Science Publishers B.V., The Netherlands, 1993.

[Gib93] J. Paul Gibson. *Formal Object-oriented Development of Software using LOTOS*. PhD thesis, University of Stirling, 1993.

[Gib94] W. Wayt Gibbs. Software's chronic crisis. *Scientific American*, pages 72–81, 1994.

[GM93] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[GMW79] M. J. Gordon, A. J. Milner, and C.P. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*. Springer-Verlag, 1979.

[GMW95] J. Górksi, Jan Magott, and A. Wardziński. Modelling fault trees using Petri Nets. In *14th International Conference on Computer Safety and Reliability and Security SAFECOMP'95*, 1995.

[GRRJ89] S. Graf, J.-L. Richier, C. Rodríguez, and J.Voiron. What are the limits of model checking methods for the verification of real life protocols? In *Automatic Verification Methods for Finite State Systems and Grenoble and FRANCE [Proceedings]*, volume 407 of LNCS, pages 275–285. Springer-Verlag [Ed. J.Sifakis], 1989.

[GTHE89] R. M. Gardner, H. Tariq, W. L. Hawley, and T. D. East. Editorial; medical information bus: The key to future integrated monitoring. *International Journal of Clinical Monitoring and Computing*, 6:205–209, 1989.

[GW91] Patrice Godefroid and Pierre Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *3rd International Workshop and CAV '91*, pages 332–342. Springer-Verlag, 1991.

[GW95] Janusz Górski and Andrzej Wardziński. Formalising fault trees. In *Third Safety-critical Systems Symposium, Brighton*, pages 311–327. Springer-Verlag and London, 1995.

[GW96] J. Górski and A. Wardziński. Deriving real-time requirements for software from safety analysis. In *8th EUROMICRO Workshop on Real-Time Systems*, 1996.

[Har82] Peter R. Harvey. Fault tree analysis of software. Master's thesis, University of California at Irvine, 1982.

[Har87] David Harel. *Algorithmics - The Spirit of Computing.* Addison-Wesley, 1987.

[HB95] Michael G. Hinchey and Jonathan P. Bowen, editors. *Applications of Formal Methods.* Prentice Hall, 1995.

[Hen88] Matthew Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[HL96] Mats P.E. Heimdahl and Nancy G. Leveson. Completeness and consistency in hierarchical state-based requirements. *IEEE Transactions on Software Engineering*, 1996.

[HM85] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the Association of Computing Machinery*, 32(1):137–161, 1985.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[HvB94] Teruo Higashino and Gregor von Bochmann. Automatic analysis and test case derivation for a restricted class of LOTOS expressions with data parameters. *IEEE Transactions on Software Engineering*, 20(1):29–42, 1994.

[Ins92] Institute of Electrical Engineers, London, UK. *IEEE Std. P1073 - IEEE Standards for Medical Device Communications: Overview and Architecture*, 1992.

[Ins94a] Institute of Electrical and Electronic Engineers. *1073.3.1 IEEE Standard for Medical Device Communications – Transport Profile – Connection Mode*, 1994.

[Ins94b] Institute of Electrical and Electronic Engineers. *1073.4.1 IEEE Standard for Medical Device Communications – Physical Layer Interface – Cable Connected*, 1994.

[Int84] Internation Standards Organisation. *ISO 7498. Information Processing Systems - Open Systems Interconnection - Basic Reference Model*, 1984.

[Int91] International Electrotechnical Commission. *IEC 65A (Secretariat) 122: Software for Computers in the application of industrial safety related systems*, 1991.

[Int92] International Electrotechnical Commission. *IEC 65A (Secretariat) 123: Functional Safety of electrical/electronic/programable systems. Generic aspects Part 1: General Requirements*, 1992.

[Int93] International Telecommunications Union, Geneva. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*, 1993.

[Int96] International Telecommunications Union, Geneva. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*, 1996.

[ISO89a] ISO/IEC 8807. *Information Processing Systems - Open Systems Interconnection - LOTOS - a formal description technique based on the temporal ordering of observational behaviour*, 1989.

[ISO89b] ISO/IEC 9074. *Information Processing Systems - Open Systems Interconnection - ESTELLE - a formal description technique based on an Extended State Transition Model*, 1989.

[ISO89c] ISO/TC97/SC21. *Information Processing Systems, Open Systems Interconnection, OSI Conformance Testing Methodology and Framework.* ISO 9646, International Standards Organisation, 1989.

[ISO95] ISO Working Group on E-LOTOS. Iso/iec jtc1/sc21/wg7/e-lotos meeting. Technical report, ISO/IEC, 1995.

[J. 94] J. Preece et al. *Human-Computer Interaction.* Addison-Wesley, 1994.

[Jac93] Jonathan Jacky. Specifying a safety-critical control system in Z. In *FME' 93: First International Symposium on Formal Methods Europe and Odense and Denmark*, volume 670 of LNCS, pages 388–402. Springer-Verlag, 1993.

[Jon90] Cliff B. Jones. *Systematic Software Development using VDM.* Prentice-Hall International, 2nd edition,1990.

[Kir94] Carron Kirkwood. *Verification of LOTOS Specifications Using Term Rewriting Techniques.* PhD thesis, University of Glasgow, 1994.

[Kir95] Carron Kirkwood. Specifying properties of Basic LOTOS processes using temporal logic. In *FORTE '95.* Chapman and Hall, London, UK, 1995.

[Klu91] A. S. Klusener. Abstraction in real time process algebra. In *Real-Time: Theory in Practice*, volume 600 of LNCS, pages 326–352, 1991.

[Kuh92] D. Richard Kuhn. A technique for analyzing the effects of changes in formal specifications. *The Computer Journal*, 1992.

[KvdLRS93] H. Kremer, J. van de Lagemaat, A. Rennoch, and G. Scollo. Protocol design using LOTOS: A critical synthesis of a standardisation experience. In *FORTE '92*, pages 231–246. IFIP, 1993.

[KZ95] D. Kapur and H. Zhang. An overview of rewrite rule laboratory. *Journal of Mathematics of Computation*, 29(2):91–114, 1995.

[Lam77] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977.

[Lan90] Rom Langerak. A testing theory for LOTOS using deadlock detection. In *9th International Symposium on Protocol Specification, Testing and Verification (PSTV IX)*, pages 87–98. Elsevier Science Publishers B.V., The Netherlands, 1990.

[Lap93] Jean-Claude Laprie. Dependability: from concepts to limits. In *SAFECOMP'93*, pages 157–168. Springer -Verlag, 1993.

[LCS91] N. G. Leveson, S. S. Cha, and T. J. Shimeall. Safety verification of Ada programs using software fault trees. *IEEE Software*, 8(7):48–59, 1991.

[Led91] Guy Leduc. Conformance relation, associated equivalence, and minimum canonical tester in lotos. In *11th International Symposium on Protocol Specification, Testing and Verification (PSTV XI)*, pages 249–264. Chapman and Hall, London, UK, 1991.

[Lev86] Nancy G. Leveson. Software safety: Why, What and How. *ACM Computing Surveys*, 18(2):125–163, 1986.

[Lev91] Nancy G. Leveson. Software safety in embedded computer systems. *Communications of the ACM*, pages 35–46, 1991.

[LF91] B.R. Ladeau and C. Freeman. Using formal specification for product development. *Hewlett-Packard Journal*, 6:62–66, 1991.

[LGS+95] Claire Loiseaux, Susanne Graf, Joseph Sifakis, Ahmed Bouajjani, and Saddek Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–35, 1995.

[LH83] Nancy G. Leveson and Peter R. Harvey. Analysing software safety. *IEEE Transactions on Software Engineering*, 9(5):569–579, 1983.

[Lin91] Huimin Lin. PAM: A process algebra manipulator. In *3rd International Workshop and CAV '91*, pages 136–146. Springer-Verlag, 1991.

[Lin93] H. Lin. A verification tool for value-passing processes. In *13th International Symposium on Protocol Specification and Testing and Verification (PSTV XIII)*, pages 79–92. Elsevier Science Publishers B.V., The Netherlands, 1993.

[Lio96] J.L. Lions, Chairman. Ariane 5: Flight 501 failure - report by the inquiry board. Technical report, European Space Agency/CNES, Paris, 1996.

[LL94] Peter B. Ladkin and Stefan Leue. What do message sequence charts mean? In *FORTE '93*, pages 301–316. Elsevier Science B.V., Sara Bergerhartstraat 25, P.O. Box 211, 1000 AC Amsterdam, The Netherlands, 1994.

[Lla91] Santiago Pavón Martín Llamas. The testing functionalities of LOLA. In *FORTE '90*, pages 559–562. Elsevier Science Publishers, P.O. Box 103, 1000 AC Amsterdam, The Netherlands, 1991.

[LM86] Kim G. Larsen and Robin Milner. A complete protocol verification using relativized bisimulation. Technical Report LFCS Report Series ECS-LFCS-86-13, LFCS, Department of Computer Science, The University of Edinburgh, 1986.

[LOT92a] LOTOSphere Consortium. Catalogue of lotos correctness preserving transformations, CEC deliverable Lo/WP1/T1.2/N0045/V04. Technical report, Esprit Project 2034, 1992.

[LOT92b] LOTOSphere Consortium. The LOTOSphere design methodology: Basic concepts, CEC deliverable Lo/WP1/T1.1/N0045/V04. Technical report, Esprit Project 2034, 1992.

[LOT92c] LOTOSphere Consortium. The LOTOSphere design methodology: Guidelines, CEC deliverable Lo/WP1/T1.1/N0044/V04. Technical report, Esprit Project 2034, 1992.

[LS87] Nancy G. Leveson and Janice L. Stolzy. Safety analysis using Petri Nets. *IEEE Transactions on Software Engineering*, 13(3):386–397, 1987.

[LT88] Kim G. Larsen and Bent Thomsen. A modal process logic. In *Third Annual Symposium on Logic in Computer Science*, pages 203–210. IEEE, 1988.

[LT93] Nancy Leveson and Clark S. Turner. An investigation of the therac-25 accidents. *IEEE Computer*, 26(7):18–41, 1993.

[Mai94] Main Commission. Report on the accident to airbus a320-211 aircraft in warsaw on 14 september 1993. Technical report, Aircraft Accident Investigation Warsaw, 1994.

[MC93] J.D. Morison and A.S. Clarke. *ELLA2000: A Language for Electronic System Design.* McGraw-Hill, 1993.

[McD93] John A. McDermid. Safety-critical software: a vignette. *Software Engineering Journal*, pages 002–003, 1993.

[McD94] J.A. McDermid. Support for safety cases and safety arguments using SAM. *Reliability Engineering and Systems Safety*, 43(3), 1994.

[MdS93] E. Madeleine and R. de Simone. Fc2: Reference manual version 1.1. Technical report, INRIA Sophia-Antipolis, 1993.

[MFV93] C. Miguel, A. Fernández, and L. Vidalier. Extending LOTOS towards performance evaluation. In *FORTE '92*, pages 103–116. Elsevier Science Publishers B.V. (N.Holland) IFIP, 1993.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[Mou91] Simon Bainbridge; Laurent Mounier. Specification and verification of a reliable multicast protocol. Technical Report HPL-91-163, Software Engineering Department Hewlett Packard, 1991.

[MP81] Z. Manna and A. Pnueli. Verification of concurrent programs: the temporal framework. In *The Correctness Problem in Computer Science*, pages 215–273. Academic Press, 1981.

[MV92a] Eric Madelaine and Didier Vergamini. Specification and verification of a sliding window protocol in LOTOS. In *FORTE '91*, pages 495–510. Elsevier Science Publishers, P.O. Box 103, 1000 AC Amsterdam, The Netherlands, 1992.

[MV92b] Eric Madeleine and Didier Verganini. Tool demonstration: Tools for process algebras. In *FORTE '91*, pages 463–470. Elsevier Science Publishers, P.O. Box 103, 1000 AC Amsterdam, The Netherlands, 1992.

[NC96] Kate Norrie and Phil Curran. Using formal methods to enhance the quality of a standard for medical device communications. In *First Workshop on Formal Methods in Software Practice and San Diego*, pages 132–140. ACM Sigsoft, 1996.

[NH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 24:83–113, 1984.

[ORR+96] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In *8th International Workshop, CAV '96*. Springer-Verlag, 1996.

[Ost92] Jonathan S. Ostroff. Formal methods for the specification and design of real-time safety-critical systems. *Journal of Systems Software*, 18:33–60, 1992.

[Ost94] Jonathan S. Ostroff. Automated modular specification and verification of real-time reactive systems. Technical Report CS-ETR-94-06, Department of Computer Science, York University, 1994.

[PA94] Graeme I. Parkin and Stephen Austin. Overview: Survey of formal methods in industry. In *FORTE '93*, pages 189–203. Elsevier Science B.V., Sara Bergerhartstraat 25, P.O. Box 211, 1000 AC Amsterdam, The Netherlands, 1994.

[Par72] D.L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[Par80] David Park. Concurrency and automata on infinite sequences. In *5th GI Conference, LNCS 104*, volume 104 of LNCS, pages 167–183. Springer-Verlag, 1980.

[Pen91] A. D. Pengelly. An early warning method for safety-critical software development. In *SAFECOMP '91*, pages 113–116. IFAC, 1991.

[Pet81] J.L. Peterson. *Petri Net theory and the modelling of systems.* Prentice-Hall, 1981.

[Phi87] Iain Phillips. Refusal testing. *Theoretical Computer Science*, 50(2):241–284, 1987.

[PvEE92] The LOTOSphere Consortium Peter van Eijk (Ed.). Tool demonstration: The LOTOSphere integrated tool environment LITE. In *FORTE '91*, pages 471–474. Elsevier Science Publishers, P.O. Box 103, 1000 AC Amsterdam, The Netherlands, 1992.

[PWCC95] M.C. Paulk, C.V. Weber, B. Curtis, and M.B. Chrissis. *The Capability Maturity Model: Guidelines for improving the software process.* Addison-Wesley, New York, 1995.

[QPF89] Juan Quemada, Santiago Pavón, and Angel Fernández. State exploration by transformation with LOLA. In *Automatic Verification Methods for Finite State Systems, Grenoble, FRANCE [Proceedings]*, volume 407 of LNCS, pages 294–302. Springer-Verlag [Ed. J.Sifakis], 1989.

[QS81] J. Quielle and J. Sifakis. Specification and verification of concurrent programs in cesar. In *5th Int. Symp. on Programming*, volume 137 of LNCS. Springer Verlag, 1981.

[SBD95] M.W.A. Steen, H. Bowman, and J. Derrick. Composition of lotos specifications. In *15th International Symposium on Protocol Specification, Testing and Verification (PSTV XV)*, pages 87–102. Chapman & Hall, 1995.

[SdLA95] A. Saeed, R. de Lemos, and T. Anderson. Safety analysis for requirements specifications: Methods and techniques. In *SAFECOMP'95*, pages 27–41. Springer -Verlag, 1995.

[Sha89] M. Michael Shabot. Standardised acquisition of bedside data: The IEEE P1073 medical information bus. *International Journal of Clinical Monitoring and Computing*, 6:197–204, 1989.

[Sis94] A. Prasad Sistla. Safety and liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6(5):495–511, 1994.

[Sis95] Riccardo Sisto. A method to build symbolic representations of lotos specifications. In *15th International Symposium on Protocol Specification, Testing and Verification (PSTV XV)*, pages 331–346. Chapman & Hall, 1995.

[Som92] Ian Somerville. *Software Engineering*. Addison-Wesley, 4th edition, 1992.

[Spa89] SpaceLabs, Inc, 4200 150th Avenue, N.E., PO Box 97013, Redmond, WA 98073-9713. *Universal Flexport Protocol*, 1989.

[Spi92] J. M. Spivey. *The Z Notation: A Reference Manual*. (2nd edition), Prentice Hall, 1992.

[Sti92a] Colin Stirling. Modal and temporal logics. In *Handbook of Logic in Computer Science*, volume 2, pages 477–563. Cambridge University Press, 1992.

[Sti92b] Colin Stirling. Modal and temporal logics for processes. Technical Report LFCS Report Series ECS-LFCS-92-221, LFCS, Department of Computer Science, The University of Edinburgh, 1992.

[SW89a] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In *Theory and Practice of Software Development (TAPSOFT)*, volume 351 of LNCS, pages 369–383. Springer-Verlag, 1989.

[SW89b] Colin Stirling and David Walker. CCS, liveness, and local model checking in the linear time mu-calculus. In *Automatic Verification Methods for Finite State Systems, Grenoble, FRANCE [Proceedings]*, volume 407 of LNCS, pages 166–178. Springer-Verlag [Ed. J.Sifakis], 1989.

[SW90] M. Michael Shabot and Jan Wittenber. The medical device data language for the P1073 medical information bus standard. *International Journal of Clinical Monitoring and Computing*, 7:91–98, 1990.

[T.F95] T.F. Melham, Editor. Special issue: theorem proving applications. *The Computer Journal*, 38(2), 1995.

[Tho93a] Martyn Thomas. The industrial use of formal methods. *Microprocessors and Microsystems*, 17(1):31–36, 1993.

[Tho93b] Muffy Thomas. A translator for ASN.1 into LOTOS. In *FORTE 92*, pages 037–052. IFIP, 1993.

[Tho94] Muffy Thomas. The story of the Therac-25 in LOTOS. *High Integrity Systems Journal*, 1(1):3–15, 1994.

[Tic85] Walter F. Tichy. RCS — A system for version control. *Software – Practice and Experience*, 13(7):637–654, 1985.

[TN96] Paul Trafford and Kate Norrie. Safety-oriented formal refinement of concurrent systems. Technical Report KUCSES-96-04, School of Computer Science and Electronic Systems and Kingston University, 1996.

[Tre94] Jan Tretmans. A formal approach to conformance testing. In *Sixth International Workshop on Protocol Test Systems*, pages 257–276. IFIP, 1994.

[Tur93] Kenneth J. Turner, editor. *Using Formal Description Techniques*. Wiley, 1993.

[UK 97] UK Department of Trade and Industry, Engineering and Physical Sciences Research Council, Distributor: MFR Software Services Ltd, 35 Benbrook Way,Macclesfield, Cheshire, SK11 9RT. *Reports and other Outputs from the DTI/EPSRC-funded R&D Programme in Safety-critical systems*, Issue 2, January 1997.

[VGRH81] W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl. *The Fault Tree Handbook*. US Nuclear Regulatory Commission, 1981.

[VS95] A. Valmari and R. Savola. Verification of the behaviour of reactive software with CFFD-semantics and ARA tools. In *International Symposium on On-board Real-time Software, ESTEC*, pages 173–180, 1995.

[VvSP93] C.A. Vissers, M. van Sinderen, and L.F. Pires. What makes industries believe in formal methods. In *13th International Symposium on Protocol Specification, Testing and Verification (PSTV XIII, Liege)*, pages 3–25. Elsevier Science Publishers B.V., The Netherlands, 1993.

[WBL91] C. D. Wezeman, S. Batley, and J. A. Lynch. Formal methods to assist conformance testing: a case study. In *FORTE '90*, pages 157–174. Elsevier Science Publishers, P.O. Box 103, 1000 AC Amsterdam, The Netherlands, 1991.

[Wez89] C.D. Wezeman. The CO-OP method for compositional derivation of conformance testers. In *9th International Symposium on Protocol Specification, Testing and Verification (PSTV IX)*, pages 145–158. Elsevier Science Publishers B.V., The Netherlands, 1989.

[Whi91] David Whitgift. *Methods and Tools for Software Configuration Management.* Wiley, 1991.

[Wic92] B.A. Wichmann, editor. *Software in safety-related systems.* Wiley, 1992.

[WM85] P. T. Ward and S. J. Mellor. *Structured Methods for Real Time systems, (3 Volumes).* Prentice-Hall, 1985.

[Z.192] CCITT Z.100. *Specification and Description Language*, 1992.

[Zui89] Han Zuidweg. Verification by abstraction and bisimulation. In *Automatic Verification Methods for Finite State Systems, Grenoble, FRANCE [Proceedings]*, volume 407 of LNCS, pages 105–116. Springer-Verlag [Ed. J.Sifakis], 1989.