Bibliography

[Ash92] P. Ashkar. Symbolic Execution of LOTOS Specifications. Master's Thesis, University of Ottawa, 1992.

[BAPM83] M. Ben-Ari, A. Pnueli, and Z. Manna. The Temporal Logic of Branching Time. In *Proc. 8th Ann. Symp. on Principles of Programming Languages* (1981) 164-176; Journal version, *Acta Informatica*, (1983) 20:207-226.

[BB87] B. Bolognesi, E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, (1987) 14: 25-59.

[BBFM82] H.K. Berg, W.E Boebert, W.R. Franta, and T.G. Mober. *Formal Methods of Program Verification and Specification*.Prentice-Hall, Engelwood-Cliffs, NJ 1982.

[BC89] T. Bolognesi, M. Caneve. Squiggles: A tool for the Analysis of LOTOS Specifications. In K. Turner (ed.), *Formal Description Techniques,*(North-Holland, Amsterdam, 1989) 201-216.

[Bou91] R. Boumezbeur. Design, Specification, and Validation of Telephony Systems in LOTOS. Master's Thesis, University of Ottawa, 1991.

[CH87] Ana R. Cavalli, F. Horn. Proof of Specification properties by Using Finite State Machines and Temporal logic. In H. Rudin and C. West, (eds.), *Protocol Specification, Testing, and Verification*, *VII*, (North-Holland), 1987.

[CCITT87] Recommendation Z.100: Specification and Description Language SCL, CCITT SG X, Contribution Com X-R 15-E, 1987.

[CE81] E.M. Clarke, E. A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In *Proc. Workshop on Logics of Programs*, *Lecture Notes in Computer Science*, (Springer, Berlin, 1981) 131:52-71.

[CES83] E.M. Clarke, E.A. Emerson and A.P. Sistla. Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications - A practical approach. In *Tenth Annual ACM Symposium on Principles of Programming Languages,* (Austin, Texas, January 1983) 117-126.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications. *ACM TOPLAS*, 8(2):244-263, April 1986.

[CP88] P. Camurati, P. Prinetto. Formal Verification of Hardware Correctness: Introduction and Survey of current Research. *IEEE Computer*, 21(7):8-19, July 1988.

[EC81] E.A. Emerson, E.M. Clarke, Characterizing Correctness Properties of Parallel Programs as Fixpoints. In *Proc. 7th Internat. Coll. on Automata, Languages, and Programming, Lecture Notes in Computer Science*, (Springer, Berlin, 1981) 85:169-181.

[Emer90] E. Allen Emerson. Temporal and Modal Logic. In J. van Leewen, (ed.), *Handbook of Theoretical Computer Science*. (Elsevier Science Publishers B.V, 1990) Chapter 16.

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

[Feh87] M.C. Fehri. A System for Validating and Executing LOTOS Data Abstractions (SVELDA). Master's Thesis, University of Ottawa, 1992.

[Fer89] J. C. Fernandez. Aldebaran: A Tool for Verification of Communicating Systems. Technical report SPECTRE, Laboratoire de Genie Informatique, Institut IMAG, 1989.

[Fer90] J. C. Fernandez. An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Science of Computer Programming*, 1990 13: 219-236.

[FGLR91] A. Fantechi, S. Gnesi, M. Leporatti, and G. Ristori. Model Checking For LOTOS. *3rd Workshop on Computer-Aided Verification*, 1991. To appear in *Lecture Notes in Computer Science*.

[FGMRS91] J.C. Fernandez, H. Graravel, L. Mounier, A. Rasse, C. Rodriguez, et J. Sifakis. Une boîte à outils pour la verification de programmes LOTOS. In O. Rafiq (ed.), *CFTP'91, Ingenierie des protocoles,* (Rennes, 1991) 479-500.

[Flo67] R.W. Floyd. Assigning Meanings to Programs. *Mathematical Aspects of Computer Science*, 1967 19:19:32.

[Fran86] N. Francez. *Fairness*. Springer-Verlag, New York, 1986.

[Gar89a] H. Garavel. Compilation et verification de programmes LOTOS. These de Doctorat, Université Joseph Fourier (Grenoble), November 1989.

[GHL88] R. Guillemot, M. Haj-Hussein, and L.Logrippo. Executing Large LOTOS Specifications. In S. Aggarwal and K. Sabnani, (eds.), *Protocol Specification, Testing, and Verification VIII*, (North-Holland, 1988) 399-410.

[GL89] R. Guillemot and L.Logrippo. Derivation of Useful Execution traces from LOTOS Specifications by using an Interpreter. In K.J. Turner, (ed.), *Formal Description Techniques*, (North-Holland, 1989) 311-325.

[GS90] H. Garavel and J. Sifakis. Compilation and Verification of LOTOS Specifications. In L. Logrippo, R.L. Probert, and H. Ural, (eds.), *Proceedings of the 10th International Symposium on Protocol Specification, Testing and Verification* , (North-Holland, June 1990) 359-376.

[HC77] G. E. Hughes and M.J. Creswell. *An Introduction to Modal Logic*. Methuen and Co., 1977.

[HH89] M. Haj-Hussein. ISLA: An Interactive System for LOTOS Applications.Master's Thesis, University of Ottawa, 1989.

[HHLS92] M. Haj-Hussein, L. Logrippo, J. Sincennes. Goal-Oriented Execution for LOTOS. Technical Report, Computer Science Department, University of Ottawa, January 1992. To appear in *FORTE'92*.

[Hoa69] C. A. R. Hoare. An Axiomatic Basis for Computer Programs. In *Communications of the ACM*, (October 1969) 12(10):576-583.

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

[ISO8807] ISO, IS 8807. *Information Processing Systems - Open Systems Interconnection* - LOTOS : A formal Description Technique based on the Temporal Ordering of Observational behaviour, May 1989.

[ISO89] ISO, IS 9074. Estelle-A Formal Description Technique Based on an Extended State Transition Model, 1989.

[Jaou92] R. Jaouani. LOTOS Based Conformance Testing: the Theory and a Tool. Master's Thesis. University of Ottawa, 1992.

[JKP89] B. Jonsson, A. H. Khan, and J. Parrow. Implementing a Model Checking Algorithm by Adapting Existing Automated Tools. *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*. Lecture Notes in Computer Science, (1989) 407:179-188.

[JT79] R.W. Jensen, C. C. Tonies. *Software Engineering*. Prentice-Hall, Engelwood, NJ. 1979.

[Karj92] G. Karjoth, C. Binding, and J.Gustafson. LOEWE: A LOTOS Engineering Workbench. To appear in Computer Networks and ISDN Systems.

[Lamp80] L. Lamport. Sometimes is Sometimes "Not Never"- on the Temporal Logic of Programs. In *Proc. 7th Annu. ACM Symp. on Principles of Programming Languages*, (1980) 174-185.

[LFH92] L. Logrippo, M. Faci, M. Haj-Hussein. An Introduction to LOTOS: Learning by Examples. *Computer Networks and ISDN Systems* 23(5) 1992 325-342.

[LL91] E. Lallemand, Guy Leduc. On LOTOS tools and their usefulness to treat large specifications. OSI 95/ULg/A/11/TR/R/V2, 1991.

[Mil80] R. Milner. A Calculus of Communicating Systems. *Lecture Notes in Computer Science*, (Springer-Verlag, 1980) No. 92.

[MP81b] Z. Manna, A. Pnueli. Verification of Concurrent Programs: the Temporal Framework. *Academic Press*, 215-273, 1981.

[OL82] S.S. Owicki, L. Lamport. Proving Liveness Properties of Concurrent Programs. *ACM Trans. on Programming Languages and Systems*, 4(3) (1982) 455-495.

[PL91] S. Pavón, M. Llsmas. The Testing Functionalities of LOLA. In J. Quemada, J. Mañas, E. Vázquez, (eds.), *Formal Description Techniques III*, (North-Holland, 1991) 559-562.

[Pnue77] A. Pnueli. The temporal Semantics of Concurrent Programs. In *18th Symposium on Foundations of Computer Science*, 1977.

[Pnue81] A. Pnueli. The Temporal Semantics of Concurrent Programs. *Theoretical Computer Science*, (1981) 13:45-60.

[QFP 88] J. Quemada, S. Pavón, and A. Fernández. Transforming LOTOS Specifications with LOLA: The Parametrized Expansion.In K. J. Turner, (ed.), *Proceedings of the 1st International Conference on Formal Description Techniques FORTE'88*, (North-Holland, Amsterdam, September 1988) 45-54.

[QS83] J. P. Queille and J. Sifakis. Fairness and Related Properties in Transition Systems - A Temporal Logic to Deal with Fairness. *Acta Informatica*,19:195-220, 1983.

[Ras90] A. Rasse. CLEO: Diagnostic des Erreurs en XESAR. These de Doctorat, Institut National Polytechnique de Grenoble, June 1990.

[RKKL85] P. Rety, C. Kirchner, H. Kirchner, and P. Lescanne. Narrower: A new Algorithm for Unification and its Application to Logic Programming. In Dijon, (ed.), *Proceedings of the First International Conference on Rewriting Techniques and Applications*, 141-157, 1985.

[Rod88] C. Rodriguez. Specification et Validation de Systemes en XESAR. These de Doctorat, Institut National Polytechnique de Grenoble, May 1988.

[Sif86] Joseph Sifakis. A Response to Amir Pnueli's “Specification and Development of Reactive Systems”. In IFIP, (Dublin, Ireland, 1986) 1183-1187.

[vE89] P. van Eijk. The Design of a simulator tool. In P. van Eijk et al., (eds.), *The Formal Description Technique LOTOS*. (North-Holland, Amsterdam, 1989) 351-390.

[WB90] C. Wu, G. Bochmann. Fairness in LOTOS. Technical report, Université de Montreal, Publication # 769, 1990.

[Wes91] C. H. West. Protocol Validation Principles and Applications. Technical Report, 1991.

Appendix A: Two LOTOS Specifications

A.1 LOTOS Specification of the Datalink Service Provider

```
specification datalink[ get, give, send, receive ]:noexit

library  Boolean endlib                (* Abstract Data Types Part *)

type sequeceNumber is Boolean          (* Type sequence number *)
       sorts seqNum
       opns  0      :                   -> seqNum
             inc    : seqNum            -> seqNum
             equal  : seqNum,seqNum     -> Bool

        eqns  forall x, y : seqNum
                    ofsort seqNum
                            inc(inc(x)) = x;
                    ofsort Bool
                            equal(x,x) = true;
                            equal(0,inc(x)) = false;
                            equal(inc(x),0) = false;
                            equal(inc(x),inc(y)) = equal(x,y)
 endtype

type bitString is Boolean     (* Type of messages is a string of bits *)

        sorts bitString
        opns  empty  :                              ->  bitString
              equal  : bitString,bitString          ->  Bool

        eqns  ofsort Bool forall x : bitString
                            equal(x,x) = true
endtype
```

```
type Frame is Boolean (* Messages exchanged are acknowledgments (ack) *)
        sorts Frame                  (* or  data (info) *)
        opns  info,ack:                    -> Frame
              equal   : Frame,Frame        -> Bool
        eqns   ofsort Bool forall x : Frame
                           equal(x,x) = true;
                           equal(ack,info) = false;
                           equal(info,ack) = false
endtype
            (* Control Part *)
behaviour

hide tout in
   (( transmitter [ get,tout,send,receive] (0)
     |||
      receiver [ give,send,receive] (0)
    )
   |[ tout,send,receive]|
      line [tout,send,receive]
   )

where

 process transmitter [get,tout,send,receive] (seq:seqNum) :noexit :=

    get ?data:bitString
   ;sending [tout,send,receive] (seq,data)
    >> transmitter [get,tout,send,receive] (inc(seq))

     where

    process sending[tout,send,receive](seq:seqNum,data:bitString:exit :=
            send !info !seq !data
            ;  ( receive !ack !inc(seq) !empty ; exit
                 []
                 tout
               ; sending [tout,send,receive] (seq,data)

             )
      endproc
endproc

process receiver [give,send,receive] (exp:seqNum) : noexit :=

    receive !info ?rec:seqNum ?data:bitString
   ; ( [rec = exp ] ->
          give !data
         ;send !ack !inc(rec) !empty
         ; receiver [give,send,receive] (inc(exp))

    []  [inc(rec) = exp] ->
          send !ack !inc(rec) !empty
         ; receiver[give,send,receive] (exp)
     )
 endproc
```

```
process line [tout,send,receive] : noexit  :=

     send ?f:Frame ?seq:seqNum ?data:bitString
    ; (  receive !f !seq !data
         ;line [tout,send,receive]
        []
          i
          ; tout
          ; line[tout,send,receive]
       )
endproc

endspec
```

A.2 LOTOS Specification of the Transport Service Handler

```
(* TS-Handler *)

 specification Handler[ConReq,ConInd,ConRes,ConCnf,
                       DatReq,DatInd,DisReq,DisInd]:noexit

 behaviour

    Handler[ConReq,ConInd,ConRes,ConCnf,
          DatReq,DatInd,DisReq,DisInd]

(*-------------------------------------------------------------*)

  where process Handler[ConReq,ConInd,ConRes,ConCnf,
                        DatReq,DatInd,DisReq,DisInd] :noexit:=

        Connection_phase[ConReq,ConInd,ConRes,ConCnf,DisReq,DisInd]
       >>
        ( Data_phase[DatReq,DatInd]
          [>
          Termination_phase[DisReq,DisInd] )
       >>
        Handler[ConReq,ConInd,ConRes,ConCnf,
              DatReq,DatInd,DisReq,DisInd]

(*-------------------------------------------------------------*)

     where process Connection_phase[CRq,CI,CR,CC,DR,DI] :exit:=

             i ; Calling[CRq,CI,CR,CC,DR,DI]
             []
             Called[CRq,CI,CR,CC,DR,DI]


            where process Calling[CRq,CI,CR,CC,DR,DI] :exit:=
                   CRq ;
              (CC ; exit
                   []
                   DI ; Connection_phase[CRq,CI,CR,CC,DR,DI])
                  endproc
                  process Called[CRq,CI,CR,CC,DR,DI] :exit:=
                  CI ;
                    (i ; CR ;exit
                  []
                    i ; DR ;Connection_phase[CRq,CI,CR,CC,DR,DI])
                  endproc

            endproc

(*-------------------------------------------------------------*)
```

```
        process Data_phase[DtR,DtI]:noexit:=
                (i ; DtR ;Data_phase[DtR,DtI]
               []
                DtI ; Data_phase[DtR,DtI])
               endproc

(*-------------------------------------------------------------*)

        process Termination_phase[DR,DI]:exit:=
                (i ;DR ;exit
               []
                DI ;exit)
                endproc

    endproc (* handler *)

endspec
```

A.3 Symbolic Trees Generated by SELA

A.3.1 Datalink Service Provider

```
bh0 * 1 get ?bitString@1:bitString [55]
bh1 * | 1 send !info !0 !bitString@1 [62,89]
bh2 * | | 1 i (specified explicitly) [93]
bh3 * | | | 1 i (hiding: tout) [65,94]
bh4 * | | | | 1 send !info !0 !bitString@1 [62,89] ==> again bh2
    * | | 2 receive !ack !inc(0) !empty
                ([ack=info] and [inc(0)=0] and [empty=bitString@1] ) [63,90]
bh5 * | | | 1 i (enable: exit) [63]
bh6 * | | | | 1 get ?bitString@2:bitString [55] ==> again bh1
    * | | 3 receive !info !0 !bitString@1 [74,90]
bh7 * | | | 1 [0=0] give !bitString@1 [76]
bh8 * | | | | 1 send !ack !inc(0) !empty [77,89]
bh9 * | | | | | 1 i (specified explicitly) [93]
bh10* | | | | | | 1 i (hiding: tout) [65,94]
bh11* | | | | | | | 1 send !info !0 !bitString@1 [62,89] ==> again bh9
    * | | | | | 2 receive !ack !inc(0) !empty [63,90]
bh12* | | | | | | 1 i (enable: exit) [63]
bh13* | | | | | | | 1 get ?bitString@3:bitString [55]
bh14* | | | | | | | | 1 send !info !inc(0) !bitString@3 [62,89] ==> again bh9
    * | | | | | 3 receive !info !inc(0) !empty
                    [info=ack]  [74,90] ==> again bh7
    * | | | 2 [inc(0)=0] send !ack !inc(0) !empty [81,89]
bh15* | | | | 1 i (specified explicitly) [93]
bh16* | | | | | 1 i (hiding: tout) [65,94]
bh17* | | | | | | 1 send !info !0 !bitString@1 [62,89] ==> again bh15
    * | | | | 2 receive !ack !inc(0) !empty [63,90]
bh18* | | | | | 1 i (enable: exit) [63]
bh19* | | | | | | 1 get ?bitString@3:bitString [55]
bh20* | | | | | | | 1 send !info !inc(0) !bitString@3 [62,89] ==> again bh15
    * | | | | 3 receive !info !inc(0) !empty
                  [info=ack]  [74,90] ==> again bh7
```

A.3.2 Transport Service Handler

```
bh0 * 1 i (specified explicitly) [29]
bh1 *   1 ConReq [35]
bh2 *     1 ConCnf [36]
bh3 *       1 i (enable: exit) [36]
bh4 *         1 i (specified explicitly) [52]
bh5 *           1 DatReq [52] ==> again bh4
    *           2 i (specified explicitly) [60]
bh6 *             1 DisReq [60]
bh7 *               1 i (enable: exit) [60] ==> again bh0
    *           3 DisInd [62] ==> again bh7
    *         2 DatInd [54] ==> again bh4
    *         3 i (specified explicitly) [60] ==> again bh6
    *         4 DisInd [62] ==> again bh7
    *     2 DisInd [38]
bh8 *       1 i (specified explicitly) [29] ==> again bh1
    *       2 ConInd [42]
bh9 *         1 i (specified explicitly) [43]
bh10*         1 ConRes [43] ==> again bh3
    *         2 i (specified explicitly) [45]
bh11*           1 DisReq [45] ==> again bh8
    * 2 ConInd [42] ==> again bh9
```

Appendix B: Examples using LMC

The following table shows the temporal logic operators used by LMC and their corresponding operators used for the implementation:

Original Operators	Implementaion Operators
AG	ag
EG	eg
AX	ax
EX	ex
AF	af
EF	ef
A[f_1 υ f_2]	all[f_1 until f_2]
E[f_1 υ f_2]	some[f_1 until f_2]
⇒	->
∨	or
∧	&
¬	~

The following session demonstrates the use of LMC to evaluate the properties related to the case studies that were described in chapter 6.

```
                    LMC
                MODEL CHECKER
                 FOR LOTOS

/* Main menu of LMC */
-------------------------------------------------------------
-------------------------------------------------------------
<1> (l)    List LOTOS specifications '.l'
<2> (p)    List Prolog files  '.pl'
<3> (d)    List directory  '*.*'
<4> (v)    Verify  Temporal Logic Formulae
<5> (vi)   vi editor
<6> (pwd)  pwd (current directory)
<7> (u)    Other unix options
<8> (q)    Exit  LMC?
-------------------------------------------------------------
-------------------------------------------------------------


enter <command> or m[enu] to return to main:
==> v

% compiling file
/tmp_mnt/home/prga/usr7/grad/bghribi/newidea/brahim_tnt.pl
% brahim_tnt.pl compiled in module user, 1.334 sec 6,676 bytes


 Press (h) for help or <RETURN> to proceed:

 Enter the file name or <RETURN> to cancel
 ==> datalink
/* Name of the specification */

% compiling file
/tmp_mnt/home/prga/usr7/grad/bghribi/newidea/datalink.pl
% datalink.pl compiled in module user, 2.217 sec 9,932 bytes

 Enter the tree file name or <RETURN> to cancel
 ==> datalinkdata   /* Behaviour tree file generated by SELA */
% compiling file
/tmp_mnt/home/prga/usr7/grad/bghribi/newidea/datalinkdata.pl
% datalinkdata.pl compiled in module user, 1.634 sec 19,600 bytes


 Enter Formula    :

     > ag(~deadlock).

     Checking Formula...

*********************************************************
     FORMULA  HOLDS
*********************************************************

   Press (y) for more Formulae or <RETURN> to  quit to Main menu: y
```

```
Enter Formula    :

     > ag( get!@1 -> ef(give!@1)).

     Checking Formula...

*********************************************************
     FORMULA  HOLDS
*********************************************************

   Press (y) for more Formulae or <RETURN> to  quit to Main menu: y


 Enter Formula    :

     > ag(get!* -> ef(give!*)).

     Checking Formula...

*********************************************************
     FORMULA  HOLDS
*********************************************************

   Press (y) for more Formulae or <RETURN> to  quit to Main menu: y


 Enter Formula    :

     > ag( receive!*!*!* -> ax(all(~ receive!*!*!* until send!*!*!* ))).

     Checking Formula...

*********************************************************
     FORMULA  HOLDS
*********************************************************

   Press (y) for more Formulae or <RETURN> to  quit to Main menu: y


 Enter Formula    :

     > ag(send!ack!0!empty -> af(receive!info!inc(0)!* &
                 ax(send!ack!0!empty) or (receive!info!0!*))).

     Checking Formula...

*********************************************************
     FORMULA  HOLDS
*********************************************************

   Press (y) for more Formulae or <RETURN> to  quit to Main menu: y


 Enter Formula    :
```

```
     > ag( send!info!*!* -> ax(all(~ send!info!*!* until
                                        receive!info!*!* ))).

     Checking Formula...


*********************************************************
    FORMULA  IS NOT SATISFIED
*********************************************************

   Press (y) for more Formulae or <RETURN> to  quit to Main menu:


                      LMC
                  MODEL CHECKER
                   FOR LOTOS

-------------------------------------------------------------
-------------------------------------------------------------
<1> (l)    List LOTOS specifications '.l'
<2> (p)    List Prolog files  '.pl'
<3> (d)    List directory  '*.*'
<4> (v)    Verify  Temporal Logic Formulae
<5> (vi)   vi editor
<6> (pwd)  pwd (current directory)
<7> (u)    Other unix options
<8> (q)    Exit  LMC?
-------------------------------------------------------------
-------------------------------------------------------------

enter <command> or m[enu] to return to main:
==> v

% compiling file
/tmp_mnt/home/prga/usr7/grad/bghribi/newidea/brahim_tnt.pl
% brahim_tnt.pl compiled in module user, 1.350 sec 1,532 bytes


 Press (h) for help or <RETURN> to proceed:

 Enter the file name or <RETURN> to cancel
 ==> ts_handler                      /* Transport service handler */

% compiling file
/tmp_mnt/home/prga/usr7/grad/bghribi/newidea/ts_handler.pl
% ts_handler.pl compiled in module user, 1.234 sec 1,820 bytes

 Enter the tree file name or <RETURN> to cancel
 ==> ts_handlerdata
            /* Behaviour tree  generated by SELA */

% compiling file
/tmp_mnt/home/prga/usr7/grad/bghribi/newidea/ts_handlerdata.pl
% ts_handlerdata.pl compiled in module user, 0.634 sec 7,324 bytes
```

```
Enter Formula    :

     > ag(~deadlock).

     Checking Formula...

*********************************************************
     FORMULA  HOLDS
*********************************************************

   Press (y) for more Formulae or <RETURN> to  quit to Main menu: y


 Enter Formula    :

     > ag('ConReq' -> af('ConCnf' or 'DisInd') ).

     Checking Formula...

*********************************************************
     FORMULA  HOLDS
*********************************************************

   Press (y) for more Formulae or <RETURN> to  quit to Main menu: y


 Enter Formula    :

     > ag('ConInd'  -> af('ConRes' or  'DisReq')).

     Checking Formula...

*********************************************************
     FORMULA  HOLDS
*********************************************************

   Press (y) for more Formulae or <RETURN> to  quit to Main menu: y


 Enter Formula    :

     > ag('ConInd' -> af('ConRes') ).

     Checking Formula...


*********************************************************
    FORMULA  IS NOT SATISFIED
*********************************************************

   Press (y) for more Formulae or <RETURN> to  quit to Main menu: y


 Enter Formula    :

     > ag('ConReq' -> some(~('ConCnf')  until  'DatReq' ) ).
```

```
     Checking Formula...

**********************************************************
     FORMULA  HOLDS
**********************************************************

   Press (y) for more Formulae or <RETURN> to  quit to Main menu:


                      LMC
                  MODEL CHECKER
                   FOR LOTOS

------------------------------------------------------------
------------------------------------------------------------
<1> (l)    List LOTOS specifications '.l'
<2> (p)    List Prolog files  '.pl'
<3> (d)    List directory  '*.*'
<4> (v)    Verify  Temporal Logic Formulae
<5> (vi)   vi editor
<6> (pwd)  pwd (current directory)
<7> (u)    Other unix options
<8> (q)    Exit  LMC?
------------------------------------------------------------
------------------------------------------------------------

enter <command> or m[enu] to return to main:
==> q
/* End of the session */
```

Appendix C: Diagnostics

This example illustrates the use of LMC and of ISLA for diagnostics. We consider the formula:

```
ag((send!info!*!*) -> ax(all(~(send!info!*!*) until (receive!info!*!*)))).
```

This formula was described in chapter 6 for example 1 (Datalink Service Provider). As expected, the formula is false because it states that whenever a message of type info is sent, then no message of type info is sent until the receiver receives a message of type info. However, we know that in the case of a timeout, the message of type info is retransmitted which allows the possibility of having a message of type info sent twice before any receive happens.

```
/* Using the Model Checker*/
---------------------------------

ENTER TEMPORAL LOGIC FORMULA    :

   >  ag((send!info!*!*) -> ax(all(~(send!info!*!*) until (receive!info!*!*)))).

     CHECKING FORMULA ...


**********************************************************
   FORMULA  IS NOT SATISFIED
**********************************************************
```

The file Traversal contains the list of states visited during the model checking procedure. It shows whether synchronization occured between the actions of the state and of the formula. For example, in this case, at state *1*, *send!info!*!** failed to synchronize with the label of the state. At state *2*, synchronization was possible and we can see that following the sequence 2, 5, two successive (*send!info!*!**) were encountered and no intermediate *recieve* was detected. This explains why the formula did not hold.

/* File Traversal */
---

checking(failed,1,send!info!*!*).
checking(success,2,send!info!*!*).
checking(failed,5,receive!info!*!*).
checking(success,5,send!info!*!*).

The following is an execution of the specification using ISLA:

/* Using ISLA for Diagnostics */

==========
No Internal actions

<1>- get ?data:bitString ---> bh1 [55] **/* The only possible action at the moment */**

==========
==> 1
Enter a value for data:bitString
=> empty **/* We choose for example an empty string */**

==========
No Internal actions

<1>- send !info:Frame !0:seqNum !empty:bitString ---> bh1 [62,89] **/* our empty string is sent */**

==========
==> 1
Passed evaluated value ==> info **/* The passed parameters */**

Passed evaluated value ==> 0

Passed evaluated value ==> empty

==========
No Internal actions

<1>- i (specified explicitly) ---> bh1 [93]
<2>- receive !info:Frame !0:seqNum !empty:bitString ---> bh2 [74,90]

==========
==> 1
Internal event is executed

==========
No Internal actions

```
  <1>- i (hiding: tout)  ---> bh1   [65,94]        /* Timeout occurs */

================================================================
==> 1
   Internal event is executed

================================================================
No Internal actions

  <1>- send !info:Frame !0:seqNum !empty:bitString  ---> bh1   [62,89]

================================================================
```

/* Another send is detected before a receive is encountered, therefore we reached our counter example */

Notice that at some point, there were two possible actions to choose from (indicated in bold). We avoided the *receive* action because we wanted to detect a path in which no *receive* is encountered after the first *send* until we meet a second *send*. It is obvious that the other action is the only alternative. By doing so, we were able to reach a state in which a new *send* is performed without an intermediate *receive*.