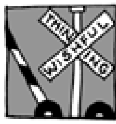
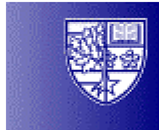



Heriot-Watt University
Edinburgh
Department of Computing & Electrical Engineering

**A simplified model of**
**"The Absolute Block System of Railway Signaling"**

# Distributed Systems Programming
## LOTOS Project

**Submitted To:** Dr. Andrew Ireland
By, Fotios Bassayiannis, DMIS

**Friday, 12 November, 1999**

# CONTENTS

Page 3 : Introduction

Page 6 : Assumptions about the specification requirements

Page 7 : Diagrammatic presentation of the specification.

Page 8 : Appendix (BOX bell's user guide)

Page 9 : LOTOS ABS specification source code

Page 16: LOLA Transcript: Safe passage of a train through my ABS model

Page 25: Unsafe System Source Code

Page 27: LOLA Transcript: Demonstration of not safe basic BLK_SEC system

Page 30: Conclusion

# INTRODUCTION

It is always a challenge to work on formal specifications of systems, but once the systems become distributed and concurrent the difficulty increases to levels that create a not previously experienced level of cognitive overload. The good news here is that with the help of robust specification languages like LOTOS and useful tools like LOLA the human mind can in fact overcome these difficulties and create effective specifications of such systems. This is what I hope I have done to some degree in this paper.

The difficulty of this assignment does not only owe to the inherent intricacies and complexities of a distributed and concurrent system but also to the need for the ABS system to be specified in a way that supports modularity and subsequently easy extension. In order to achieve this modularity I have tried to think of the BOX process as communicating with another instance of itself. I believe that the system I am presenting here is modular, but I cannot guarantee that since I have not thoroughly tested a multi-box version of my specification on LOLA.

As it was asked in the assignment, the specification contains 3 process definitions:

- The given BLK_SEC process
- The SIGNAL process
- The BOX process

The last two are briefly described below.

### The *BOX* process

```
PROCESS BOX[TrackSide,
            enterBS,
            ForInst,    AccInst,
            AdBellSend, AdBellGet,
            ReBellSend, ReBellGet]
            (BoxMode         : mode,
             BoxOrientation : orientation,
             Status          : ins)

            : NOEXIT :=
```

Process *BOX* is declared with 8 gates and 3 parameters.

The **TrackSide** gate is used by the BOX for communicating with the SIGNAL process and passing through it the value for the track side signal.

The **enterBS** gate is used for symbolic synchronization with the enterBS action offered by the SIGNAL and BLK_SEC processes when the train enters the block section right after the box. This is done to ensure that only after the train has passed to the next block section, the track side signal is returned to *danger* and to ensure synchronization of the whole behaviour expression on this important event which is actually the essence of our specification (the train passage). Also, this synchronization event ensures atomicity of the action sequence: *exitBS1; enterBS2*, which is not necessary for the specification to be semantically correct, but it is necessary for the specification to be pragmatically correct since in the real world these two action actually coincide. In the formal and abstract environment of LOTOS however they can easily exist in separation as different actions.

The **ForInst** gate is used for receiving the value for the BOX's forwarding instrument from the BOX in advance of it.

The **AccInst** gate represents the BOX's accepting instrument and is used for sending the value for the forwarding instrument from our BOX to the BOX in the rear.

The **AdBellGet** and **AdBellSend** gates are used for getting and sending bell signals respectively, from and to the advance BOX's bells.

The **ReBellGet** and **ReBellSend** gates are used for getting and sending bell signals from and to the rear BOX's bell.

The BOX's three parameters are used to control the initial state of a newly instantiated BOX process. The **BoxMode** parameter is used to determine whether the BOX will be in *accepting* or *forwarding* mode, i.e. whether the BOX will work in order to receive a train in the BLK_SEC it controls or send forward a train that already is in the BLK_SEC it controls.

The **BoxOrientation** parameter is not used in this specification, but is nevertheless incorporated in the BOX process definition because it makes it easy for a LOTOS specifier to add new BOX instances in the specification without having to explicitly do the tedious task of gate relabelling but by instead only instantiating a BOX process with **BoxMode** set to *reverse.*

The **Status** parameter simply records the BOX process' forwarding instrument's status so that it retains its value between exits of the process.

**The *SIGNAL* process:**

```
PROCESS SIGNAL[exitBS, enterBS,
               TrackSide]        : NOEXIT
```

This process provides the gateway between two consecutive block sections. It communicates with the BOX process through the **TrackSide** gate through which it receives the track side signal value and synchronizes accordingly with the two BLK_SEC instances it connects through the **exitBS** and **enterBS** gates.

**The specification's BEHAVIOUR expression:**

```
BEHAVIOUR

  HIDE TrackSide IN
  (

    BLK_SEC[enterBS1, exitBS1] (0)

    |[exitBS1]|


        SIGNAL[exitBS1, enterBS2, TrackSide]

        |[TrackSide, enterBS2]|

        BOX[TrackSide,
            enterBS2,
            ForInst,    AccInst,
            AdBellSend, AdBellGet,
            ReBellSend, ReBellGet]
            (accepting, normal, line_blocked)


    |[enterBS2]|

    BLK_SEC[enterBS2, exitBS2] (0)

  )
```

As we can see the **TrackSide** gate is hidden from the environment as it was asked in the assignment.

Numbered gates are used so that proper synchronisations can be enforced (basically vis distinguishing different enterBS and exitBS type of actions by their numbers, e.g enterBS1 is different than enterBS2) in a multi-process and potentially very long behaviour expression. Numbered gates are also very useful for testing in LOLA since the tester can immediately know which process offers a certain action he/she sees in the LOLA menu of offered actions.

The SIGNAL process synchronises with the first BLK_SEC on **exitBS1** and with BOX and the second BLK_SEC on **enterBS2** . Of course the **enterBS1** action is always offered to the environment and the LOLA tester is responsible for picking it up on the right moment so that the ABS protocol is followed correctly in the simulation. (The tester actually plays the role of the BOXes in front and in the rear of the BOX in our specification as well as the role of the operator of our BOX - the BOX in the middle).

Note that the BOX is initialised in *accepting* mode with Status set to *line_blocked* and both BLK_SECs are initialised with 0 trains in them.

# ASSUMPTIONS

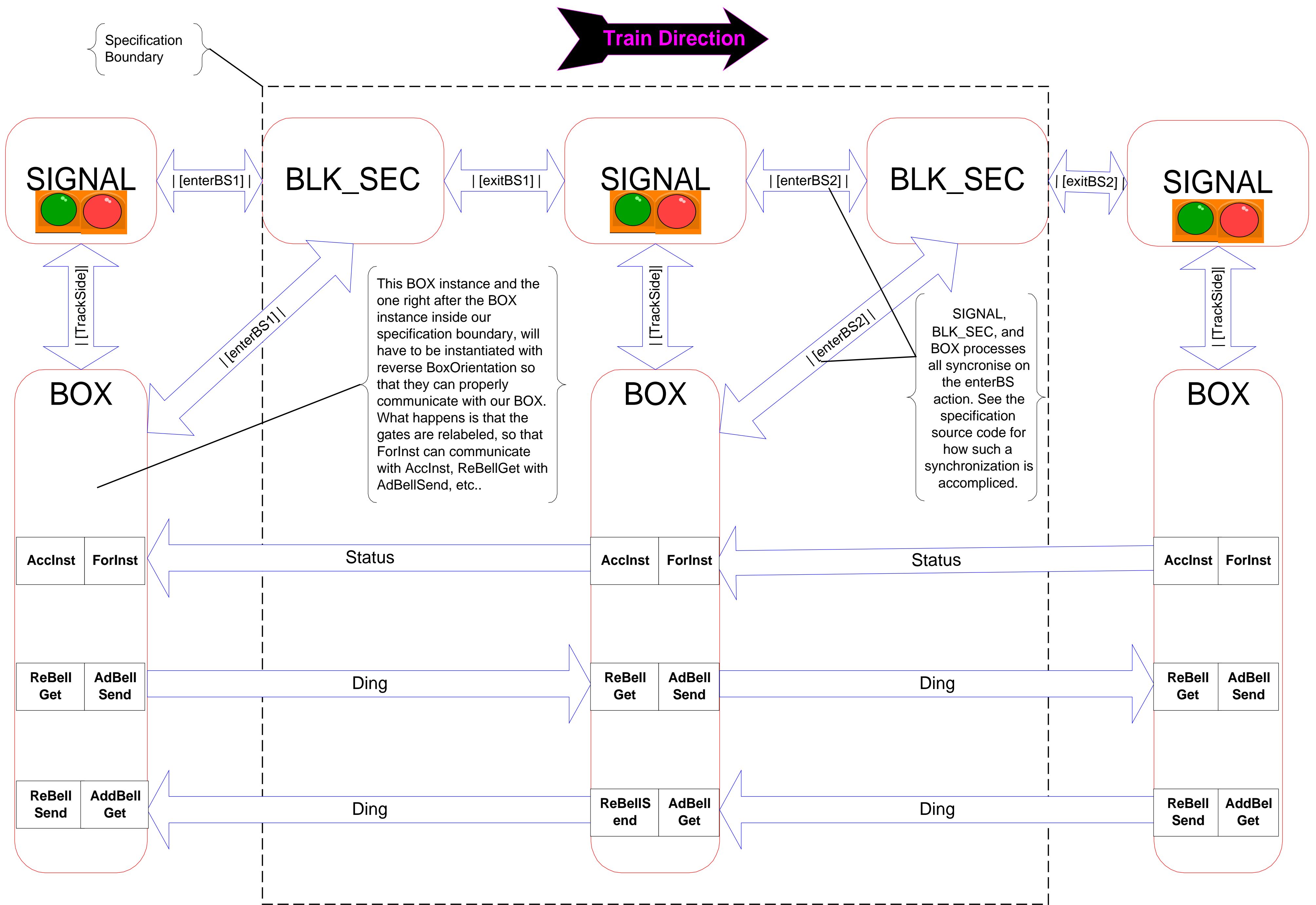
## On the specification requirements

- The box operators must follow the protocol (i.e. They must not try to trick the system into undefined states).

- There are BOXes both in front and in the rear of the BOX modelled in our specification.

- The trains must stop in front of a "danger" signal and move on when the signal becomes "clear".

- We assume that initially the specification's BOX's **BoxStatus** is *line_blocked*, it's **BoxMode** is *accepting* and it's **BoxOrientation** is *normal*. Also, we initialise both BLK_SECs with 0 train counts (i.e. they both do not contain any trains). We could easily and without problems start with a different initial condition; as long as this condition is consistent with the ABS protocol, i.e. a BOX cannot be initialised to accepting mode if it's BLK_SEC (the block that the box has control over) is initialised to a train count of inc(0).


Specification Boundary
Train Direction
SIGNAL
| [enterBS1] |
BLK_SEC
| [exitBS1] |
SIGNAL
| [enterBS2] |
BLK_SEC
| [exitBS2] |
SIGNAL
| [TrackSide] |
| [enterBS1] |
This BOX instance and the one right after the BOX instance inside our specification boundary, will have to be instantiated with reverse BoxOrientation so that they can properly communicate with our BOX. What happens is that the gates are relabeled, so that ForInst can communicate with AccInst, ReBellGet with AdBellSend, etc..
| [TrackSide] |
| [enterBS2] |
SIGNAL, BLK_SEC, and BOX processes all syncronise on the enterBS action. See the specification source code for how such a synchronization is accompliced.
| [TrackSide] |
BOX
BOX
BOX
AccInst
ForInst
Status
AccInst
ForInst
Status
AccInst
ForInst
ReBell Get
AdBell Send
Ding
ReBell Get
AdBell Send
Ding
ReBell Get
AdBell Send
ReBell Send
AddBell Get
Ding
ReBellS end
AdBell Get
Ding
ReBell Send
AddBel Get
Legend:
:Symbolic Synchronization
:Value Passing

# Appendix

## ABS Bell Operator's Manual

**(Important: Please keep this page with you while testing the specification in LOLA)**

This page provides bell signal names used in the specification in a convenient tabular format. The long bell signal names have been shrinked to convenient acronyms so that the tester can easily and quickly input them while testing the specification in LOLA.

| *Bell Signal* | *Meaning* |
|---|---|
| ***call*** | call attention |
| ***ilc*** | is line clear for a train |
| ***tin*** | train entering section |
| ***tout*** | train out of section |

**Signal Name Conventions**

# ABS Source Code

```
(*************************************************************************)
(* The following specification represents an "Absolute Block System"      *)
(* of Railway Signalling simplified into a single one-way train track     *)
(*************************************************************************)


SPECIFICATION SINGLE_TRACK_ABS[enterBS1, exitBS1,
                               enterBS2, exitBS2,
                               TrackSide,
                               ForInst,    AccInst,
                               AdBellSend, AdBellGet,
                               ReBellSend, ReBellGet] : NOEXIT

(*************************************************************************)


(****************************IMPORT LIBRARIES*****************************)

(*Import standard data types from available libraries*)

LIBRARY Boolean ENDLIB

(*************************************************************************)



(****************************TYPE DEFINITIONS*****************************)

(*=======================================================================*)
(*The Count data type is used to count the number                        *)
(*of trains in a block section.                                          *)

TYPE Count IS Boolean
SORTS
  count

OPNS
  0        :               -> count
  inc, dec : count         -> count
  _eq_     : count, count  -> Bool
  noi, nod : count         -> Bool

EQNS FORALL x,y : count
  OFSORT count
    inc(dec(x)) = x;
    dec(inc(x)) = x;

  OFSORT Bool
    nod(0) = true; nod(inc(x)) = nod(x);
    noi(0) = true; noi(dec(x)) = noi(x);

    0 eq 0 = true;

    inc(x) eq y = x eq dec(y);
    dec(x) eq y = x eq inc(y);

    nod(x) => 0 eq inc(x) = false;
    noi(x) => 0 eq dec(x) = false;

ENDTYPE
(*=======================================================================*)
```

```
(*=====================================================================*)
(*The Box can be in either forwarding or accepting mode. These two     *)
(*possible modes of box operation are modelled by this ADT.            *)

TYPE Box_Mode IS Boolean
  SORTS mode

  OPNS
    accepting  :              -> mode
    forwarding :              -> mode
    _ eq _     : mode , mode -> Bool

  EQNS FORALL x: mode
    OFSORT Bool
      x          eq x          = true;
      accepting  eq forwarding = false;
      forwarding eq accepting  = false;

ENDTYPE
(*=====================================================================*)

(*=====================================================================*)
(*The Box process can be instantiated as a normal or a reverse instance.  *)
(*Instantiating the Box process as reverse does the gate relabelling     *)
(*needed for communicating with itself(reversing the gates) automatically *)
(*for the user. This is a feature only used if the specification is      *)
(*extended to contain more than one boxes                                *)

TYPE Box_Orientation IS Boolean
  SORTS orientation

  OPNS normal  :                          -> orientation
       reverse :                          -> orientation
       _eq_    : orientation, orientation -> Bool

  EQNS FORALL x: orientation
    OFSORT Bool
      x       eq x       = true;
      normal  eq reverse = false;
      reverse eq normal  = false;

ENDTYPE
(*=====================================================================*)

(*=====================================================================*)
TYPE Track_Side_Signal IS Boolean
  SORTS tss

  OPNS  clear, danger :          -> tss
        _ eq _        : tss, tss -> Bool

  EQNS FORALL x: tss
    OFSORT Bool
      x      eq x      = true;
      danger eq clear  = false;
      clear  eq danger = false;

ENDTYPE
(*=====================================================================*)
```

```
(*======================================================================*)
TYPE Bell_Signal IS Boolean
  SORTS bs

  OPNS  call   :        -> bs   (*call attention*)
        ilc    :        -> bs   (*is line clear for a train*)
        tin    :        -> bs   (*train entering section*)
        tout   :        -> bs   (*train out of section*)
        _ eq _ : bs, bs -> Bool

  EQNS FORALL x: bs
    OFSORT Bool
      x    eq x    = true;
      call eq ilc  = false;
      call eq tin  = false;
      call eq tout = false;
      ilc  eq call = false;
      ilc  eq tin  = false;
      ilc  eq tout = false;
      tin  eq call = false;
      tin  eq ilc  = false;
      tin  eq tout = false;
      tout eq ilc  = false;
      tout eq call = false;
      tout eq tin  = false;

ENDTYPE
(*======================================================================*)

(*======================================================================*)
TYPE Instrument_Signal IS Boolean
  SORTS ins

  OPNS  line_blocked  :           -> ins
       line_clear    :           -> ins
       train_on_line :           -> ins
       _ eq _        : ins, ins -> Bool

  EQNS FORALL x: ins
    OFSORT Bool
      x             eq x             = true;
      line_blocked  eq line_clear    = false;
      line_blocked  eq train_on_line = false;
      line_clear    eq line_blocked  = false;
      line_clear    eq train_on_line = false;
      train_on_line eq line_blocked  = false;
      train_on_line eq line_clear    = false;

ENDTYPE
(*======================================================================*)


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

```
(******************************BEHAVIOUR***********************************)
(**************************************************************************)
(*The behaviour expression is set up in such a way that it makes possible *)
(*the synchronisation of the SIGNAL, BOX, and 2nd BLK_SEC on the enterBS  *)
(*action, which signifies the completion of the train passing. This is    *)
(*implicitly required by the protocol since it makes sure that the        *)
(*track-side signal is returned to "danger" only after the completion     *)
(*of the train passing                                                    *)

BEHAVIOUR

  HIDE TrackSide IN  (*We hide this as we were asked*)
  (
    BLK_SEC[enterBS1, exitBS1] (0)

    |[exitBS1]|


      SIGNAL[exitBS1, enterBS2, TrackSide]

      |[TrackSide, enterBS2]|

      BOX[TrackSide,
          enterBS2,
          ForInst,    AccInst,
          AdBellSend, AdBellGet,
          ReBellSend, ReBellGet]
          (accepting, normal, line_blocked)

                 (*We instantiate this box as an accepting one   *)
                 (*with the forwarding instument status set      *)
                 (*to "line blocked",as we have chosen to start  *)
                 (*with a system completely devoid of trains.    *)
                 (*We could choose otherwise with no problem!    *)

      |[enterBS2]|

    BLK_SEC[enterBS2, exitBS2] (0)

  )

(**************************************************************************)
(**************************************************************************)


WHERE
```

```
(**********************PROCESS DEFINITIONS*******************************)

(*======================================================================*)
PROCESS SIGNAL[exitBS, enterBS, TrackSide] : NOEXIT :=

      TrackSide ? Signal : tss;

      (
        [Signal eq clear] ->
        (
          exitBS;
          enterBS;
          SIGNAL[exitBS, enterBS, TrackSide]
        )

        []

        [Signal eq danger] -> SIGNAL[exitBS, enterBS, TrackSide]
      )

ENDPROC
(*======================================================================*)

(*======================================================================*)
PROCESS BLK_SEC[enterBS, exitBS](trains:count) : NOEXIT :=

  (
    enterBS;
    BLK_SEC[enterBS, exitBS](inc(trains))
  )

  []

  (
    [not(trains eq 0)] -> exitBS;
                          BLK_SEC[enterBS, exitBS](dec(trains))
  )

ENDPROC
(*======================================================================*)

(*======================================================================*)
PROCESS BOX[TrackSide,
            enterBS,
            ForInst,    AccInst,
            AdBellSend, AdBellGet,
            ReBellSend, ReBellGet]
            (BoxMode        : mode,
             BoxOrientation : orientation,
             Status         : ins)
            : NOEXIT :=
(

  (*This first guarded condition takes care of the case where*)
  (*a box with its gates relabeled to reversed positions     *)
  (*needs to be instantiated. This feature is not used in    *)
  (*present specification                                    *)

  [BoxOrientation eq reverse] ->
  (

    exit(BoxMode, normal, Status)

    >>

    ACCEPT NewMode        : mode,
           NewOrientation : orientation,
           NewStatus      : ins
    IN
    Box[TrackSide,
        enterBS,
        AccInst,   ForInst,
        AdBellGet, AdBellSend,
        ReBellGet, ReBellSend]
        (NewMode, NewOrientation, NewStatus)
  )
```

```
  []

  [ ( BoxMode eq accepting ) and ( not(BoxOrientation eq reverse) ) ] ->
  (
    ReBellGet  ? Ding : bs [Ding eq call];
    ReBellSend ! call;
    ReBellGet  ? Ding : bs [Ding eq ilc];
    ReBellSend ! ilc;

    AccInst ! line_clear;
    ReBellGet  ? Ding : bs [Ding eq tin];
    ReBellSend ! tin;
    AccInst ! train_on_line;

    exit(forwarding, normal, Status) (*exit to forwarding mode*)
  )

  []

  [ ( BoxMode eq forwarding ) and ( not(BoxOrientation eq reverse) ) ] ->
  (
    (*If the box is called in forwarding mode with its forwarding instrument*)
    (*showing "train_on_line", then complete the protocol and re-exit to    *)
    (*forwarding mode with the Box Status being "line_blocked" this time.   *)
    (*This is taken care of by the following condition                      *)

    [Status eq train_on_line] ->
    (
      AdBellGet  ? Ding : bs [Ding eq call];
      AdBellSend ! Call;
      AdBellGet  ? Ding : bs [Ding eq tout];
      ForInst ? Status : ins [Status eq line_blocked];

      exit(forwarding, normal, Status)
    )

    []

    [Status eq line_blocked] ->
    (
      AdBellSend ! call;
      AdBellGet  ? Ding : bs [Ding eq call];
      AdBellSend ! ilc;
      AdBellGet  ? Ding : bs [Ding eq ilc];

      ForInst ? Status : ins [Status eq line_clear];
      TrackSide  ! clear;
      AdBellSend ! tin;
      AdBellGet  ? Ding : bs [Ding eq tin];
      ForInst ? Status : ins [Status eq train_on_line];

      enterBS;

      TrackSide ! danger;
      ReBellSend ! call;
      ReBellGet  ? Ding : bs [Ding eq call];
      ReBellSend ! tout ;

      AccInst ! line_blocked;

      exit(accepting, normal, Status)
      (*After the train is forwarded exit to accepting mode*)
    )
  )
)

>>

ACCEPT NewMode        : mode,
       NewOrientation : orientation,
       NewStatus      : ins
IN
```

```
Box[TrackSide,
    enterBS,
    ForInst,    AccInst,
    AdBellSend, AdBellGet,
    ReBellSend, ReBellGet]
    (NewMode, NewOrientation, NewStatus)

ENDPROC
(*========================================================================*)

(**************************************************************************)

ENDSPEC
```

# LOLA Transcript: Safe passage of a train through my ABS model

```
$ lola nabs8.lot -l is

 LOLA. Version 3.6 1995.
 Departamento de Ingenieria Telematica. ETSIT.
 Universidad Politecnica de Madrid.

load

  Loading specification from nabs8.
  Loading library from is.

TOPO_3R6 (Mon Jan 23 15:19:15 MET 1995) /u1/staff/air/topo
TOPO_3R6 (Mon Jan 23 15:19:15 MET 1995) /u1/staff/air/topo
`nabs8.lsf' is up to date.

  Restoring nabs8.lsf.

lola> s
step

 Rewriting expressions in the specification.
 Rewriting done.
 Analysing unguarded conditions.
 Analysis done.


 [  1]  enterbs1;
 [  2]  rebellget  ? ding_45:bs [ding_45 eq call = true];

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  Enter expressions for the variable definitions (CR for none):
   ding_45:bs = call

  ==> rebellget  ! call;

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


 [  1]  enterbs1;
 [  2]  rebellsend  ! call;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> rebellsend  ! call;

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


 [  1]  enterbs1;
 [  2]  rebellget  ? ding_49:bs [ding_49 eq ilc = true];

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  Enter expressions for the variable definitions (CR for none):
   ding_49:bs = ilc

  ==> rebellget  ! ilc;

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


 [  1]  enterbs1;
 [  2]  rebellsend  ! ilc;
```

The model specification begins by offering the enterBS1 action (offered by the first BLK_SEC) ,which due to the way BLK_SEC is defined will keep being offered in every single step of the simulation regardless of the train count. This is not a problem since we assume that there is another BOX and SIGNAL process combo in front of it that will prevent the enterBS action from being picked up at inappropriate times. In event 2 the BOX process, since it is in accepting mode, it listens for a call_attention signal from the bell of the BOX in the rear of it.

Here we assume that the forwarding instrument of the BOX in the rear displays line_blocked. This is why the BOX in the rear sends ilc. If the rear BOX's status was train_on_line then it would wait for the train out signal from our BOX and the subsequent setting of its forwarding instrument status (which is also the BOX status in my spec) to line_blocked.

```
<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> rebellsend  ! ilc;

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


 [  1]  enterbs1;
 [  2]  accinst  ! line_clear;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> accinst  ! line_clear;
```

Here our BOX sets the forwarding instrument of the BOX in the rear to line_clear.

```
----------------------------------------------------------------------------


 [  1]  enterbs1;
 [  2]  rebellget  ? ding_50:bs [ding_50 eq tin = true];

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  Enter expressions for the variable definitions (CR for none):
   ding_50:bs = tin

  ==> rebellget  ! tin;

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


 [  1]  enterbs1;
 [  2]  rebellsend  ! tin;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> rebellsend  ! tin;

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


 [  1]  enterbs1;
 [  2]  accinst  ! train_on_line;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> accinst  ! train_on_line;

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


 [  1]  enterbs1;
 [  2]  i; (* exit (forwarding, normal, line_blocked) *)

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 1

  ==> enterbs1;
```

This is where it is our responsibility as LOLA users to pick up the offered enterBS1 action in order to follow the protocol. We assume that there is an exitBS type of action somewhere behind the rear BOX and then it is the turn for the enterBS1 action.

```
----------------------------------------------------------------------------


 [  1]  enterbs1;
 [  2]  i; (* exit (forwarding, normal, line_blocked) *)

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> s 1

 Event 1:
```

```
blk_sec [enterbs1,exitbs1] (0) (* line 192 *)
blk_sec [enterbs,exitbs] (inc(0)) (* line 256 *)
 enterbs1 (* line 255 *);

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> m

 [  1]  enterbs1;
 [  2]  i; (* exit (forwarding, normal, line_blocked) *)

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> i; (* exit (forwarding, normal, line_blocked) *)
```

Here we can see that the train count in the first BLK_SEC is inc(0) as expected after the enterBS1 action.

The BOX process exits to forwarding mode with Status line_blocked (the status of our box has not changed since the BOX in front of us has not modified our forwarding instrument's status.

```
-------------------------------------------------------------------


 [  1]  enterbs1;
 [  2]  adbellsend  ! call;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> adbellsend  ! call;

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


 [  1]  enterbs1;
 [  2]  adbellget  ? ding_54:bs [ding_54 eq call = true];

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  Enter expressions for the variable definitions (CR for none):
   ding_54:bs = call

  ==> adbellget  ! call;

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


 [  1]  enterbs1;
 [  2]  adbellsend  ! ilc;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> adbellsend  ! ilc;

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


 [  1]  enterbs1;
 [  2]  adbellget  ? ding_55:bs [ding_55 eq ilc = true];

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  Enter expressions for the variable definitions (CR for none):
   ding_55:bs = ilc

  ==> adbellget  ! ilc;

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


 [  1]  enterbs1;
 [  2]  forinst  ? status_56:ins [status_56 eq line_clear = true];

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  Enter expressions for the variable definitions (CR for none):
```

Our forwarding instrument is listening for the line_clear signal from the advance BOX's accepting instrument.

```
  status_56:ins = line_clear

 ==> forinst  ! line_clear;

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


 [  1]  enterbs1;
 [  2]  i; (* trackside ! clear *)

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

 ==> i; (* trackside ! clear *)
```

After the Status becomes line_clear our BOX sets it's corresponding SIGNAL process' track side signal value to clear.

```
--------------------------------------------------------------------------


 [  1]  enterbs1;
 [  2]  exitbs1;
 [  3]  adbellsend  ! tin;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> s 2

 Events involved in the Synchronization 2:

   blk_sec [enterbs1,exitbs1] (0) (* line 192 *)
   blk_sec [enterbs,exitbs] (inc(0)) (* line 256 *)
    exitbs1 (* line 262 *);
 |[exitbs1]|
   signal [exitbs1,enterbs2,trackside]  (* line 197 *)
    exitbs1 (* line 238 *);

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 3

 ==> adbellsend  ! tin;
```

Right after the track side signal is set to clear the exitBS1 action is offered. Following the protocol it will be picked up after the tin>tin>train_on_line signal sequence.

```
--------------------------------------------------------------------------


 [  1]  enterbs1;
 [  2]  exitbs1;
 [  3]  adbellget  ? ding_57:bs [ding_57 eq tin = true];

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 3

 Enter expressions for the variable definitions (CR for none):
  ding_57:bs = tin

 ==> adbellget  ! tin;

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


 [  1]  enterbs1;
 [  2]  exitbs1;
 [  3]  forinst  ? status_58:ins [status_58 eq train_on_line = true];

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 3

 Enter expressions for the variable definitions (CR for none):
  status_58:ins = train_on_line

 ==> forinst  ! train_on_line;

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


 [  1]  enterbs1;
```

```
 [  2]  exitbs1;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> exitbs1;

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


 [  1]  enterbs1;
 [  2]  enterbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> s 2

 Events involved in the Synchronization 2:

   signal [exitbs1,enterbs2,trackside]  (* line 197 *)
    enterbs2 (* line 239 *);
 |[enterbs2,trackside]|
     box [trackside,enterbs2,forinst,accinst,adbellsend,adbellget,rebellsend,rebellget]
(accepting,normal,line_blocked) (* li
ne 201 *)
     box [trackside,enterbs,forinst,accinst,adbellsend,adbellget,rebellsend,rebellget]
(forwarding,normal,line_blocked) (* li
ne 379 *)
      enterbs2 (* line 357 *);
   |[enterbs2]|
     blk_sec [enterbs2,exitbs2] (0) (* line 216 *)
      enterbs2 (* line 255 *);

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> enterbs2;
```

Right after exitBS1 is picked up the enterBS2 action is the only one offered (ignoring the ever-present enterBS1). Thus the spec ensures atomicity of the exitBS;enterBS type of action sequence.

```
---------------------------------------------------------------------------


 [  1]  enterbs1;
 [  2]  i; (* trackside ! danger *)
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> s 3

 Event 3:

 blk_sec [enterbs2,exitbs2] (0) (* line 216 *)
 blk_sec [enterbs,exitbs] (inc(0)) (* line 256 *)
  exitbs2 (* line 262 *);

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> i; (* trackside ! danger *)
```

exitBS2 becomes available

Immediatelly after (but not before or during) the enterBS2 the trackside signal is returned to danger.

```
---------------------------------------------------------------------------


 [  1]  enterbs1;
 [  2]  rebellsend  ! call;
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> rebellsend  ! call;

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


 [  1]  enterbs1;
```

```
[  2]  rebellget  ? ding_61:bs [ding_61 eq call = true];
[  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  Enter expressions for the variable definitions (CR for none):
   ding_61:bs = call

  ==> rebellget  ! call;

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


[  1]  enterbs1;
[  2]  rebellsend  ! tout;
[  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> rebellsend  ! tout;
```

Our BOX lets the BOX in the rear (which may be waiting in forwarding mode with Status train_on_line) that the train is out of its section.

```
----------------------------------------------------------------------------


[  1]  enterbs1;
[  2]  accinst  ! line_blocked;
[  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> accinst  ! line_blocked;
```

Then it sets the rear BOX's Status to line_blocked

```
----------------------------------------------------------------------------


[  1]  enterbs1;
[  2]  i; (* exit (accepting, normal, train_on_line) *)
[  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> i; (* exit (accepting, normal, train_on_line) *)
```

After it has forwarded the train to the next section, our BOX exits to accepting mode so that it may accept in it's BLK_SEC a train from the BLK_SEC in the rear. Note that the Status is train_on_line. At this point, successful and safe passage of a train has been demonstrated, but stay tuned for a full tour of the spec.

```
----------------------------------------------------------------------------


[  1]  enterbs1;
[  2]  rebellget  ? ding_62:bs [ding_62 eq call = true];
[  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  Enter expressions for the variable definitions (CR for none):
   ding_62:bs = call

  ==> rebellget  ! call;

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


[  1]  enterbs1;
[  2]  rebellsend  ! call;
[  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> rebellsend  ! call;
```

```
------------------------------------------------------------------------


 [  1]  enterbs1;
 [  2]  rebellget  ? ding_66:bs [ding_66 eq ilc = true];
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  Enter expressions for the variable definitions (CR for none):
   ding_66:bs = ilc

  ==> rebellget  ! ilc;

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


 [  1]  enterbs1;
 [  2]  rebellsend  ! ilc;
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> rebellsend  ! ilc;

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


 [  1]  enterbs1;
 [  2]  accinst  ! line_clear;
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> s 1

 Event 1:

 blk_sec [enterbs1,exitbs1] (0) (* line 192 *)
 blk_sec [enterbs,exitbs] (inc(0)) (* line 256 *)
 blk_sec [enterbs,exitbs] (0) (* line 263 *)
  enterbs1 (* line 255 *);
```



```
<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> accinst  ! line_clear;

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


 [  1]  enterbs1;
 [  2]  rebellget  ? ding_67:bs [ding_67 eq tin = true];
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  Enter expressions for the variable definitions (CR for none):
   ding_67:bs = tin

  ==> rebellget  ! tin;

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


 [  1]  enterbs1;
 [  2]  rebellsend  ! tin;
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2
```

```
  ==> rebellsend  ! tin;

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


 [  1]  enterbs1;
 [  2]  accinst  ! train_on_line;
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> accinst  ! train_on_line;

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


 [  1]  enterbs1;
 [  2]  i; (* exit (forwarding, normal, train_on_line) *)
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 1

  ==> enterbs1;
```

A new train enters

```
---------------------------------------------------------------------------


 [  1]  enterbs1;
 [  2]  i; (* exit (forwarding, normal, train_on_line) *)
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> i; (* exit (forwarding, normal, train_on_line) *)
```

Notice that Status is still train_on_line

```
---------------------------------------------------------------------------


 [  1]  enterbs1;
 [  2]  adbellget  ? ding_68:bs [ding_68 eq call = true];
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  Enter expressions for the variable definitions (CR for none):
   ding_68:bs = call

  ==> adbellget  ! call;

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


 [  1]  enterbs1;
 [  2]  adbellsend  ! call;
 [  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> adbellsend  ! call;

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


 [  1]  enterbs1;
 [  2]  adbellget  ? ding_72:bs [ding_72 eq tout = true];
 [  3]  exitbs2;
```

Our BOX waits for the *tout* signal from the BOX in advance

```
<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2
```

```
Enter expressions for the variable definitions (CR for none):
 ding_72:bs = tout

==> adbellget  ! tout;

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


[  1]  enterbs1;
[  2]  forinst  ? status_73:ins [status_73 eq line_blocked = true];
[  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

Enter expressions for the variable definitions (CR for none):
 status_73:ins = line_blocked

==> forinst  ! line_blocked;
```

Our BOX's Status becomes line_blocked again.

```
----------------------------------------------------------------------------


[  1]  enterbs1;
[  2]  i; (* exit (forwarding, normal, line_blocked) *)
[  3]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

==> i; (* exit (forwarding, normal, line_blocked) *)

----------------------------------------------------------------
```

Our BOX exits to forwarding mode with Status line_blocked so that the train can be forwarded to the section in advance. Of course we assume that the exitBS2 is picked up at some point so that the model's state is consistent with the ABS protocol, although this is not explicitly shown in this LOLA transcript. Beyond this point the protocol is just repeated.

# Unsafe System Source Code

This LOTOS specification is used to demonstrate that the basic BLK_SEC system is not safe. We assume that for that to be demonstrated we only need to show that more than one trains can "happily" co-exist in the same BLK_SEC.

This specification actually extends the given basic BLK_SEC specification into one of three sequentially connected instances of the BLK_SEC process in order to show how multiple trains are permitted to move from one BLK_SEC to the next before the accepting BLK_SEC forwards any of them. This means that BLK_SECs are permitted to have more than one trains at one time.

See the following LOLA session transcript for a demonstration of this.

```
SPECIFICATION UNSAFE_BLK_SECS[enterBS1, exitBS1, exitBS2, exitBS3] : NOEXIT

(****************************IMPORT LIBRARIES******************************)

(*Import standard data types from available libraries*)

LIBRARY Boolean ENDLIB

(*************************************************************************)



(****************************TYPE DEFINITIONS******************************)

(*=====================================================================*)
(*The Count data type is used to count the number                      *)
(*of trains in a block section.                                        *)

TYPE Count IS Boolean
SORTS
  count

OPNS
  0        :               -> count
  inc, dec : count         -> count
  _eq_     : count, count  -> Bool
  noi, nod : count         -> Bool

EQNS FORALL x,y : count
  OFSORT count
    inc(dec(x)) = x;
    dec(inc(x)) = x;

  OFSORT Bool
    nod(0) = true; nod(inc(x)) = nod(x);
    noi(0) = true; noi(dec(x)) = noi(x);

    0 eq 0 = true;

    inc(x) eq y = x eq dec(y);
    dec(x) eq y = x eq inc(y);

    nod(x) => 0 eq inc(x) = false;
    noi(x) => 0 eq dec(x) = false;

ENDTYPE
(*=====================================================================*)
(*************************************************************************)
```

```
(****************************BEHAVIOUR**********************************)
(*********************************************************************)
(*Three consecutive BLK_SECs connected with middle one having its gates   *)
(*relabeled to reverse positions so that action synchronisation can occur.*)
(*The system is initialised to a state completely devoid of trains,       *)
(*i.e. the train count in all three of the block sections is initially 0  *)

BEHAVIOUR


  BLK_SEC[enterBS1, exitBS1] (0)

  |[exitBS1]|

  BLK_SEC[exitBS1, exitBS2]  (0)

  |[exitBS2]|

  BLK_SEC[exitBS2, exitBS3]  (0)

(*********************************************************************)
(*********************************************************************)

WHERE

(**********************PROCESS DEFINITIONS*******************************)
(*=================================================================*)
PROCESS BLK_SEC[enterBS, exitBS](trains:count) : NOEXIT :=

  (
    enterBS;
    BLK_SEC[enterBS, exitBS](inc(trains))
  )

  []

  (
    [not(trains eq 0)] -> exitBS;
                          BLK_SEC[enterBS, exitBS](dec(trains))
  )

ENDPROC
(*=================================================================*)
(*********************************************************************)

ENDSPEC
```

## LOLA Transcript: Demonstration of not safe basic BLK_SEC system

```
lola unsafe2.lot -l is

 LOLA. Version 3.6 1995.
 Departamento de Ingenieria Telematica. ETSIT.
 Universidad Politecnica de Madrid.

load

  Loading specification from unsafe2.
  Loading library from is.

TOPO_3R6 (Mon Jan 23 15:19:15 MET 1995) /u1/staff/air/topo
TOPO_3R6 (Mon Jan 23 15:19:15 MET 1995) /u1/staff/air/topo
`unsafe2.lsf' is up to date.

  Restoring unsafe2.lsf.

lola> s
step

 Rewriting expressions in the specification.
 Rewriting done.
 Analysing unguarded conditions.
 Analysis done.


 [  1]  enterbs1;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 1

  ==> enterbs1;

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


 [  1]  enterbs1;
 [  2]  exitbs1;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 1

  ==> enterbs1;

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


 [  1]  enterbs1;
 [  2]  exitbs1;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> exitbs1;
```

```
-------------------------------------------------------------------------

[  1]  enterbs1;
[  2]  exitbs1;
[  3]  exitbs2;
```

ExitBS1 also means train entering BLK_SEC number 2

ExitBS2 also means train entering BLK_SEC number 3

```
<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> s 2

 Events involved in the Synchronization 2:

   blk_sec [enterbs1,exitbs1] (0) (* line 62 *)
   blk_sec [enterbs,exitbs] (inc(0)) (* line 83 *)
   blk_sec [enterbs,exitbs] (inc(inc(0))) (* line 83 *)
   blk_sec [enterbs,exitbs] (inc(0)) (* line 90 *)
    exitbs1 (* line 89 *);
 |[exitbs1]|
   blk_sec [exitbs1,exitbs2] (0) (* line 66 *)
   blk_sec [enterbs,exitbs] (inc(0)) (* line 83 *)
    exitbs1 (* line 82 *);
```

As we can see here, there were 2 trains in the first BLK_SEC and then the one moved to the second BLK_SEC

```
<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> s 3

 Events involved in the Synchronization 3:

   blk_sec [exitbs1,exitbs2] (0) (* line 66 *)
   blk_sec [enterbs,exitbs] (inc(0)) (* line 83 *)
    exitbs2 (* line 89 *);
 |[exitbs2]|
   blk_sec [exitbs2,exitbs3] (0) (* line 70 *)
    exitbs2 (* line 82 *);

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> exitbs1;

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


 [  1]  enterbs1;
 [  2]  exitbs2;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> s 2

 Events involved in the Synchronization 2:

   blk_sec [exitbs1,exitbs2] (0) (* line 66 *)
   blk_sec [enterbs,exitbs] (inc(0)) (* line 83 *)
   blk_sec [enterbs,exitbs] (inc(inc(0))) (* line 83 *)
    exitbs2 (* line 89 *);
 |[exitbs2]|
   blk_sec [exitbs2,exitbs3] (0) (* line 70 *)
    exitbs2 (* line 82 *);

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> exitbs2;
```

As we can see here there are now 2 trains in the second BLK_SEC instance. As was the case with the first BLK_SEC holding two trains at the same time the main security mandate of a single train at any one time in a block section, is breached.

```
-------------------------------------------------------------------------


 [  1]  enterbs1;
 [  2]  exitbs2;
 [  3]  exitbs3;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> exitbs2;
```

```
---------------------------------------------------------------------------

 [  1]  enterbs1;
 [  2]  exitbs3;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> s 2

 Event 2:

 blk_sec [exitbs2,exitbs3] (0) (* line 70 *)
 blk_sec [enterbs,exitbs] (inc(0)) (* line 83 *)
 blk_sec [enterbs,exitbs] (inc(inc(0))) (* line 83 *)
  exitbs3 (* line 89 *);

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> exitbs3;
```

As we can see here the two trains now coexist in the third BLK_SEC instance.

```
---------------------------------------------------------------------------

 [  1]  enterbs1;
 [  2]  exitbs3;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?> 2

  ==> exitbs3;

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

 [  1]  enterbs1;

<n>,Undo,Menu,Refused,Sync,Print,Trace,Exit,?>
```

The trains have left our system and the enterBS1 action is again the only one on offer

# Conclusion

Writing specifications for concurrent and distributed systems in LOTOS is not an easy task, but it certainly is a far more a robust way to specify a system than using a pseudocode or flow-chart scheme before actually developing it using some high level language, and it is far less time consuming than debugging a lengthy and full of errors or possible deadlock situations program. This way its value becomes apparent, as a system can be specified and debugged of any logical errors before the actual lengthy and painful implementation phase.

LOTOS and TOPO/LOLA, when used in an appropriate way can make software development a much more rewarding process and they can provide a kind of foolproof security that is not experienced with much contemporary software/firmware.

Although the whole process of preparing this paper/specification has been a tremendous learning experience and fun (sometimes), I have to admit that it has been hard too. Maybe too hard…