Rigorous Methods for Software Engineering

Coursework 2

Design Level Modelling and Verification Exercise – Promela and iSPIN

F21RS1

Boris Mociłov
(H00180016)

Heriot – Watt University, Edinburgh
December 2014
Contents

1. Introduction ........................................................................................................................................ 3
2. Requirements ...................................................................................................................................... 4
   2.1. Assumptions .................................................................................................................................. 5
3. Refined system’s design ...................................................................................................................... 7
4. Promela source code .......................................................................................................................... 8
5. Verification ........................................................................................................................................... 8
6. Discussion ........................................................................................................................................... 8
7. Conclusion .......................................................................................................................................... 10
References ............................................................................................................................................... 11
Bibliography ........................................................................................................................................... 11
Appendix A ............................................................................................................................................ 12
Appendix B ............................................................................................................................................ 13
Appendix C ............................................................................................................................................ 14

Figures

Figure 1 Refined system .......................................................................................................................... 7
Figure 2 Refined system (detailed) ............................................................................................................ 8
Figure 3 Bank producer-consumer problem ............................................................................................ 9
Figure 4 Verification result 1 .................................................................................................................... 13
Figure 5 Verification result 2 .................................................................................................................... 14

Tables

Table 1: Requirements ........................................................................................................................... 5
Table 2: Assumptions ............................................................................................................................... 6

Code snippets

Code snippet 1 Source code for the extended TrainWare model ............................................................ 12
1. Introduction

This report discusses the coursework 2 for the module stated in the title. The purpose of this assignment is to extend the given system (TrainWare) based on the requirements given in the requirements section of this report. In addition, the extended version of the system will be formally verified using SPIN verification tool.

The report will first state the posed requirements, discussing their purpose in regards to the overall aim of the assignment. Later, assumption will be given, that will pose some additional requirements/constraints and show how identified assumptions affected decisions made during extensions and verifications processes. After, the refined system will be presented together with graphical overall and detailed view of the system as well as implementation source code. Following the presented refined model of the system, verification results will be given. At the end a short discussion section will present another problem that could be modeled and verified using SPIN tool.
2. Requirements

Identified requirements are split into two parts, where first part is concerned with the requirements for the implementation of the desired system, while second part identifies verification requirements for the refined model.

<table>
<thead>
<tr>
<th>Task</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modelling Task</strong></td>
<td></td>
</tr>
<tr>
<td>Extend the original model by adding constraints to it</td>
<td>To guarantee given safety property</td>
</tr>
<tr>
<td>Extensions should be distributed across the model</td>
<td>To enforce distributed communication paradigm</td>
</tr>
<tr>
<td>Each station process must include a track-side signal</td>
<td>Track-side signal tells trains to either proceed into a tunnel or to stop and wait at a station</td>
</tr>
<tr>
<td>Each track-side signal must be controlled by an associated to it signal box</td>
<td>Trains must be able to know when to proceed</td>
</tr>
<tr>
<td>A signal box must only be able to communicate with adjacent stations signal boxes</td>
<td>A signal box should be able to determine whether train can proceed further or not</td>
</tr>
<tr>
<td>A station must communicate only with its associated signal box</td>
<td>There must be some kind of communication between two mechanisms in order to be able to progress further</td>
</tr>
<tr>
<td>A station and associated signal box can only observe trains as they arrive and depart from their stations</td>
<td>Mimic the real-life set-up</td>
</tr>
<tr>
<td>Nor station or associated signal box can look inside tunnels adjacent to a station</td>
<td>Otherwise the problem would be easy to solve (real-life set-up does not allow such functionality [i.e. what if tunnel is an underground tunnel?]}}</td>
</tr>
<tr>
<td>A station and its associated signal box may observe trains when they arrive at that particular station</td>
<td>Inferred from previous requirement – station must be able to observe trains somehow and communicate new state to</td>
</tr>
</tbody>
</table>
A signal box must know when a train has passed its associated station (i.e. track-side signal) to communicate with the station and associated signal box. This communication is necessary to keep their states synchronized.

**Verification Task**

- **Verify desirable safety property using global system assertions**
  - Check for safety of the system under model execution.

- **Verify desirable safety property using reasoning about temporal properties**
  - Guarantee correctness of the system in regard to safety property.

### Requirements identified: 12

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1. Assumptions</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modelling Task</strong></td>
<td></td>
</tr>
<tr>
<td>During initialisation phase, a train may be pushed into a tunnel, while another train is already inside the same tunnel</td>
<td><strong>Setup</strong> process consults signal box at a station before pushing the train inside a tunnel.</td>
</tr>
<tr>
<td>Distributed systems design should use some underlying communication mechanism</td>
<td>Three communication channels are used to support communication:</td>
</tr>
<tr>
<td>a) Between a station and associated signal box</td>
<td></td>
</tr>
<tr>
<td>b) Between associated signal box and stations’ signal box at one side of adjacent tunnel</td>
<td></td>
</tr>
<tr>
<td>c) Between associated signal box and stations’ signal box at another side of adjacent tunnel</td>
<td></td>
</tr>
<tr>
<td>A station’s associated signal box should use same parameters as setup of a</td>
<td>Station’s process runs a signal box process from within itself with relevant</td>
</tr>
<tr>
<td>station’s process arguments</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Every station’s associated signal box must be able to communicate with adjacent stations’ signal boxes and with the station itself</td>
<td>3 communication channels are used at every station setup phase</td>
</tr>
</tbody>
</table>

**Verification Task**

<table>
<thead>
<tr>
<th><strong>ltl</strong> temporal logic should be used for specification of a never claim instead of never {...} definition</th>
<th><strong>ltl</strong> temporal logic had been used for the verification of the model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desirable safety property must be satisfied at all the times</td>
<td><strong>ltl</strong> ‘always’ property had been used for the verification. This means that all states of the system satisfy safety property</td>
</tr>
<tr>
<td>Validation labels should be used to validate productive progress of the system</td>
<td>‘Progress’ labels are used for every station and for every signal box associated to a station</td>
</tr>
<tr>
<td><strong>ltl</strong> formulae should be used as an acceptance condition</td>
<td><strong>ltl</strong> formulae is defined and used for the verification of the model to support the never claim (enforce desirable safety property)</td>
</tr>
<tr>
<td>Additional label should be used to be able to perform extended verification of the developed model</td>
<td>Verifications, such as ‘progress’ and ‘end’ are used, while ‘accept’ is generated from the <strong>ltl</strong></td>
</tr>
</tbody>
</table>

**Assumptions made: 9**

Table 2: Assumptions
3. Refined system’s design

The original system’s design is marked with blue color, while everything else is a refinement (additional constraints).

Figure 1 Refined system
Detailed excerpt of the overall system, where elements marked with blue color correspond to the original system, while the rest is additional constraints imposed on the final system.

Figure 2 Refined system (detailed)

4. Promela source code
Source code for the extended TrainWare model is presented in the Appendix A

5. Verification
1. Safety property verification (w/ invalid endstates & assertion violations & ltl never claim property)
   Results presented in Appendix B
2. Liveness property verification (w/ non-progress cycles & strong fairness [default])
   Results presented in Appendix C

6. Discussion
Apart from this exercises modeling application of SPIN and an array of inspiring applications of SPIN presented at (Inspiring Applications of Spin), this tool can be used for most of the producer-consumer problems. For instance, it would be interesting to use SPIN to model
such classical problems as, for example, the “Dining philosophers problem”. Apart from theoretical problems, SPIN can be used for practical situations as well.

Lets look closer at the Banking System, where main instance of a bank tries to keep accounts balances up-to-date and provide services, such as: withdrawals and deposits to many users of that bank. For described banking system a control mechanism could be modeled using SPIN, where each transaction would require an enforced protocol between the instance of a bank and an instance of a user. Graphically the problem could be viewed as:

![Graphical representation of the banking system]

Figure 3 Bank producer-consumer problem

Where users either deposit or withdraw money from the bank and the bank has a stack of money in its vault, which it distributes or fills-up based on the operations from its users.

If such a system is to be modeled and verified in SPIN, then following properties would have to be satisfied to be able to verify that the designed model/system is correct.

1. Safety property
   • At all the times, the amount of money in the banks vault must be greater than 0 (and less than was printed by the Treasury)
   • In case of a deposit account:

   At all times users account balance must be greater or equal to 0

2. Response property
   • Whenever a user makes a request (deposit or withdraw), he will eventually be serviced by the bank.

3. Precedence property
• For a user to be able to retrieve money from his account, he must make request before getting a response
• The bank must get a request before serving money to user

7. Conclusion
In conclusion, the extension task of the given ‘unsecure’ TrainWare model had been achieved. Verification of the extended model had been performed and it can be stated that the model is ‘secure’ – the safety property “A tunnel can only be occupied by one train at a time” is satisfied.
The report should contain enough information regarding the extended secure version of the TrainWare. Additionally, another interesting application of the SPIN tool is presented, which can benefit from the modeling tool to see whether identified properties can be met.
References


Bibliography


Appendix A

```c
mtype = { free_token }
mtype = { stop, proceed }
mtype = { A, B, C, D }

typedef train_holder_struct
{
    byte train_id;
    mtype station;
} chan TunnelAB = [2] of (byte);
chan TunnelBC = [2] of (byte);
chan TunnelCD = [2] of (byte);
chan TunnelDA = [2] of (byte);

chan CommunicationChannelAB = [0] of {mtype};
chan CommunicationChannelIBC = [0] of {mtype};
chan CommunicationChannelICD = [0] of {mtype};
chan CommunicationChannelIDA = [0] of {mtype};
chan CommunicationChannelA = [0] of {mtype};
chan CommunicationChannelB = [0] of {mtype};
chan CommunicationChannelC = [0] of {mtype};
chan CommunicationChannelD = [0] of {mtype};

#define p (len(TunnelAB) < 2 && len(TunnelBC) < 2 && len(TunnelCD) < 2 && len(TunnelDA) < 2)
lif p1 {} p

proctype Station(mtype station; chan in_track, out_track, communication_channel_at_station, previous_communication_channel, next_communication_channel)
{
    run SignalBox(communication_channel_at_station, previous_communication_channel, next_communication_channel);
    train_holder_struct train_holder;
    train_holder.station = station;
    progress_station: do
        :: in_track?train_holder.train_id; communication_channel_at_station!stop; communication_channel_at_station?proceed;
        out_track!train_holder.train_id
    od
}

proctype Setup(chan track, communication_channel_at_station; byte train)
{
    end_setup: communication_channel_at_station?proceed; track!train;
}

proctype SignalBox(chan communication_channel_at_station, previous_communication_channel, next_communication_channel)
{
    progress_signal_box: do
        :: communication_channel_at_station?proceed; next_communication_channel?free_token
        :: communication_channel_at_station?stop; previous_communication_channel?free_token
    od
}

proctype Monitor(){
    progress_monitor: do
        :: assert(nfull(TunnelAB) && nfull(TunnelBC) && nfull(TunnelCD) && nfull(TunnelDA))
    od
}

init {
    atomic{
        run Setup(TunnelBC, CommunicationChannelB, 1);
        run Setup(TunnelDA, CommunicationChannelD, 2);
        run Station(A, TunnelDA, TunnelAB, CommunicationChannelA, CommunicationChannelIDA, CommunicationChannelAB);
        run Station(B, TunnelAB, TunnelBC, CommunicationChannelB, CommunicationChannelBIC, CommunicationChannelIBC);
        run Station(C, TunnelBC, TunnelCD, CommunicationChannelC, CommunicationChannelBC, CommunicationChannelICD);
        run Station(D, TunnelCD, TunnelIDA, CommunicationChannelID, CommunicationChannelCD, CommunicationChannelIDA);
        run Monitor();
    }
}
```

Code snippet 1 Source code for the extended TrainWare model
Appendix B

verification result:
spin -a last_try.pml

l11 p1: [ ( ((len(TunnelAB)<2)) && ((len(TunnelBC)<2))) && ((len(TunnelCD)<2))) &&
((len(TunnelDA)<2)))
gcc -DMEMLIM=1024 -D2 -DXUSAFE -DSAFETY -w -o pan pan.c
./pan -m10000

Pid: 3052
warning: never claim + accept labels requires -a flag to fully verify

(Spin Version 6.3.2 -- 17 May 2014)
+ Partial Order Reduction

Full statespace search for:
never claim + (p1)
assertion violations + (if within scope of claim)
cycle checks = (disabled by -DSAFETY)
invalid end states = (disabled by never claim)

State-vector 256 byte, depth reached 255, errors: 0
625 states, stored
1109 states, matched
1734 transitions (= stored+matched)
6 atomic steps
hash conflicts: 0 (resolved)

Stats on memory usage (in Megabytes):
0.169 equivalent memory usage for states (stored*(State-vector + overhead))
0.288 actual memory usage for states
128.000 memory used for hash table (-w24)
0.534 memory used for DFS stack (-m10000)
128.730 total actual memory usage

unreached in proctype Station
  last_try.pml:39, state 11, "-end-")(1 of 11 states)
unreached in proctype Setup
  (0 of 3 states)
unreached in proctype SignalBox
  last_try.pml:51, state 8, "-end-")(1 of 8 states)
unreached in proctype Monitor
  last_try.pml:57, state 5, "-end-")(1 of 5 states)
unreached in init
  (0 of 9 states)
unreached in claim p1
  _spin_nvr.tmp:8, state 10, "-end-")(1 of 10 states)

pan: elapsed time 0.01 seconds
No errors found -- did you verify all claims?

Figure 4 Verification result 1
Appendix C

verification result:
spin -a last_try.pml

ltl p1: [[] (((len(TunnelAB)<2)) & ((len(TunnelBC)<2))) &
((len(TunnelCD)<2)) &
((len(TunnelDA)<2)))
gcc -DMEMLIN=1024 -D2 -DXUSAFE -DNP -W -O pan pan.c
./pan -m10000 -l
PId: 9389

(Spin Version 6.3.2 -- 17 May 2014)
+ Partial Order Reduction

Full statespace search for:
never claim + (np_)
assertion violations + (if within scope of claim)
non-progress cycles + (fairness disabled)
invalid end states - (disabled by never claim)

State-vector 256 byte, depth reached 255, errors: 0
1160 states, stored
2839 states, matched
3999 transitions (= stored+matched)
12 atomic steps
hash conflicts: 0 (resolved)

Stats on memory usage (in Megabytes):
0.314 equivalent memory usage for states (stored*(State-vector + overhead))
128.000 memory used for hash table (-w24)
0.534 memory used for DFS stack (-m10000)
128.827 total actual memory usage

unreached in proctype Station
last_try.pml:39, state 11, "-end="
(1 of 11 states)
unreached in proctype Setup
(0 of 3 states)
unreached in proctype SignalBox
last_try.pml:51, state 8, "-end="
(1 of 8 states)
unreached in proctype Monitor
last_try.pml:57, state 5, "-end="
(1 of 5 states)
unreached in init
(0 of 9 states)
unreached in claim p1
_spin_nvr.tmp:3, state 6,
"!((((len(TunnelAB)<2))&((len(TunnelBC)<2)))&((len(TunnelCD)<2))&((len(TunnelDA)<2)))"
_spin_nvr.tmp:3, state 6, "(1)"
_spin_nvr.tmp:8, state 10, "-end="
(2 of 10 states)

pan: elapsed time 0.01 seconds
No errors found -- did you verify all claims?

Figure 5 Verification result 2

14