Proof Of Modes Management : Follow up

Roméo Courbis

March 20, 2015

Contents

1 Introduction

In the context of the Open-ETCS project, Scade models are developped according to specifications. This documents presents how to verify that those models correspond to their specification. A methodology of verification is presented section 3 and the next sections present verifications that have been made over differents parts of the Scade model.

2 Documents used

Local ID	Title	Reference	Issue	Date
D1	System Requirements Specification Chapter 5 Procedure	SUBSET-026-5	3.3.0	07/03/2012
D2	System Requirements Specification Chapter 4 Modes and Transitions	SUBSET-026-4	3.3.0	07/03/2012

3 Methodology for flowchart verification

This section presents the methodology used to verify that the SRS 5 is correctly implemented into a scade model.

3.1 Overview of proof methodology

The document D1 is a specification written with flowcharts. A scade model is made according to this specification.

We want to known that the scade model is a correct implementation of the specification. To do so, we will prove that the scade model is equivalent to flowcharts.

Starting with one flowchart, a first step is to identify to which scade inputs and outputs this flowchart refers to. Usually, there are states which send data (scade model outputs) and there are conditions depending on variable(s) value(s) (scade model inputs).

Then, the flowchart is transalted into an equivalent HLL program using the scade inputs names. Scade outputs names are used to write proof obligation.

Once those steps done, it is possible to prove (or disprove) that for all possible inputs, the scade model and the flowcart HLL program compute the same outputs.

3.2 Flowchart to HLL program

In this section we will see how to transform a flowchart from the SRS documentation into an equivalent HLL program. Section 3.2.1 presents definitions of flowchart shapes, the section 3.2.2 explains how a flowchart is interpreted in the context of the synchronous language HLL and the section 3.2.3 shows a generic methodology for the translation of this type of flowchart into a HLL program.

3.2.1 Flowcharts in SRS documentation

Here is a list of flowchart shapes and there meaning :

Rectangle The rectangle usually means that an action has to be taken. In our case, this shape is assimilated to a state where a value can be send through one or more outputs.

Rounded rectangle This shape is equal to the rectangle shape except when you read it (see section 3.2.2). If no arrow is leaving from such shape, it is considered as a rectangle.

Diamond This shape usually corresponds to a question and the answer indicates the reading direction. Answers are listed on different arrows leaving the diamond.

Arrow This shape indicates the reading direction. An arrow should start from a shape and end to another shape. The arrow may be labeled. This label is usually a condition which must be fulfilled to continue to the next shape.

Circle This is usually a terminal shape, which indicates an end of the flowchart. No more actions will be taken.

Also, a definition of states (which correspond to rectangles and rounded rectangles) is given in the SRS at section 5.3.2.3.

A definition of transitions (which correspond to diamonds and arrows) is given in the SRS at section 5.3.2.4.

3.2.2 Reading a flowchart in a HLL context

The concept of cycle is used in the synchronous language HLL and this impacts how a flowchart is read when we want to translate it into a HLL program.

At the first cycle, we start to read the flowchart from an initial state which is a state where there is only leaving arrows. Then, at each cycle we go from one state A to a state B, where A could be equal to B, until we reach a final state which is a state where there is no leaving arrows.

During a cycle execution it is not possible to read more than one state, so when reaching a state the reading is stopped even if it is possible to leave the current state following arrows.

Moreover, intial states can sometimes behave as a global condition which trigger or stop (and re-initialize) the reading of the flowchart. In this case, this initial state will not be seen as a state as mentionned above. Reading corresponding description of the requirement should help in state interpretation.

3.2.3 Transformation to HLL program

In this section, we will describe how to translate examples of simples flowcharts into a HLL programs. Combining those examples is a way to translate more complex flowcharts.

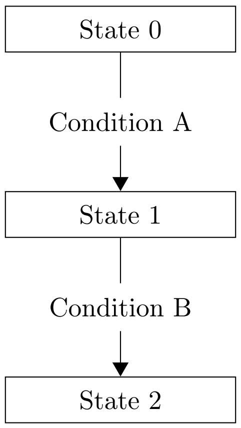

Figure 1: Three states and two transitions.

Rectangle -Labeled arrow-> Rectangle -Labeled arrow-> Rectangle.
The following HLL program is translation of the flowchart figure 1 :

```
Types:
enum{'state_0','state_1','state_2'} states;

Declarations:
states last_states; // State reached at previous cycle
states current_states; // Current state

states s_0;
states s_1;

bool s_0_var_state;
bool s_1_var_state;
bool s_2_var_state;

Definitions:
s_0_var_state := State 0;
s_1_var_state := State 1;
s_2_var_state := State 2;

last_state := pre(current_state,'state_0'); // 'state_0' is an initial state
current_state := (last_state
                 | 'state_0' => s_0
                 | 'state_1' => s_1
                 | 'state_2' => 'state_2' // 'state_2' is a final state
                 );
```

```
s_0 := if (Condition A) then
       'state_1'
       else
       'state_0';

s_1 := if (Condition B) then
       'state_2'
       else
       'state_1';

Proof Obligations:
(current_state = 'state_0') <-> s_0_var_state;
(current_state = 'state_1') <-> s_1_var_state;
(current_state = 'state_2') <-> s_2_var_state;
```

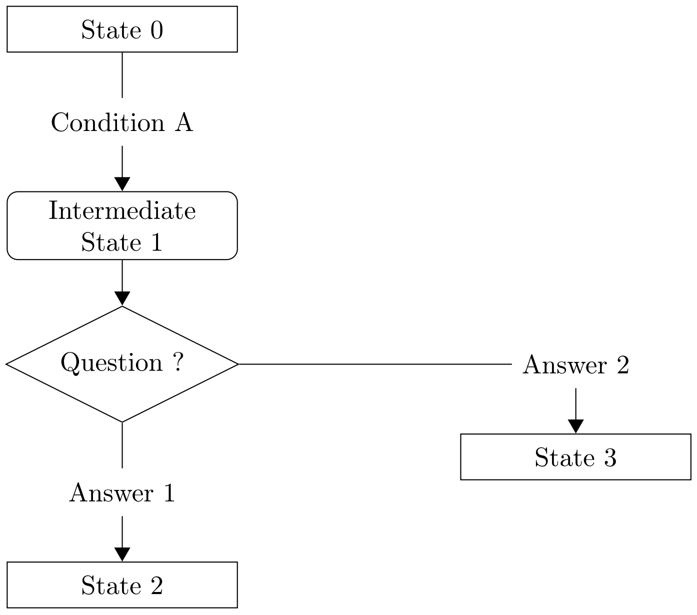

Figure 2: Three states and two transitions.

Rectangle -Labeled arrow-> Rounded rectangle –> Diamond -» Rectangle x 2. The following HLL program is translation of the flowchart figure 2 :

```
Types:
enum{'state_0','inter_state_1','state_2','state_3',
     'state_0_from_diamond','state_2_from_state_1','state_3_from_state_1'} states;

Declarations:
states last_states; // State reached at previous cycle
states current_states; // Current state

states s_0;
states inter_s_1;

bool s_0_var_state;
bool s_2_var_state;
bool s_3_var_state;
bool inter_s_1_state;

Definitions:
s_0_var_state := State 0;
s_2_var_state := State 2;
s_3_var_state := State 3;
inter_s_1_state := Inter_State 1;

last_state := pre(current_state,'state_0'); // 'state_0' is an initial state
current_state := (last_state
                 | 'state_0' => s_0
                 | 'state_0_from_diamond' => s_0
                 | 'state_2_from_state_1' => 'state_2'
                 | 'state_3_from_state_1' => 'state_3'
                 | 'state_2' => 'state_2' // 'state_2' is a final state
                 | 'state_3' => 'state_3' // 'state_3' is a final state
                 );

s_0 := if (Condition A) then
       inter_s_1
       else
       'state_0';

inter_s_1 := if (Diamond_cond_1) then
             'state_2_from_state_1'
             elif (Diamond_cond_2) then
             'state_3_from_state_1'
             else
             'state_0_from_diamond';


Proof Obligations:
(current_state = 'state_0') <-> s_0_var_state;
(current_state = 'state_0_from_diamond') <-> s_0_var_state & inter_s_1_state;
(current_state = 'state_2_from_state_1') <-> s_2_var_state & inter_s_1_state;
(current_state = 'state_3_from_state_1') <-> s_3_var_state & inter_s_1_state;
(current_state = 'state_2') <-> s_2_var_state;
(current_state = 'state_3') <-> s_3_var_state;
```

Here is only a few examples of flowcharts translation. An objective can be to automatize this translation.

3.3 Proofs that a flowchart corresponds to its model

In HLL programs translated from 1 and 2, we can see that proof obligations are equivalences between states. Usually, right hand side of the equivalence are streams composed with Scade model outputs.

4 Proof of : Shunting Initiated By Driver On

4.1 Files used for the proof

Used node	Property file
Procedures::SH_Initiated_By_Driver_On	shunting_initiated_by_driver_on.hll

4.2 What is proved ?

We want to prove that the procedure SH_Initiated_By_Driver_On is a correct implementation of the section 5.6 Shunting Initiated By Driver (cf. D1).

This procedure is part of several scade sub-models as the procedure Procedure_StartOfMission or SH_Initiated_By_Driver.

To prove this, a modelisation of the flowchart is proposed in the property file. However, the flowchart is not entirely modelised because the Scade model does not seem to represent all of it. Requirements that are not modelised are the following : `D030`, `A030`, `A095`, `S100` and `A115`.

Moreover, the `S0` state of this flowchart is not modelised. In fact, this state appears to be a condition that must be fulfilled to start the reading of the flowchart. If at any state this condition is no more true, then the reading of the flow chart stop and is reinitialized.

Constraints used One constraint is used in this model. Without this constraint, a counter-example is reachable.

This constraint specifies that the flowchart shall return to his initial state a cycle after the dead-end state `A100` is reached.

Actually, if this constraint is not verified, the Scade model could return `False` for the request of "End of Mission" procedure which contradicts the flowchart (`D040` and `A100` requirements).

In our implementation of the flowchart, this constraint is modelised by verifying that when we are in `A100` the request of "End of Mission" procedure correspond to the value of the input `On-going Mission` as it is modelised in the Scade model.

This modelisation is justified by the fact that, according to `A050`, `D040` and `A100`, if the input `On-going Mission` is `True` then the "End of Mission" request is `True`, so equal to `On-going Mission`. Also, as transition to SH mode is enable (`A050`) when "End of Mission" request is made, the system should go to SH mode (or another mode except SB mode). So the next cycle the driver should not execute the shunting selection procedure and the next time this pocedure is executed it should start at the initial state.

4.3 Results

Considering the constraint and our model, the Scade model of SH_Initiated_By_Driver_On corresponds to the specification.

5 Proof of : Shunting Initiated By Driver

5.1 Files used for the proof

Used node	Property file
Procedures::SH_Initiated_By_Driver	shunting_initiated_by_driver.hll

5.2 What is proved ?

We want to prove that the procedure SH_Initiated_By_Driver is a correct implementation of the section 5.6 Shunting Initiated By Driver (cf. D1).

To prove this, a modelisation of the flowchart is proposed in the property file. However, the flowchart is not entirely modelised because the Scade model does not seem to represent all of it. Requirements that are not modelised are the following : `D030`, `A030`, `A095`, `S100` and `A115`.

Constraints used One constraint is used in this model. Without this constraint, a counter-example is reachable.

This constraint specifies that the flowchart shall return to his initial state a cycle after the dead-end state `A100` is reached.

Actually, if this constraint is not verified, the Scade model could return `False` for the request of "End of Mission" procedure which contradicts the flowchart (`D040` and `A100` requirements).

In our implementation of the flowchart, this constraint is modelised by verifying that when we are in `A100` the request of "End of Mission" procedure correspond to the value of the input `On-going Mission` as it is modelised in the Scade model.

This modelisation is justified by the fact that, according to `A050`, `D040` and `A100`, if the input `On-going Mission` is `True` then the "End of Mission" request is `True`, so equal to `On-going Mission`. Also, as transition to SH mode is enable (`A050`) when "End of Mission" request is made, the system should go to SH mode (or another mode except SB mode). So the next cycle the driver should not execute the shunting selection procedure and the next time this pocedure is executed it should start at the initial state.

5.3 Results

Considering the constraint and our model, the Scade model of SH_Initiated_By_Driver corresponds to the specification.

6 Proof of : Start Of Mission

6.1 Files used for the proof

Used node	Property file
Procedures::Procedure_StartOfMission	startofmission_proof.hll
ManageModes	startofmission_topnode_proof.hll

6.2 What is proved ?

We want to prove that the procedure Procedure_StartOfMission is a correct implementation of the section 5.6 Procedure Start of mission (cf. D1).

The flowchart is not entirely modelised in the Scade model and an implementation of emergency brake management is present in the Scade model.

States modelised are S0, from S10 and from S20. In fact, S0 is a condition which could trigger S10 and S20 states or stop the procedure execution at an time.

Also, emergency brake can be triggered at any time during the procedure execution.

The "On Going Mission" variable, input of SH Initiated by Driver, is forced to False. This is justified by SRS 5, section "5.4.6 Entry to Mode Considered as a Mission".

TODO : make the actual flowchart modelised in HLL

Constraints used

1. Level should not change : During Start of Mission procedure, Level should not change.
2. Train Data should not change : During Start of Mission procedure, train data should not be modified. Then, their validity should not change.
3. The train shall be at standstill to start the pocedure.
4. Same constraint used for the procedure SH Initiated By Driver proof.

6.3 Results

Considering constraints and our model, the Scade model of Procedure Start of Mission corresponds to the specification.