Proof Of Modes Management : Follow up

Roméo Courbis

April 14, 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 these 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 would like to known that the scade model is a correct implementation of the specification. To do so, the scade model is proven equivalent to its 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 translated into an equivalent HLL model using the scade inputs names. Scade outputs names are used to write proof obligation.

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

3.2 Flowchart to HLL model

In this section, a presentation of flowchart (from the SRS documentation) transformation into an equivalent HLL model is made. 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 model.

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 also means that an action has to be taken, like rectangle shape, but rounded rectangle are read differently. We will call them action.

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 model.

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.

When, during a cycle, the reading go through an action (rounded rectangle) then its coresponding actions are taken and the reading continue. So it is not possible to stop the reading in an action.

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, initial 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 mentioned above. Reading corresponding description of the requirement should help in state interpretation.

3.2.3 Transformation to HLL model

In this section, we will describe how to translate examples of simple flowcharts into a HLL models. Combining these examples is a way to translate more complex flowcharts.

In this document, there is only a few examples of flowcharts translation. An objective can be to automatize this translation.

3.2.4 Example: Rectangle -Labeled arrow-> Rectangle -Labeled arrow-> Rectangle.

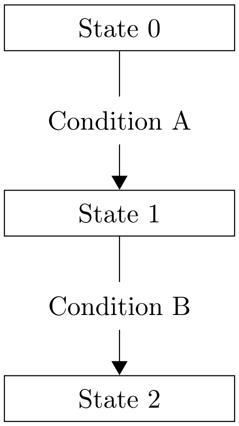

Figure 1: Three states and two transitions.

The following HLL model 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

// Transitions:
states transition_A;
states transition_B;

/* Declarations of actions took (= output(s) are set to a value) for
 each state: */
bool s_0_outputs;
bool s_1_outputs;
bool s_2_outputs;

/* Main proof obligation */
bool po;

Definitions:
s_0_outputs := State 0;
```

```
s_1_outputs := State 1;
s_2_outputs := State 2;

// Definition of transitions between states:
last_state := pre(current_state,’state_0’); // ’state_0’ is an initial state
current_state := (last_state
                 | ’state_0’ => transition_A
                 | ’state_1’ => transition_B
                 | ’state_2’ => ’state_2’ // ’state_2’ is a final state
                 );

transition_A := if (Condition A) then
                ’state_1’
                else
                ’state_0’;

transition_B := if (Condition B) then
                ’state_2’
                else
                ’state_1’;


/* Main proof obligation */
po := if (current_state = ’state_0’) then s_0_outputs
      elif (current_state = ’state_1’) then s_1_outputs
      elif (current_state = ’state_2’) then s_2_outputs
      else False;

Proof Obligations:
po;
```

3.2.5 Example: Rectangle -Labeled arrow-> Rounded rectangle –> Diamond -» Rectangle x 2.

The following HLL model is translation of the flowchart figure 2:

```
Types:
enum{’state_0’,’state_2’,’state_3’} states;

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

// Transitions:
states transition_A;
states transition_question;

/* Declarations of actions took (= output(s) are set to a value) for
 each state: */
bool state_0_outputs;
bool state_2_outputs;
bool state_3_outputs;

// Activation state of rounded rectangle states
bool action_1_activation;
```

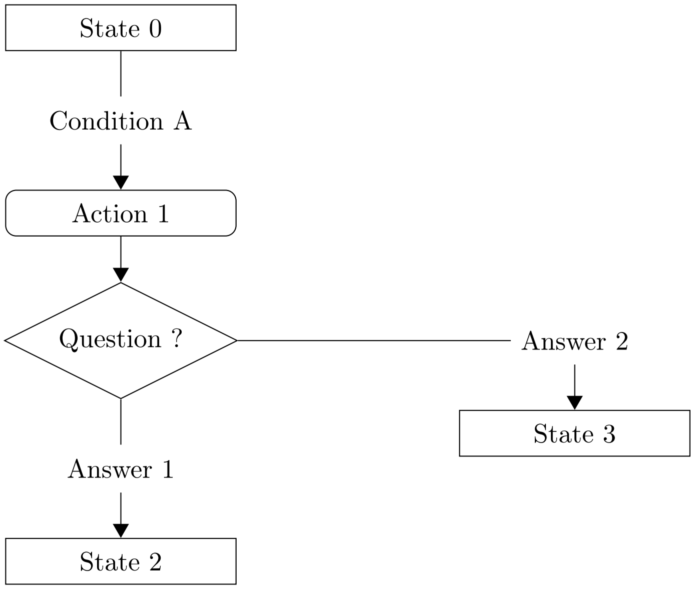

Figure 2: Three states, one rounded rectangle state, one diamond and one transition.

```
/* Main proof obligation */
bool po;

Definitions:
state_0_outputs := State 0;
state_0_action_1_outputs := State 0 & Action 1;
state_2_outputs := State 2;
state_2_action_1_outputs := State 2 & Action 1;
state_3_outputs := State 3;
state_3_action_1_outputs := State 3 & Action 1;

// Definition of transitions between states:
last_state := pre(current_state,'state_0'); // 'state_0' is an initial state
current_state := (last_state
                 | 'state_0' => transition_A
                 | 'state_2' => 'state_2' // 'state_2' is a final state
                 | 'state_3' => 'state_3' // 'state_3' is a final state
                 );

transition_A := if (Condition A) then
                transition_question
                else
                'state_0';

transition_question := if (Answer 1) then
                       'state_2'
                       elif (Answer 2) then
                       'state_3'
                       else
                       ’state_0’;

action_1_activation := (last_state = ’state_0’) & Condition A;

/* Main proof obligation */
po := if (current_state = ’state_0’ & ~action_1_activation) then state_0_outputs
      elif (current_state = ’state_0’ & action_1_activation) then state_0_action_1_outputs
      elif (current_state = ’state_2’ & ~action_1_activation) then state_2_outputs
      elif (current_state = ’state_2’ & action_1_activation) then state_2_action_1_outputs
      elif (current_state = ’state_3’ & ~action_1_activation) then state_3_outputs
      elif (current_state = ’state_3’ & action_1_activation) then state_3_action_1_outputs
      else False;

Proof Obligations:
po;
```

3.3 Equivalence proof between flowchart and Scade model

We want to prove that our flowchart take the same actions than the Scade model for each flowchart state.

To do this, three elements are required to construct the proof obligation:

1. State names, usualy they are listed in an enumerated type.

2. Action (corresponding to rounded rectangle) activation conditions. They are implemented by boolean variables which are `True` when the action is read during a cycle execution, `False` otherwise.

3. Boolean variables representing outputs values for each states. Each Scade model outputs must be present in each variable.

Once these three elements are implemented, the proof obligation can be constructed using an `if ... then ... elif ... else ...;` structure as we can see in previous examples.

This structure will represent a following statement:

```
if "flowchart is in a state (and if action(s) is taken (or not))"
then "Scade outputs are set to some values"
elif ...
```

The `... else False;` part is a security: if we forgot a case the proof obligation will be `False` and our attention will be caught on this problem.

With this proof obligation, S3 tool will be able to prove or disprove (or fail to prove) that, at each cycle, actions taken by a flowchart HLL model are the same as Scade model.

4 Proof of: Shunting Initiated By Driver

4.1 Files used for the proof

Used node	Property file
Procedures::SH_Initiated_By_Driver	shunting_initiated_by_driver.hll
ManageModes	shunting_initiated_by_driver_topnode.hll

4.2 What we want to prove

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 modelization of the flowchart is proposed in the property file. However, the flowchart is not entirely modelized by the Scade model, assumptions are taken:

1. A030 and National Trip occurrence is covered by Trip conditions and procedures with higher priority.

2. Communication with RBC is out of the scope of this function (repetition of message to RBC, waiting for acknowledgment, ...)

These assumptions have the following consequences:

1. `D030` and `A030` are not in this HLL model. This implies that we have to deal with the "NTC" transition leaving D020. To resolve this problem, we assume this transition is merged with the "0/1" transition.

2. `D080`, `A095`, `S100`, `A115` are not in this HLL model. This implies the "No" transition leaving D040 is going nowhere. In this case, we assume this transition leads to an "End" state, which means that process shall end.

4.3 Results and conclusions

Considering assumptions, SH_Initiated_By_Driver Scade model does not correspond to the specification.

This is explained by the following differences between the specification and the Scade model:

1. In the specification, `S0` state has no associated requirement other than the `E015` requirement. Considering only the flowchart, `S0` is an initial state which can be left when the conditions explained in `E015` requirement are fulfilled. Usually, states are not conditions. The Scade model is considering `S0` state as a re-initialization state: `S0` conditions must be fulfilled to allow flowchart execution.

2. The flowchart have end states where no actions are made. Actually, `A100` (drawn by rounded rectangles) is not a state, so it is read one time contrary to a state and then the process (of reading the flowchart) is stopped. This

contradicts the Scade model where this element is seen and modelized as a state.

3. In the case of A220, the flowchart leads to an end state after this action. However, the Scade model leads to the initial state (a state just before E015).

4. The A100 action is specified by a requirement which says that the process shall go to the "End of Mission" procedure. In the Scade model, we assume this requirement is corresponding to the output called End_Of_Mission_Procedure_Req. This output shall be True when conditions to reach A100 are fulfilled, False in all other cases. However this output is equal to the input On_Going_Mission and, because of the previous point, this implies that output End_Of_Mission_Procedure_Req can have a changing value when Scade model is in a dead end state where conditions to reach A100 were fulfilled.

Here is propositions of solutions to resolve previous problems:

Proposition for 1. The ambiguity of S0 definition must be clarified: either it is a state with a clear action in its definition, or it is a condition which trigger or not the procedure.

Proposition for 3. To be equivalent to the specification, the Scade model should end in a dead end state instead of going back to its initial state.

Proposition for 2. and 4. To be equivalent to the specification, the Scade model should not have dead end states where actions can be taken (outputs are set to values). Actions in these dead end state should be modelized as "emit".

However, in the case where the Scade model reaches a final state of the Shunting Initiated By Driver procedure and where it always leads to a mode change at next cycle, which is not a mode in S0 condition, then the procedure will go back to its initial state after one cycle in a final state.

So, if this behavior is verified, this will resolve points 2. and 4.

5 Proof of: Shunting Initiated By Driver On

5.1 Files used for the proof

Used node	Property file
Procedures::SH_Initiated_By_Driver_On	shunting_initiated_by_driver_on.hll

5.2 What we want to prove

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).

To prove this, the same HLL model created for the verification of SH_Initiated_By_Driver (section 4) is used except that S0 state is not modelized.

5.3 Results and conclusions

As in section 4, same conclusions are made with the exception of S0 state.

6 Proof of: Start Of Mission

6.1 Files used for the proof

Used node	Property file
Procedures::Procedure_StartOfMission	start_of_mission.hll

6.2 What we want to prove

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 modelized in the Scade model, communication with the driver doesn't seem to be implemented.

Moreover, an implementation of emergency brake management is present in the Scade model. This may be an implementation of "Standstill supervision". Emergency brake can be triggered at any time during the procedure execution and the train shall be at standstill to start the procedure.

States modelized are S0, state S10, states S20 to S25, states NL/SR/FS/OS/LS/SH/SN/UN mode and state See procedure ''Shunting initiated by Driver''.

A part of the actual flowchart is drawn by figure 3, only the part which is changed can be seen.

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".

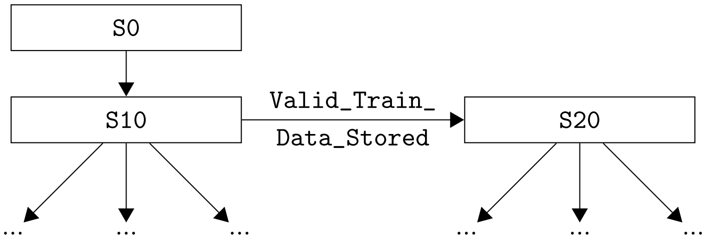

Figure 3: Part of actual flowchart of Start of Mission procedure.

Also, assumptions are made for this procedure:

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.

6.3 Results and conclusions

Considering assumptions made, the Scade model of Procedure Start of Mission does not correspond to the specification.

This is explained by the following differences between the specification and the Scade model:

1. The state `S0` is not specified as a usual state. It is specified as a condition that trigger the procedure. In the Scade model, `S0` is implemented as a condition that must be `True` to engaged the Start of Mission procedure. When the condition is `False`, the procedure is stopped.

2. Same observations present in section 4.3.

Here is propositions to resolve previous problems:

Proposition for 1. The ambiguity of `S0` definition must be clarified: either it is a state with a clear action in its definition, or it is a condition which trigger or not the procedure.

Proposition for 2. It is the same propositions that are made in section 4.3.

7 Proof of: Condition 1

7.1 Files used for the proof

Used node	Property file
ManageModes	condition_1_poof.hll

7.2 What we want to prove

We want to prove that, from any mode, if condition 1 is True then the mode will switch to IS.

7.3 Results and conclusions

Scade model corresponds to the specifcation of condition 1 (cf. Transitions Table).