System Validation and Testing

Aleksandr Popov
0924595
a.popov@student.tue.nl

Devin van Broekhoven
0839004
d.v.broekhoven@student.tue.nl

Jiddo van Vliet
0821894
j.g.v.vliet@student.tue.nl

Yuntao Li
0910663
y.li.2@student.tue.nl

Dan Cristian Chirascu
0923784
d.c.chirascu@student.tue.nl

Group 17

April 14, 2015

Department of Mathematics and Computer Science

1 Introduction

This document discusses the requirements for the future PP2 Assembly program code for the sorting machine project. It also describes in detail how the machine and software should work depending on given inputs.

2 Code Review

Code Review was conducted for both Java code and PP2 Assembly code. The results of several code review sessions are given below. Most of it was done by means of a simple walkthrough for both Java and Assembly code.

2.1 Java Code Review Summary

The review was done by means of pair programming on 31.03.2015 and by means of walkthrough on 07.04.2015.

Pair programming:
D.C.Chirascu wrote the draft. A.Popov optimized it in order to describe the various machine states properly and accurately by means of rewriting and expanding most of the code. D.C.Chirascu reviewed the result. No bugs were found, however, this code was never executed as it is only pseudocode.

Since our Java code is aimed to be the guidance for the PP2 Assembly code, Alex and Dan finalised it when the final draft of the Assembly code was done. They did not modify it anymore, as even though the PP2 Assembly code would be modified as a result of testing, the Java code had fulfilled its purpose of describing how the machine states work. These states do not change even though parts of the PP2 Assembly code were modified so that the machine would work properly.

There are some important improvements in the modification phase:

1. We merged the Start/StopButton class and AbortButton class because they share the same functionalities. We also merged the PreDetector class and SideDetector class due to their same functionalities.
2. We used enumeration to create State type variables that can have as values all our machine states (Init, SP, SD etc.):
   `private enum State {INIT, INIT1, INIT2, READY, SP, SD, SW1, SW2, SW3, SB1, SB2, SB3, SWS, SBS}`

   All machine state properties are explained in the software design document.

Walkthrough:
No corrections have been made, program appears to perform according to specification.

2.2 PP2 Assembly Code Review Summary

Most bugs were corrected before the program was first compiled by the author of the code, therefore, those code review sessions are not documented. However, there have been two review sessions by means of walkthrough:

- 01.04.2015 — Corrected wrong behaviour of PWM and Abort due to stack overflow.
- 03.04.2015 — Improved the black/white discs recognition and slightly tweaked both the code and the machine. No further bugs were found.

3 Test Cases

This report is aimed to deliver the result of testing cases, which describes the input sequence together with the expected outputs via testing. The testing is based on simulating the system with specific inputs and comparing the results produced by this system. The results should be equal to the expectation from the specification. In this project, a system under test will typically be a UPPAAL model. So these criteria were being tested:

3.1 Statement Coverage

For this criteria we need to show that every transition in the UPPAAL model has been executed. We just set the number of discs to 500 and randomized the transitions.

From the simulation trace we confirmed that every transition is enabled in the testing. (See Figure 1 in Appendix A: Figures.)

Statement coverage was also checked for the PP2 Assembly code. For that purpose excessive testing directly following the Use Cases was conducted by Aleksandr Popov and Devin van Broekhoven on 03.04.2015. Several mechanical issues were found and fixed. They have established that the machine performs according to prescribed use cases.

3.2 Condition Coverage

In this stage we need to show the truth value of every Boolean expression. The corresponding condition could always be fulfilled. In the previous section Statement Coverage we have already checked the transition of main control. Here we just focus on the other parts of the model which have the Boolean expressions, which are Belt Engine Control, Black/White Detector and Belt Controller. During the inspection no errors or exceptions were discovered. (See Figures 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 in Appendix A: Figures.)

3.3 Decision Coverage

Based on the result of Condition coverage, we use the simulator of UPPAAL to ensure that guards take all their possible outcomes.

- **BeltEngineControl**
  (See Figures 2, 3, 4, 5.)
  - ***Initial***
    The whole machine is in the resting state.
  - ***Condition 1***
    If the update is `mov_left=true` then the next state should be `Move_Left`
  - ***Condition 2***
    If the update is `mov_right = true` then the next state should be `Move_Right`

  When it reaches the state `Move_Left` (or `Move_Right`), as the boolean `mov_left = false` (or `mov_right = false`), or both Booleans turn to false, then it goes back to `Belt_Off` state.
- **Black/White Detector**
  (See Figures 6, 7, 8.)
  - ***Initial***
    The whole machine is in the resting state, there is nothing to detect.
  - ***Condition 1***
    If there is a white disc below the detector, we could get the update `curColor = false`. The next state will be White.
  - ***Condition 2***
    If we get the update `curColor = true`. The next state will be Black.
- **BeltController**
  (See Figures 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19.)
  - ***Initial***
    The whole machine is in the resting state.
  - ***Condition 1***
    If the belt moves to the left, then the Boolean `mov_left` should be true and `mov_right` should be false. If at the time when the `mov_left=true` and `mov_right=true` at same time, we need to rewrite the value of `mov_right` and then keep the value `true` for `mov_left`.
  - ***Condition 2***
    If the belt moves to the right, then the Boolean `mov_right` should be true and `mov_left` should be false. If at the time when the `mov_right = true` and `mov_left = true` at same time, we need to rewrite the value of `mov_left` and then keep the value `true` for `mov_right`.

3.4 MC/DC

In this part we mainly focus on the modification of Decision coverage. What we need to do in this stage is reverse the value we have used in Decision coverage. After that we compare the outcomes with what we have gotten from Decision coverage.

Since we have the states for whether a Boolean is true and false respectively, the outcome is the same as in Decision coverage.

4 Formal Proofs

In this section we need to ensure two properties:

1. There is no deadlock in the graph. There should not be any deadlocks in our graph, the transition should reach all the states and go to next state. (See Figure 20 in Appendix A: Figures.)
2. For PreDetector, if the checking procedure is fulfilled then the transition will reach the 'Ready' state. Since all the Boolean functions are verified in previous test, here we need to ensure the inputs are in position for the automaton. (See Figure 21 in Appendix A: Figures.)

Appendix A: Figures

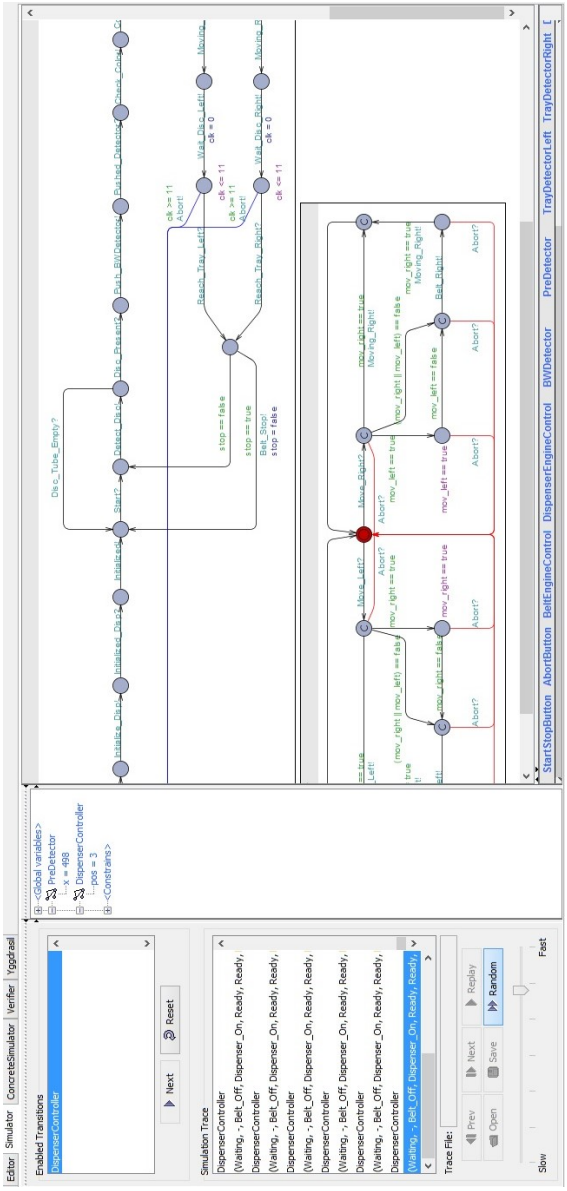

Figure 1: An illustration of simulation

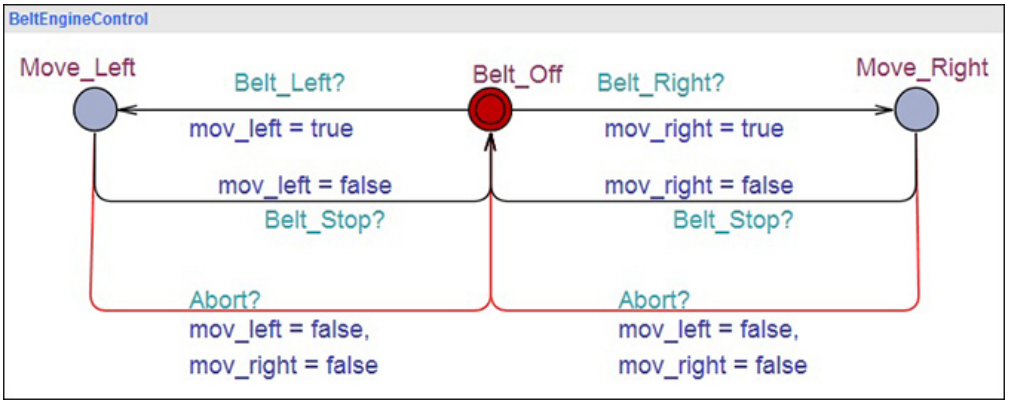

Figure 2: Resting stage of the belt

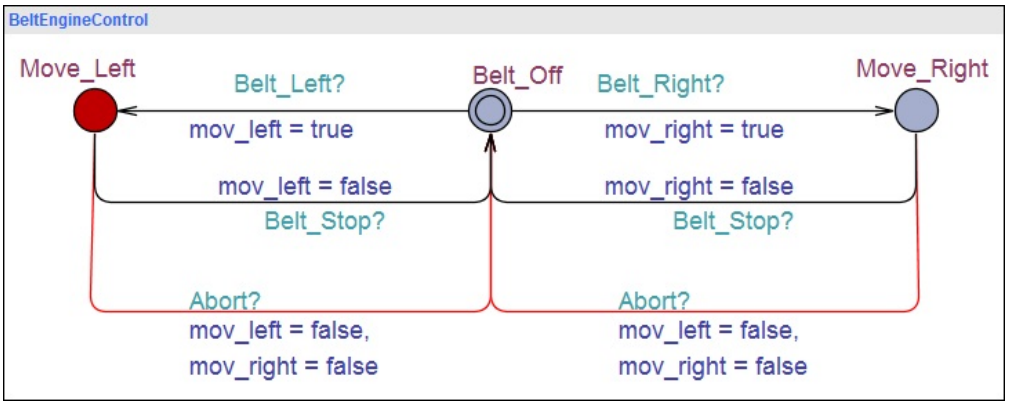

Figure 3: Belt moves to the left

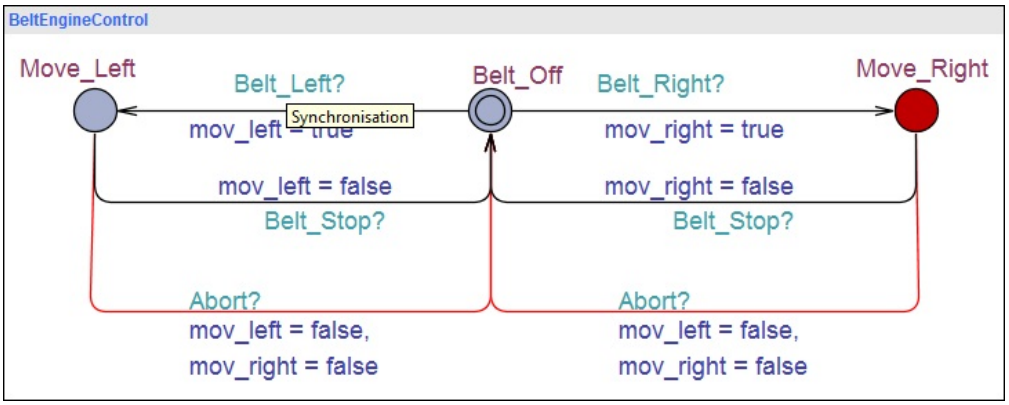

Figure 4: Belt moves to the right

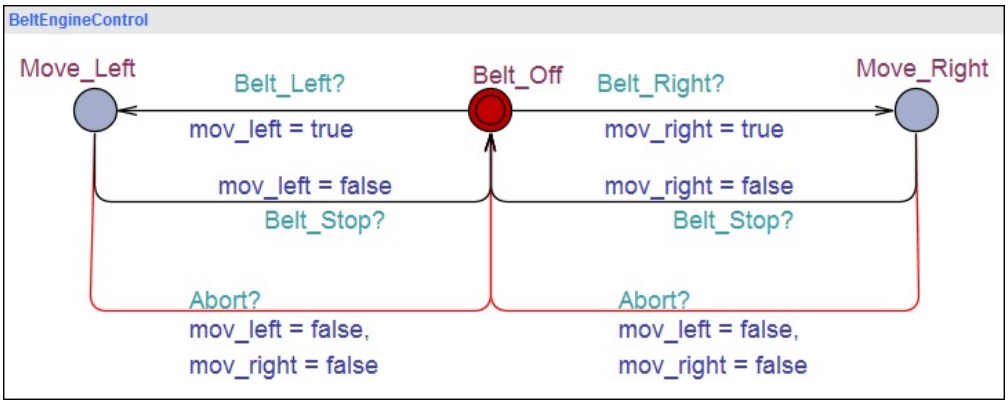

Figure 5: Belt stops

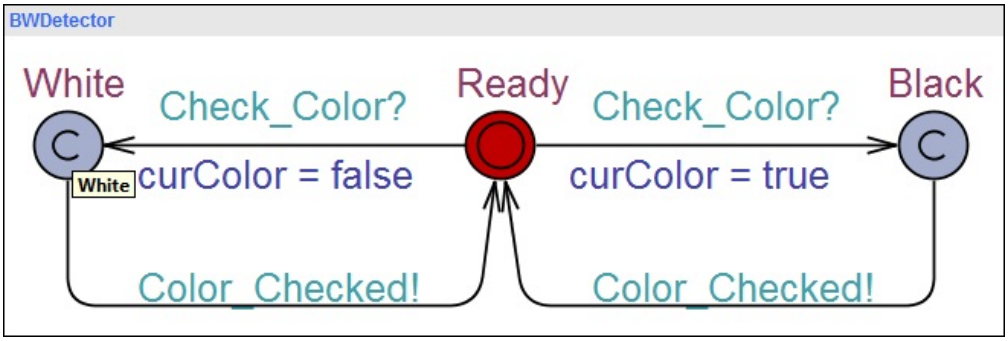

Figure 6: Resting stage of the Black/White detector

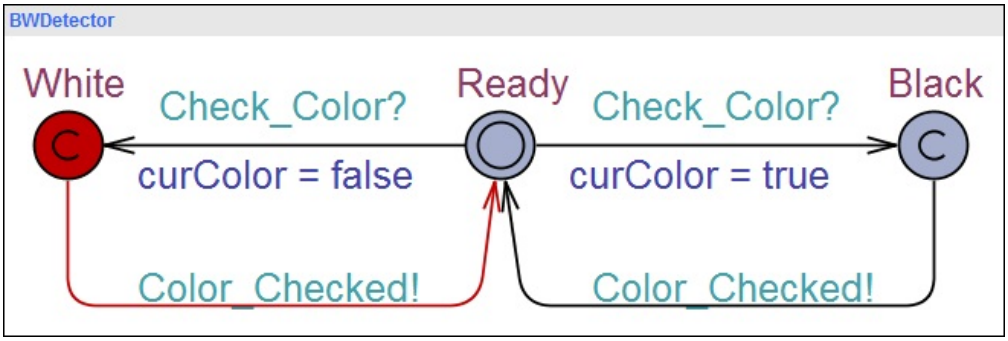

Figure 7: A white disc has been detected

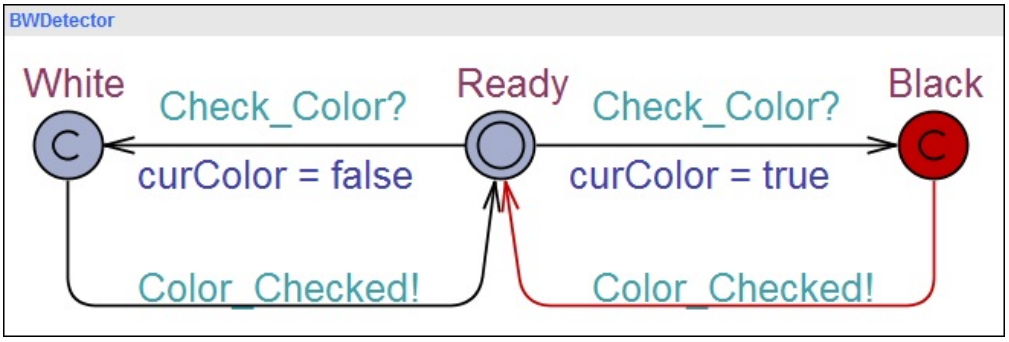

Figure 8: A black disc has been detected

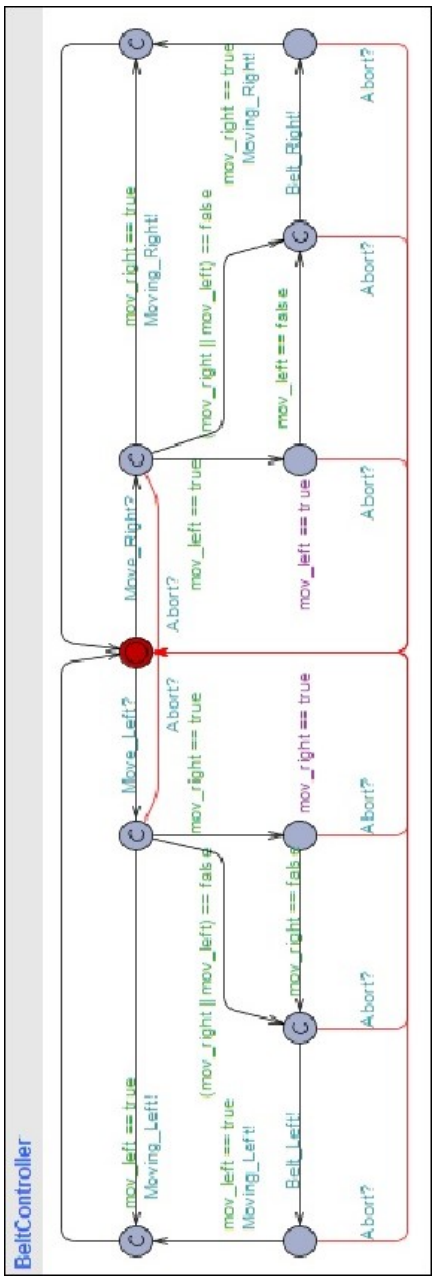

Figure 9: Resting stage of the belt controller

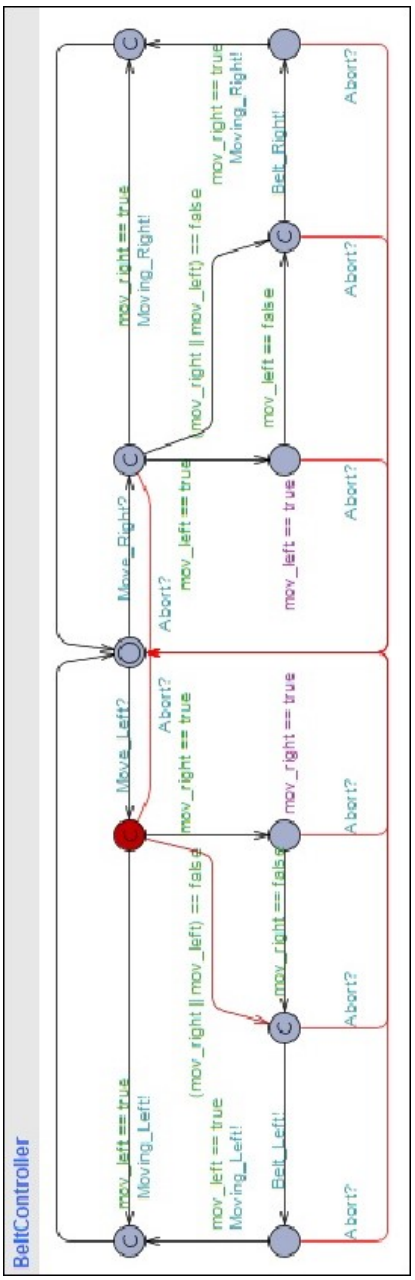

Figure 10: An order to make belt move left

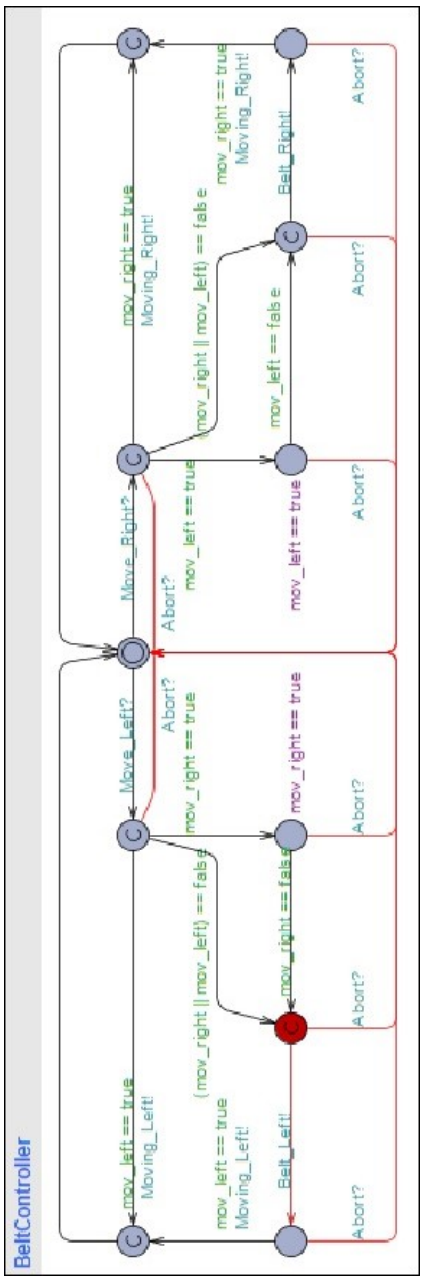

Figure 11: Check the current direction of the belt

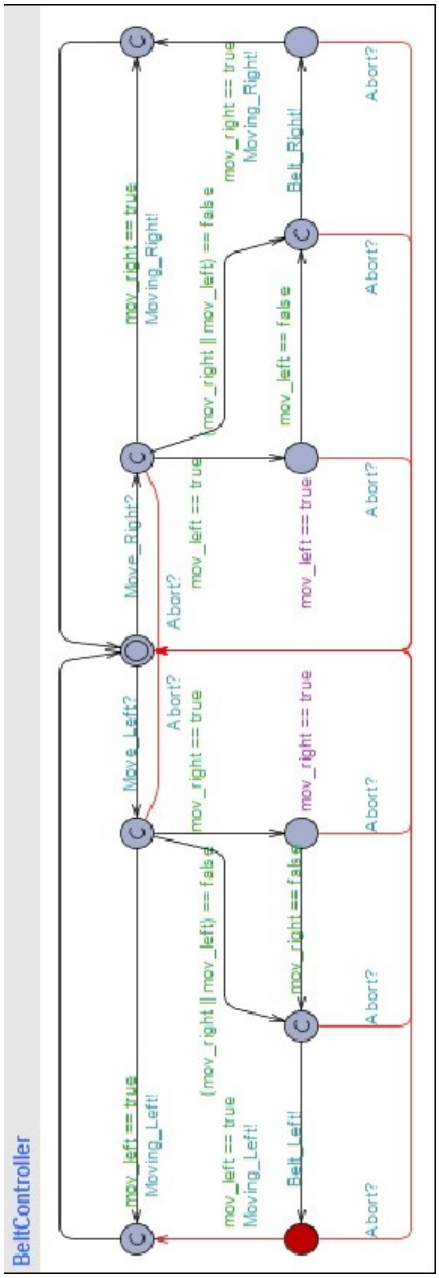

Figure 12: Fix the boolean value

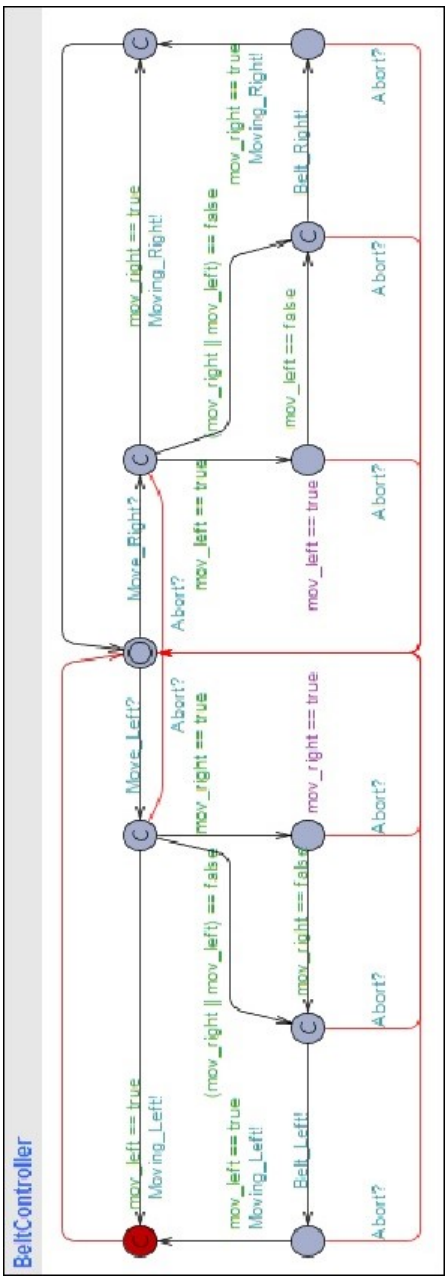

Figure 13: Move belt to the left

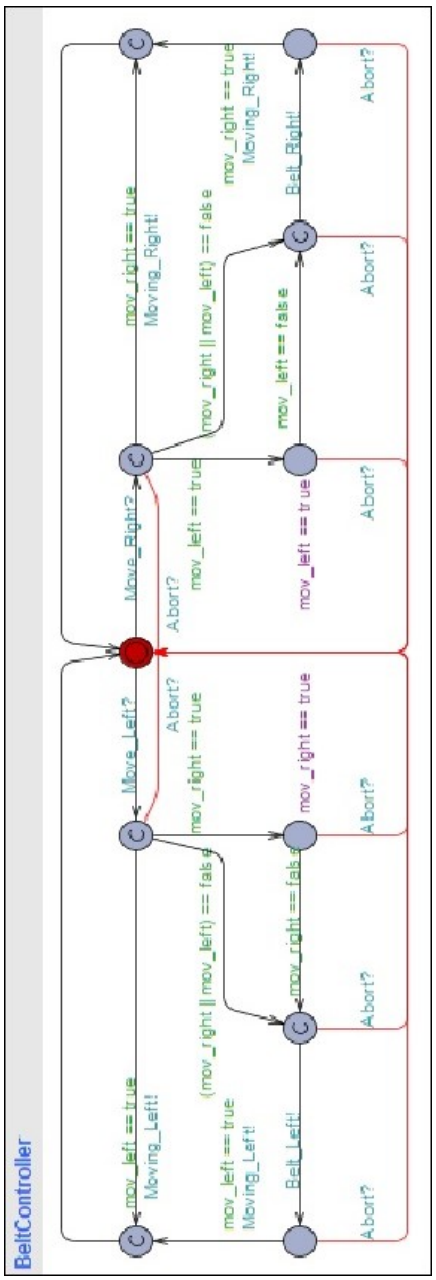

Figure 14: Movement finished

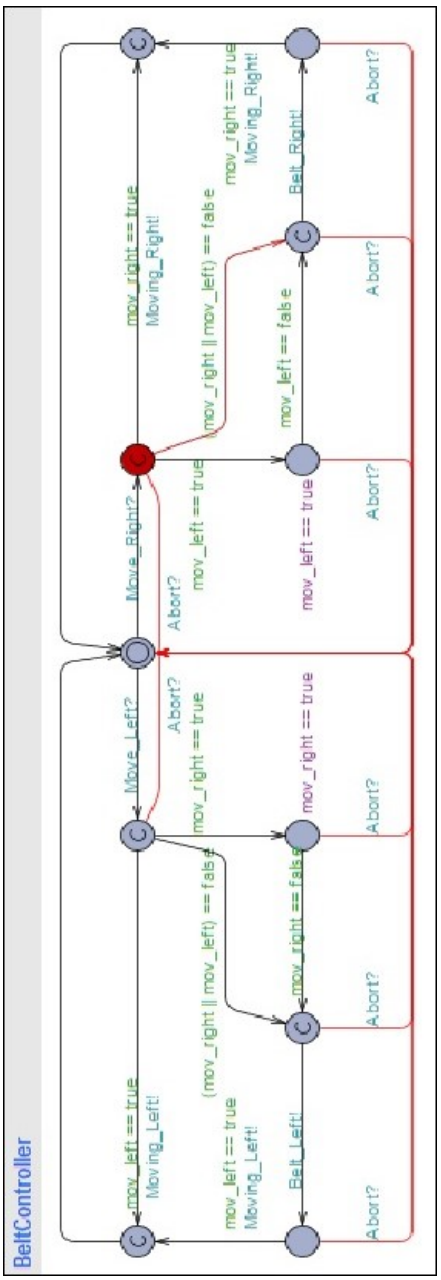

Figure 15: An order to make belt move right

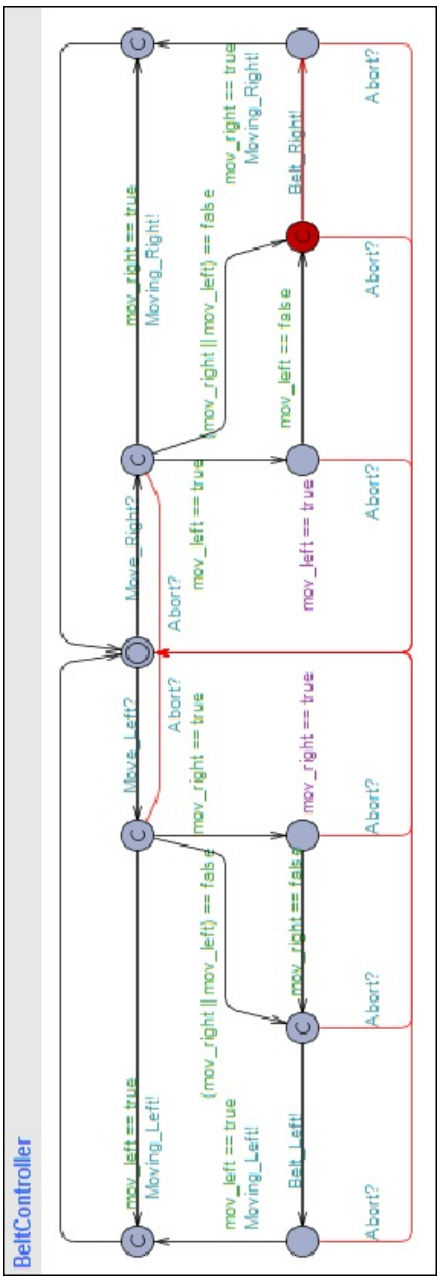

Figure 16: Check the current direction of the belt

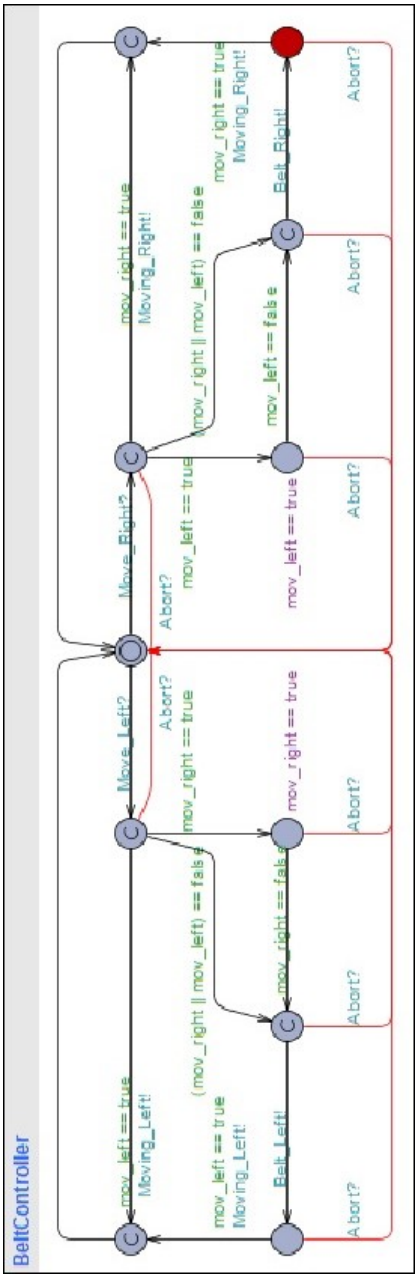

Figure 17: Fix the boolean value

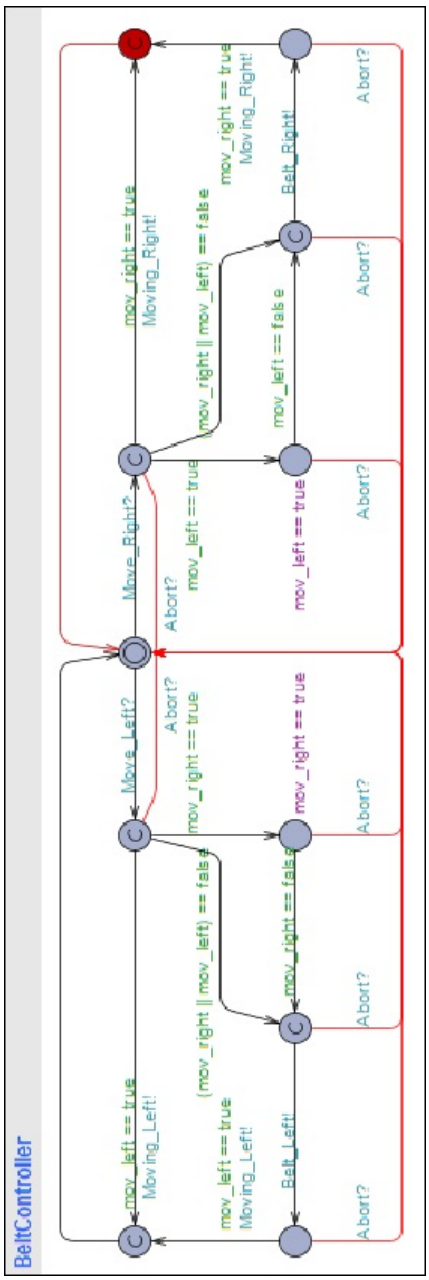

Figure 18: Move belt to the right

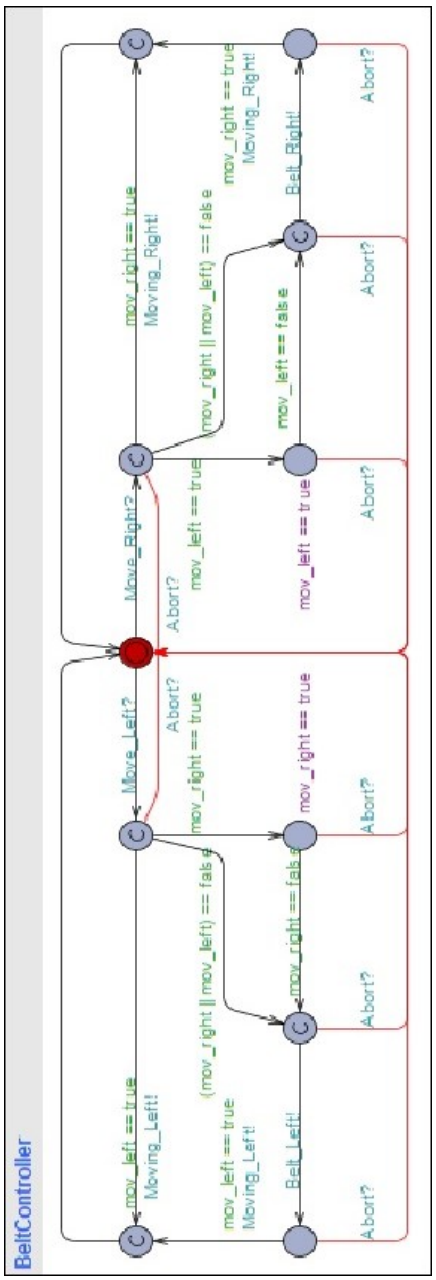

Figure 19: Movement finished

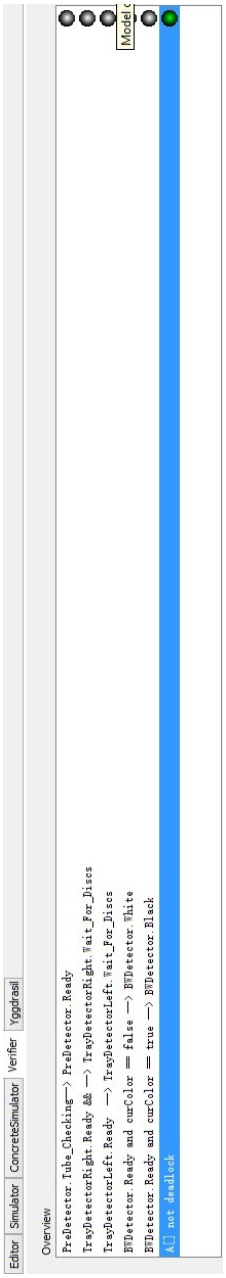

Figure 20: Verifier 1: No Deadlock

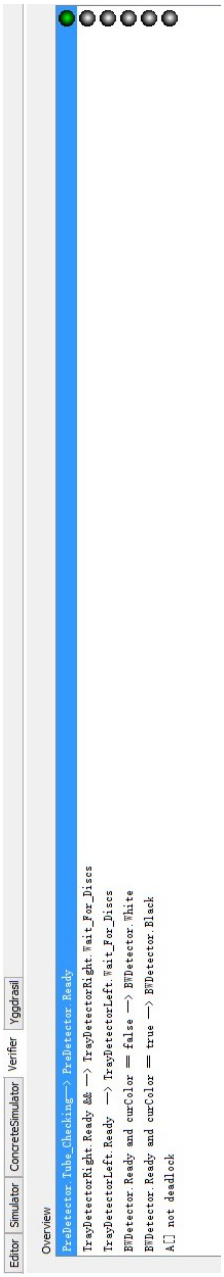

Figure 21: Verifier 2: Check PreDetector