POLITECNICO
MILANO 1863

Formal Methods
For Concurrent and Real-Time Systems

Formal Digital Twin of a Lego Mindstorms Production Plant

Model Checking with UPPAAL

Authors:
Alex Amati – alex.amati@mail.polimi.it – 103174
Luca Molteni – luca9.molteni@mail.polimi.it – 103072

Instructors:
Prof. Pierluigi San Pietro
Dr. Eng. Livia Lestingi

Academic Year: 2022–2023

1 Introduction

This project's objective is to model a Lego Mindstorms production plant using UPPAAL, allowing for effective simulation. The plant's execution will be simulated by tracking the movement of an object through the plant's various stages.

2 Design

The design of the model is composed by these main components:

- a Conveyor Belt;
- the Objects;
- the Slots;
- the Processing Stations.

The Conveyor Belt component instructs the other components about when a conveyor belt `shift` is happening, the time duration between `shift`s is set by the `inverseSpeed` constant.

The Object component represents an object traveling along the plant, it interacts with the components it passes through.

The Slot component represents the single slot of the conveyor belt.

The Processing Station component represents a crucial stage in an object's journey around the plant. When an object reaches this station, it undergoes processing and is temporarily halted for a specific wait time.

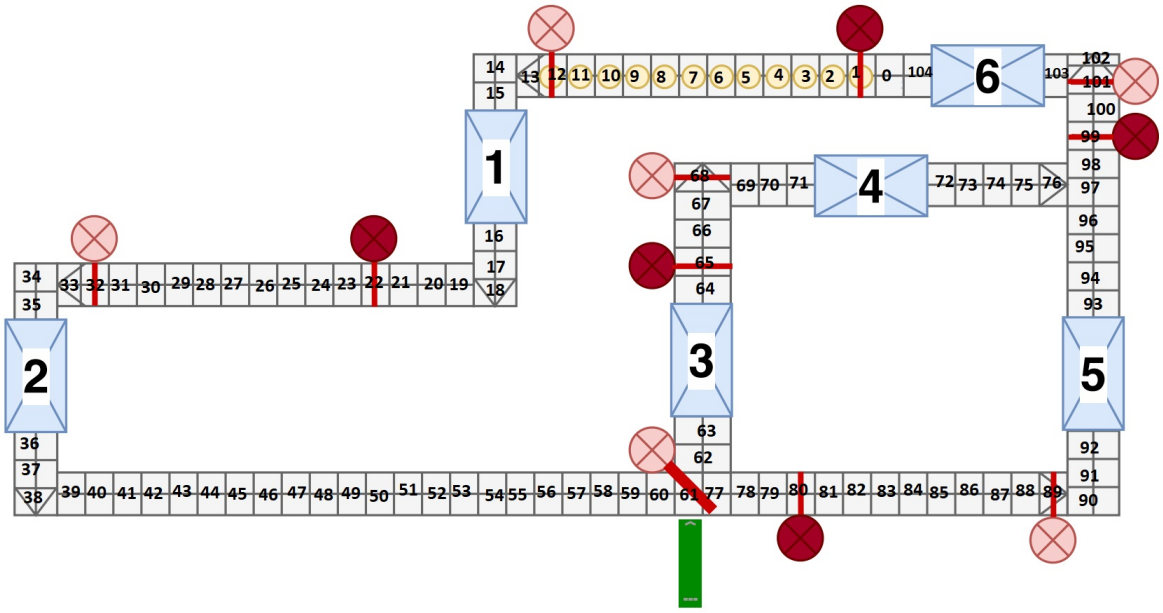

Figure 1: Component enumeration

3 Global Declarations

In this section some configuration parameters are declared to make the model a little more versatile:

Parameter	Description
`inverseSpeed`	The frequency at which the belt moves
`PS_TIME`	The processing time of each Processing Station
`SENSOR_QUEUE_INDEXES`	The indexes of slots containing sensors guarding the queues of the processing stations
`SENSOR_PS_INDEXES`	The indexes of slots containing sensors guarding the entrances of the processing stations

Table 1: Global Declarations

There are some other parameters that define the path and its components (i.e. processing stations and branch's positions). These parameters and functions are omitted here but can be reviewed in the code along with their comments. This approach guaranteed easy change between different plant configurations.

3.1 Stochastic

In the stochastic version there are additional parameters to specify the probability of failure of the sensors and the `PS_TIME` is replaced by `PS_TIME_MEAN` and `PS_TIME_STD`. The other parameters are the same.

Parameter	Description
`PS_TIME_MEAN`	The processing time mean of each Processing Station
`PS_TIME_STD`	The processing time standard deviation of each Processing Station
`SENSOR_QUEUE_FN_WEIGHT`	Probability of false negative for sensor guarding the queues of the processing stations [‰]
`SENSOR_QUEUE_FP_WEIGHT`	Probability of false positive for sensor guarding the queues of the processing stations [‰]
`SENSOR_PS_FN_WEIGHT`	Probability of false negative for sensor guarding the entrances of the processing stations [‰]
`SENSOR_PS_FP_WEIGHT`	Probability of false positive for sensor guarding the entrances of the processing stations [‰]

Table 2: Stochastic Global Declarations

4 Components Breakdown

Here is a high level description of how all components operate and interact with one another.

4.1 Conveyor Belt

This is a component that dictates when all the objects should move between components (the conveyor belt has fixed velocity in all areas of the plant), this component has a constant that determines the frequency rate at which the belt `shift` should happen (`inverseSpeed`).

4.1.1 Automaton

The conveyor belt component is composed of a single state. To make the `shift` happen exactly every `inverseSpeed` time there is an invariant on such state that obligates the state to take its only arc before or at `inverseSpeed` time. Its only arc handles the clock reset and can be taken only at clock greater or equal than `inverseSpeed`. This combined with the invariant on the state constrains the simulation to take the arc at exactly `inverseSpeed` time. This arc when crossed fires the `shift` signal.

4.1.2 Code

This code is composed by a clock `t` and a method `setBranchSwitch` that handles the controller switch at the branch at slot 61, the policy that applies is a switch at each `shift`. This can be customized with another expression.

4.1.3 Stochastic Version

The stochastic version of this module has a random branch switch policy. It differs also by the fact that it has two nodes: the first node is added for array initialization reasons and it triggers the `initialize` method. At every `shift` this version of the component also randomizes the false positives events of both types of sensors.

4.2 Object

In this module is modeled the flow of actions performed by an object:

- entrance in slot;
- exit from a slot;
- entrance in a processing station;
- wait for processing inside a processing station;
- exiting from a processing station.

4.2.1 Behaviour

The behavior of this automaton starts with the entrance in slot 0 (given of course that this slot is free). If then the next element of the path is a slot and the slot is free (or it is ready to become free) it occupies that slot and at the next shift it switches its position to there. It also exits the slot it was in. Same goes if the next element is a processing station, it occupies that station and at the next shift it enters the station while exiting the previous slot. Then it waits for the station signal that the processing has finished and at the first shift it moves to the next slot.

4.2.2 Automaton

This automaton is divided essentially in three parts, the first part handles the initial insertion into the belt, the second part handles the shift if the next component is a slot and the third part handles a shift if the next component is a processing station.

NB: The automaton does not handle the cases of initial insertion into a processing station and processing station to processing station shift.

4.2.3 Code

In this component there are two boolean variables that signal the presence of the object inside a slot or a station along with two identifiers for such components.

4.3 Slot

The slot component handles the request for occupancy, the subsequent transition along the slot and the subsequent request to be freed to allow a new object to come through.

4.3.1 Automaton

The automaton for this component starts waiting for a request to be occupied, it then waits a shift to enter the occupied state and frees the slot at the next shift whenever the object component calls for it. It then can be occupied right away or set free. This automaton has some more details that handle the sensor signaling, explanation for this part is in section 5.3

4.3.2 Code

The code for this component involves support methods in order to keep the automaton a little bit cleaner.

4.3.3 Stochastic version

The only substantial difference in the stochastic version lies in the management of the sensors, resulting from the consideration of failure probabilities.

4.4 Processing Station

While fairly similar to the slot component the processing station adds processing time to passing objects, the processing time can be set in the configuration for each processing station in the belt.

4.4.1 Behavior

Once the signal `busyPS` is triggered it waits a `shift` signal to start counting for the processing time. Once it has completed processing it signals to the object component that it has ended. There is another condition for the object to remain halted, if queue sensor of the next station signals that the queue is full it waits until the queue is gone to release the object. In the end it waits for the signal coming from the object to free the station.

4.4.2 Code

The code involves only simple support methods to check on sensors.

4.4.3 Stochastic Version

The stochastic version of this component only changes its configuration parameters to be the mean and variance of the processing time. This is done by adding a simple method called at station occupancy that changes processing time using included math functions to compute normally distributed values.

5 Design Choices

Hereby are presented the choices made to develop the model and how these choices reflected on the simulation.

5.1 The Belt Shift

To model the synchronized movement of all the objects, the ConveyorBelt component sends a signal on the shift channel periodically and all the objects and the other components wait for that signal before proceeding in certain locations. Between shifts the Object component attempts to occupy the next slot (or station) by sending out a `busySlot` (or `busyPS`) signal. This signal can only be received by the following belt component. If the signal successfully passes through, it indicates that the next component is available and the object can notify the previous component that it is about to be freed by sending a `freeSlot` (or `freePS`) signal. This ensures that the objects can move in a sequential manner, with each object positioned right behind the other. During the next shift, the positions are updated, and it is possible for other objects to have entered the previous component. This approach guarantees also that the objects will automatically form a queue behind any halted piece.

5.2 Leaving a Processing Station

In order to maintain the previous strategy, a double signaling system between the Object component and the ProcessingStation component is introduced. This involved the station sending a signal indicating the completion of object processing. Subsequently, another signal is sent to try to occupy the next slot. If this second signal is successfully sent the object sends a signal back, indicating that the station is about to be freed.

5.3 Sensors

The sensors are modeled using boolean array variables, this is done to avoid further complicating this model with other signals. If a slot is responsible for stopping components before a processing station it sets the variable `sensorPS` when an object is detected and prevents the object from moving until this variable is set to false by the processing station. There is a second sensor component that sets the `sensorQueue` variable when an object is halted in a position for any reason, this sensor is present in every slot, this is done to let the verifier check for halted pieces easily, even if there's no need for it in the model.

5.4 Stochastic version notes

In the non-stochastic version the most of the synchronization is done exploiting the fact that through non broadcast channels both the edge that sends the signal and the one which receives it needs to wait one for the other. So, due to the fact that in the stochastic version all the channels need to be broadcast, boolean variables were added, these are used by the edges which receive the signals to tell the senders that they are ready to take the transitions which receive them.

6 Verification

6.1 Queries

These are the 4 required queries:

1. it never happens that a station holds more than one piece;
2. it never happens that two pieces occupy the same belt slot;
3. no queue ever exceeds the maximum allowed length;
4. the plant never incurs in deadlock.

The first two relied on the object component variables: the booleans that signal the presence inside a station (or a slot) and the ids of those.

The third uses the sensors before the slots that had a queue sensor to see if they were occupied by a queued piece (these are the sensors present in every slot as explained in section 5.3).

The fourth one is done by stating that an object that enters the plant has to always get at the end of it and be able to restart at slot 0. The deadlock construct of UPPAAL was avoided for this check because it wouldn't detect the situation where the ConveyorBelt was looping but all Objects were stalled in their position.

Some further checks were added:

- A test that checks that the ConveyorBelt component continues looping;
- A test that checks that the plant does not reach a deadlock state (this is already checked by the previous test but it was added anyway);

6.2 Tests

To verify the model three different configurations were built:

1. one with the components at their intended place (like Figure 1), inverseSpeed at 10 and processing station time at 1;
2. one with queue lengths set to one (except for the first processing station that remains like Figure 1), processing station time set at 100 and inverseSpeed at 1;
3. one with sensors for the queue kept at one and processing station time kept at 100 but the first processing station time is changed to 1.

In all tests 3 objects are used, resulting in a very long computation time but allowing to create certain situations that with only two objects couldn't have been possible. The branch policy is set to switch the branch at every ConveyorBelt shift.

In test 1 all queries passed implying that the model was working as intended.

Test 2 was specifically designed to see if the queues limited to one piece could have impacted constraint 3 but in this case nothing happened. This called for test 3. Indeed, test 3 failed, revealing a case where two pieces were halted behind station 2. This is possible due to the fact that if an object arrives at the queue sensor when an another object is already released by the station beneath it will queue up behind the first one.

These tests had the first queue sensor at its original position, this is done because the objects are inserted at slot 0 and a failed test here wouldn't have highlighted the problem correctly.

In all the tests all the other queries were successful.

7 Stochastic Verification

7.1 Queries

These are the queries present in the verifier:

1. probability that never happens that a station holds more than 1 piece;
2. probability that never happens that two pieces occupy the same belt slot;
3. probability that no queue ever exceeds the maximum allowed length;
4. probability that at least one object exits the first processing station;
5. probability that at least one object arrive at the end of the conveyor belt;
6. expected value of the maximum number of objects in the queues;
7. expected value of the minimum number of objects in the queues.

In the concrete simulation it seems that an object could not exit a processing station (causing a deadlock) for no apparent reason. So the fourth query was added to ensure the problem does not affect the verification of the other queries.

7.2 Tests

To check the model behavior three different configurations were tested:

1. one with the fault probability of the sensors at 0;
2. one with the fault probability of the sensors guarding the queue of the processing station at 10‰ ;
3. one with the fault probability of both types of sensors at 10‰.

All these tests were runned with the sensors as in Figure 1, 3 objects, processing time in the order of hundreds and frequency of the belt at 10.

Query	Test 1	Test 2	Test 3
1	≥ 0.950056	≥ 0.950056	≥ 0.950056
2	≥ 0.950056	≥ 0.950056	≥ 0.950056
3	~ 0.944646	~ 0.953848	≥ 0.950056
4	≥ 0.950056	≥ 0.950056	~ 0.874266
5	≥ 0.950056	≥ 0.950056	~ 0.0553542
6	~ 2	~ 2	~ 3
7	~ 0	~ 0	~ 0

Table 3: Stochastic Verification Results

From the similarity of values between test 1 and test 2 it's possible to see that a fault of a sensor guarding the queue of a processing station doesn't really affect the correct proceeding of the system. This is because those sensors are modeled in the way that if one of them fails at the next movement of the belt the sensor, if it doesn't fail again, will correct the problem.

Instead, in the third test's results the probability of at least one object to exit the first station and for at least one object to reach the end of the belt are significantly lower than in the previous two tests. This discrepancy is attributed to the potential failure of a sensor that guards the entrance of a processing station. If such a sensor were to malfunction and wrongly detect an object passing through when there is none, it would not be able to perform a second check after the conveyor belt movement because the object would be in a different slot. Consequently the problem would remain unresolved and the sensor would block all objects attempting to enter the station until it is eventually notified of an object exiting, which, in this case, may never happen. Also, confirming this, the expected value of the maximum number of objects in the queues is 3. This means that all the objects can be waiting for a processing station to finish, without an object being processed.

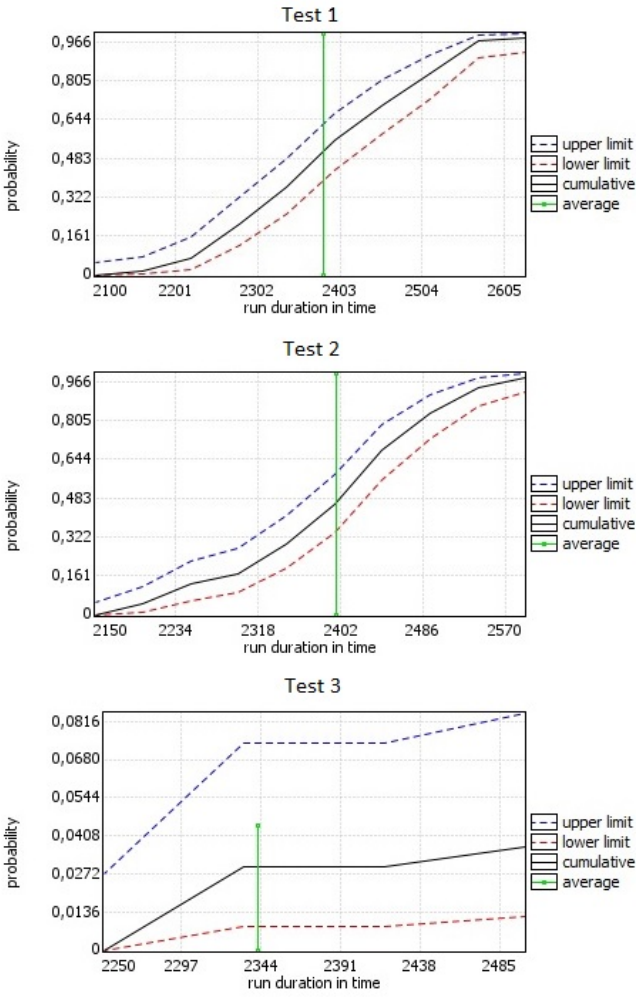

Figure 2: Cumulative probability that at least one object gets to the end of the belt