POLITECNICO
MILANO 1863

Formal Digital Twin of a LEGO® MINDSTROMS™ Production Plant

Formal Methods for Concurrent and Real-Time Systems

A.Y. 2022-2023

Andrea Infantino
Person ID 10671720
Student ID 225757

Riccardo Motta
Person ID 10658639
Student ID 218685

Matteo Negro
Person ID 10642961
Student ID 222025

Contents

1 Model Description

This section provides a description of the production plant with some implementation details, together with all the components that make it up. The motivations behind these modelling choices can be found in Section 2: Design Decisions (page 6).

1.1 The Production Plant

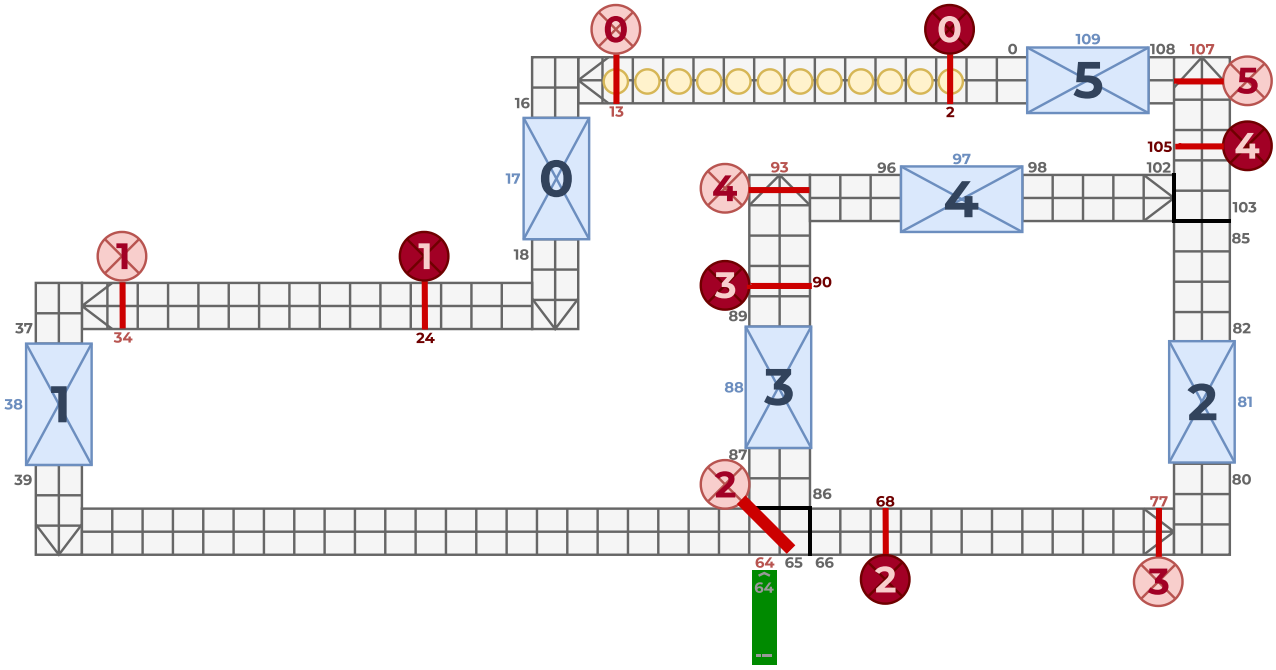


Figure 1: the production plant we modelled.

This is the plant we modelled in our project. The yellow circles are the workpieces, the blue rectangles are the processing stations, the light and dark red circles are the sensors and the green bar is the flow controller.

Notes on the scheme We enhanced the original scheme with the number the various stations and laser sensors have inside our project and with the numbers of the positions of the conveyor belt, in order to respect what we have done inside the project.

1.2 General Overview

The model of our system is made of 6 different components which interact between them in order to coordinate the entire production plant. Some of them are also instantiated many times in order to have a simpler modelling of the entire system.

Initializer It's a fictitious component which represents the entry point of the whole system. It allows us to instantiate all the workpieces in the correct positions of the conveyor belt (positions which start from position 13 and ends in position 2 in our model) and works as the general clock of the system.

Motor This is the real motor of the system. According to the speed at which the conveyor belt should be moving, uses the clock to make all the workpieces move and synchronizes the whole system.

Conveyor belt It's the set of all the various conveyor belts of the plant. It's the one in charge of moving around all the workpieces. In our model it's also the one in charge of blocking workpieces and stations from proceeding according to the pieces of information gathered by the laser sensors and the stations and also manages the positions where the two branches of the plant start and merge.

Processing stations They are the ones in charge of processing the various workpieces one they get into them. We have a single template for them which is instantiated as many times as needed in order to recreate the plant. They have all a position on the conveyor belt which is managed in a special way.

Laser sensors We have modelled them in two different ways according to their functionalities. The ones guarding the entrance of a station are called `InSensor`, while the ones guarding the queue and preventing the station right before it to release a workpiece are called `OutSensor`. Like the stations, they are represented as a single template (one for each type) instantiated multiple times.

Flow controller This is the green piece of Figure 1. It is pre-configured with a specific policy (which is customizable) and decides whether to send the workpieces once they get at position 65 of the conveyor belt. The available policies are the following ones:

0. Sends the workpieces on the alternative branch until its queue is full, otherwise lets the workpieces proceed on the main branch;
1. Lets all the workpieces to flow on the main branch;
2. Sends all the workpieces on the alternative branch;
3. Every time a workpiece is in front of him sends it on a different path with respect to the one taken by the previous workpiece.

1.3 Initializer (`Initializer`)

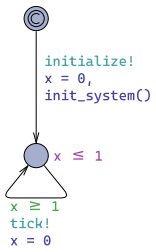


Figure 2: the Timed Automata of the initializer.

The initializer is a meta component that performs two different actions on the system and is the fundamental entry point of our whole modelling.

Initialization Initializes the system by placing the workpieces starting from the in sensor of the first station (index 0) backward with an upper limit of 12 workpieces.

Global synchronization In order to have a simpler model to verify, we decided to have a unique clock for the whole system. This TA performs as the general clock and all the components of our plant directly or indirectly synchronizes with it (usually by means of the signals generated by the motor).

1.4 Motor (`Motor`)

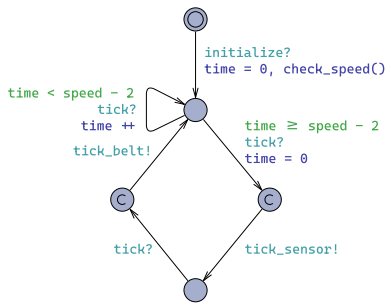


Figure 3: the Timed Automata of the motor.

As the name says, this is the meta component that manages all the signals used by the sensors and, especially, the conveyor belt and the stations in order to synchronize the movement of the workpieces. The two signals which it generates are presented here.

Conveyor belt's tick (`tick_belt`, also used by the stations) It's the signal which makes the conveyor belt move all the workpieces it can one step forward (taking into account also the branch in the conveyor belt).

Laser sensors and stations' tick (`tick_sensor`) In order to avoid any potential race condition, we decided to send a signal right before `tick_belt`. This allows us to update the state of the laser sensors (i.e., they can check if there is a workpiece right below them) and to perform some state-changing actions on all the stations (if they need to).

1.5 Conveyor Belt (`ConveyorBelt`)

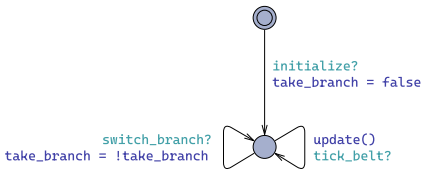


Figure 4: the Timed Automata of the conveyor belt.

It's the entity which collects all the different sectors of the plant's conveyor belt. It's a small automaton since all the underlying complexity of the conveyor belt's management is hidden behind the `update()` function. This is the most complex component of our plant since manages a lot of different things.

Workpieces position Any workpiece on a non-critical position (which is any in the following situations) is moved ahead of one slot every time the `update()` function is called.

Stations and sensors management Once a workpiece passes the input sensor of a station, it blocks any other workpiece wanting to proceed to the station itself until the sensor tells that the station is free and a new workpiece can flow again towards it.

Queue management Every time a workpiece is blocked in a specific location and some other one wants to proceed, the conveyor belt blocks it until the first one is free to move again. Since we are scanning the belt from the last positions backward this situation is easily managed since we can't place a workpiece on top or another one.

Conveyor belt branch and merge Because of the `switch_branch?`, we keep track of the selected branch where to send the workpieces. This allows us to directly use the `update()` function to move the workpieces on the correct branch. Also, the merge of the two branches is (easily) managed by the conveyor belt, since, by scanning it backward, once we move a piece in the merge position of the belt (the one coming from the alternative branch, if any), no other one can be put in the same place, thus, blocking it from proceeding, which is exactly what happens in reality.

1.6 Processing stations (`Station`)

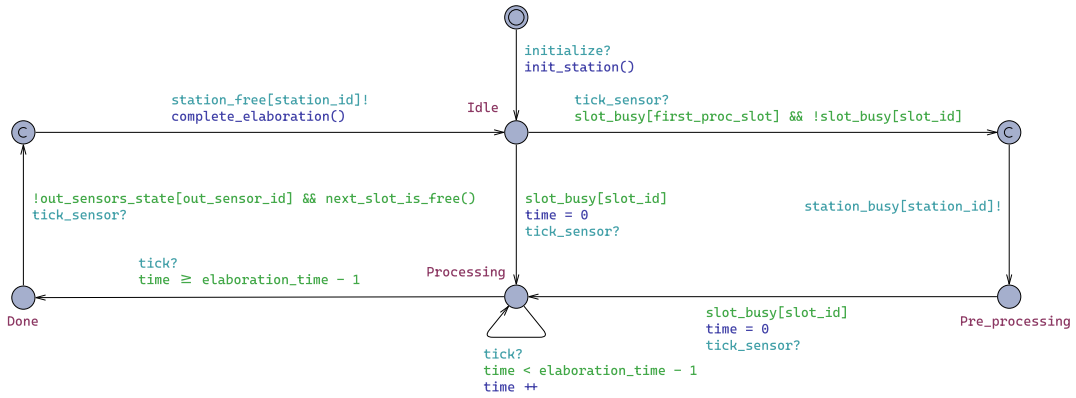


Figure 5: the Timed Automata of every station.

The stations model the various processes the workpieces need to go through during production. The automaton is divided in four different main states, each one of the for a different portion of the processing.

Idle This is the main state of the station, in which it doesn't have to perform any action but to wait for a workpiece to be ready for the processing.

Preprocessing When the laser sensors which is guarding the entrance of the station signals that a workpiece is approaching to it, the station moves to this state waiting for the workpiece to be in the right position.

Processing Once the workpiece is in the right position, the station starts processing it. In the model, the automaton isn't performing any real task on the workpiece, but it's simply waiting for its processing time to pass. In order to improve verification performance, we discretized the time, and so we use a counter synchronized with the global clock in order to model the time passing.

Done Once the processing time is elapsed, the station moves to this state where it waits for the right time to release the workpiece on the conveyor belt. It synchronizes with the conveyor belt itself and with the sensors guarding the entrance queue of the following station in order to know if the workpiece needs to be release or hold inside the station itself. Once the workpiece is released, the station returns to its *idle* state waiting for the next processing cycle.

1.7 Processing station's input guard (`InSensor`)

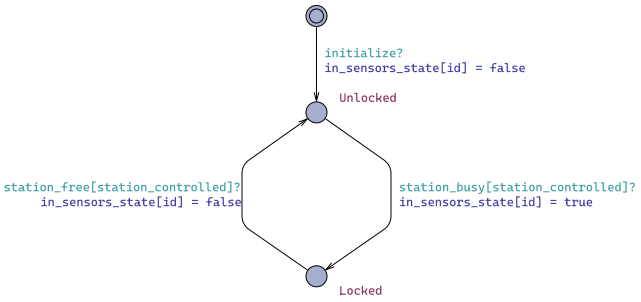


Figure 6: the Timed Automata of every input sensor.

These are the laser sensors that guards the entrance of a specific station. If a workpiece oversteps it, the laser sensor is equipped with a physical barrier which blocks any other following workpiece from proceeding towards the station itself until the station is free again.

1.8 Queue's guard (`OutSensor`)

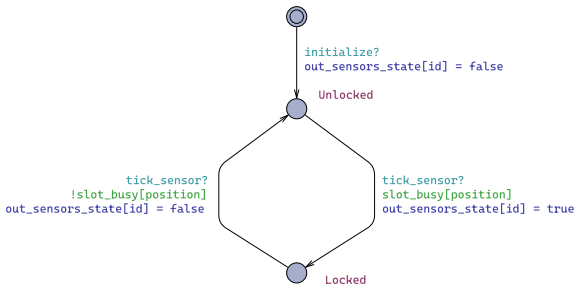
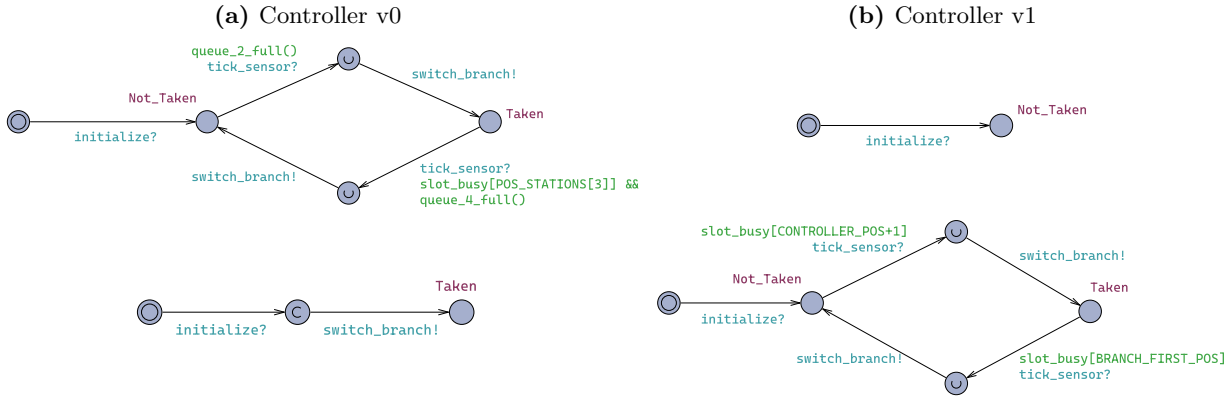


Figure 7: the Timed Automata of every output sensor.

Right before a station there is an `InSensor` which guards its entrance, but there may also be another sensor before it which guards the queue. Once the station's guard blocks a workpiece, others may arrive forming a queue of workpieces. This queue may have a maximum length. If the queue reaches its maximum, the sensor blocks the preceding station from releasing a workpiece if it contains one until the queue is no more full.

1.9 Flow controller (`FlowController_*`)



(a) Controller v0 **(b)** Controller v1

(c) Controller v2 **(d)** Controller v3

Figure 8: the Timed Automata of the various types of controllers.

As explained in the introduction, these are the various policy schedules we implemented. Each one of them represents a different way to schedule the workpieces on the processing stations. Starting from the easiest ones, the ones in which all the workpieces follow the same path (either the direct one or the alternative one), to more complex ones, like the one in which we keep track of the queues in front of the first stations of the branch.

Since the real movement of the various workpieces is managed by the conveyor belt, these automata, if they need to change the branch where the workpieces need to be sent, send a message to it in order to notify the change in the path.

2 Design Decisions

2.1 Hypothesis

TODO Hypothesis (numbering, flow controller, maximum number of disks...).

2.2 Optimizations

Here we present all the main optimization steps we performed in order to have the final model.

2.2.1 The basic idea

Idea At first, we decided to model every single entity of the plant. We had the template for generating all the single slots of the conveyor belt. This has been done in order to have a precise (and discrete) representation of the plant pointing out all the atomic components that we have.

Situation The main issue with this representation is the fact that by doing this, we had a huge verification time created by the huge number of entities and states the software had to process each time.

2.2.2 Unique conveyor belt

Idea In order to simplify the modelling of the plant and our code, we decided to group all together the various slots of the conveyor belt into a logically single entity (the `ConveyorBelt`).

Situation This simplified a lot the management of all the various cases (and indeed is still present in our project) since we have a unique entity which manages all the complexity deriving from the movement of the various workpieces. This also works in synchronicity with the various processing stations and sensors in order to block the pieces when needed (and not only when there is a workpiece in front that doesn't move) and with the flow controller which tells the `ConveyorBelt` where to send the workpieces on the branch. By doing this we also improved a lot the verification time needed to check the properties, but we still had a huge memory consumption.

2.2.3 Unique clock

Idea By talking with some colleagues and with the instructors we realized that we had 7 different clocks inside our project (one for the general clock of the system and one for each station) and that they tend to be one of the biggest bottlenecks during the verification phase, we decided to reduce them to just a general one, and to have all the various templates synchronizing with it.

Situation We added a new template, the `Motor`, which sends the right signals at the right moment to all the components that need it respecting the speed of the conveyor belt's movement, and we synchronized the stations' processing times with the general clock of the system (which is not related to the speed of the conveyor belt). By doing this we reduced the number of possible transitions the system can take at any instance, and we make sure that everything is synchronized with just one single entity. This allowed us to have a skyrocketing verification time, even with 12 different workpieces circulating in the production plant at the same time.

3 Properties

TODO Properties we are verifying (in TCTL and not in UPPAAL's language).

4 Stochastic version

TODO Explain it and point out the differences with the automata we changed.

5 Scenarios

These are some possible scenarios which want to show that the production plant we modelled works correctly in any (normal) situation. There are some pathological cases in which the plant fails to guarantee some properties, but it's completely normal since they are all pathological cases. Some of them are depicted here.

5.1 Scenario 1: the normality

This is a general scenario in which the structure of the plant is the same as the one depicted in the introduction. The stations' processing times have been chosen without any specific criterion, and we instructed the flow controller to equally send the workpieces on the two different branches of the conveyor belt (each one on the opposite branch with respect to the previous workpiece).

Parameters

SPEED	DISKS	Policy	POS_OUT_SENSORS	STATIONS_ELABORATION_TIME
1	12	3	[2, 24, 68, 90, 105]	[6, 7, 8, 9, 8, 7]

Simulations This is what we obtained by simulating the system for 1000 time instants:

Results All the properties are *verified*. As it's supposed to, the plant is working without any problem and all the workpieces flow all around the plant without any problem. The sensors are guaranteeing the required properties and the stations do their job flawlessly.

5.2 Scenario 2: short queues

With this scenario we want to test if the plant is working in a normal production situation (in which all the processing times of the various stations are balanced), but with short queues in front of the various stations.

Parameters

SPEED	DISKS	Policy	POS_OUT_SENSORS	STATIONS_ELABORATION_TIME
1	12	3	[12, 33, 76, 92, 106]	[2, 15, 5, 3, 2, 5]

Simulations This is what we obtained by simulating the system for 1000 time instants:

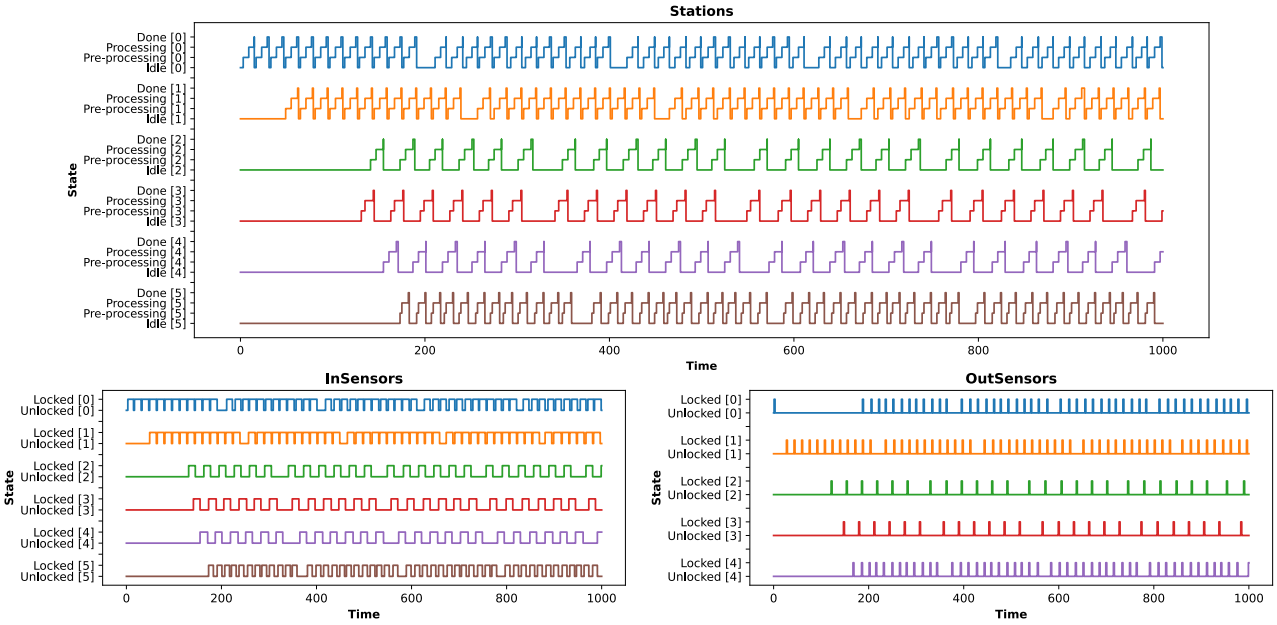


Figure 9: the results of the simulation over 1000 time instants.

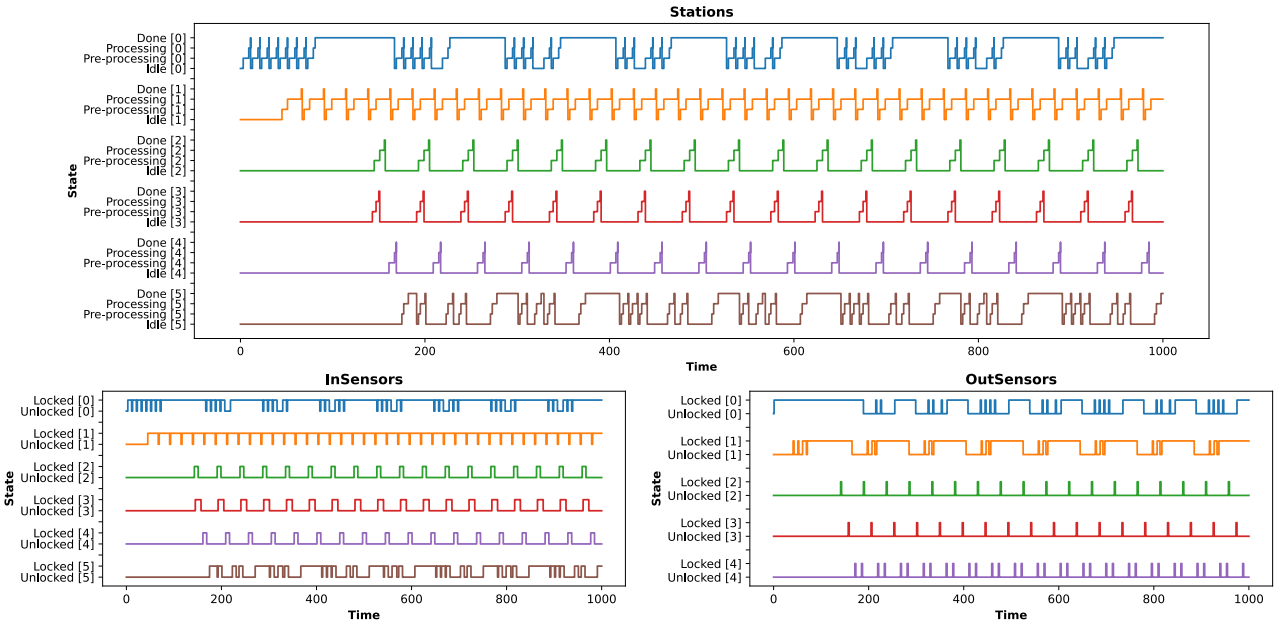


Figure 10: the results of the simulation over 1000 time instants.

Results All the properties are *verified*. The graphics show us that the system is working correctly without any particular delay on the stations.

Particular case If we try to play a bit with the processing times of the stations, we can create a situation where station 2 and 4 release a piece in such a way that they saturate the queue in front of station 5, exceeding its limit. An example could be a very short processing time on the first two stations in order to let the workpieces flow, the same overall time on the two branches, and very long one at the last station. In this way we break the third property, but it's completely expected, since the plant is not smart enough for preventing this situation to happen.

5.3 Scenario 3: one way

Here we wanted to test a different scheduling policy. In particular, we choose to route all the workpieces through the branch with the two processing stations. The only thing we changed with respect to the first scenario is the scheduling policy.

Parameters

SPEED	DISKS	Policy	POS_OUT_SENSORS	STATIONS_ELABORATION_TIME
1	12	2	[2, 24, 68, 90, 105]	[6, 7, 8, 9, 8, 7]

Simulations This is what we obtained by simulating the system for 1000 time instants:

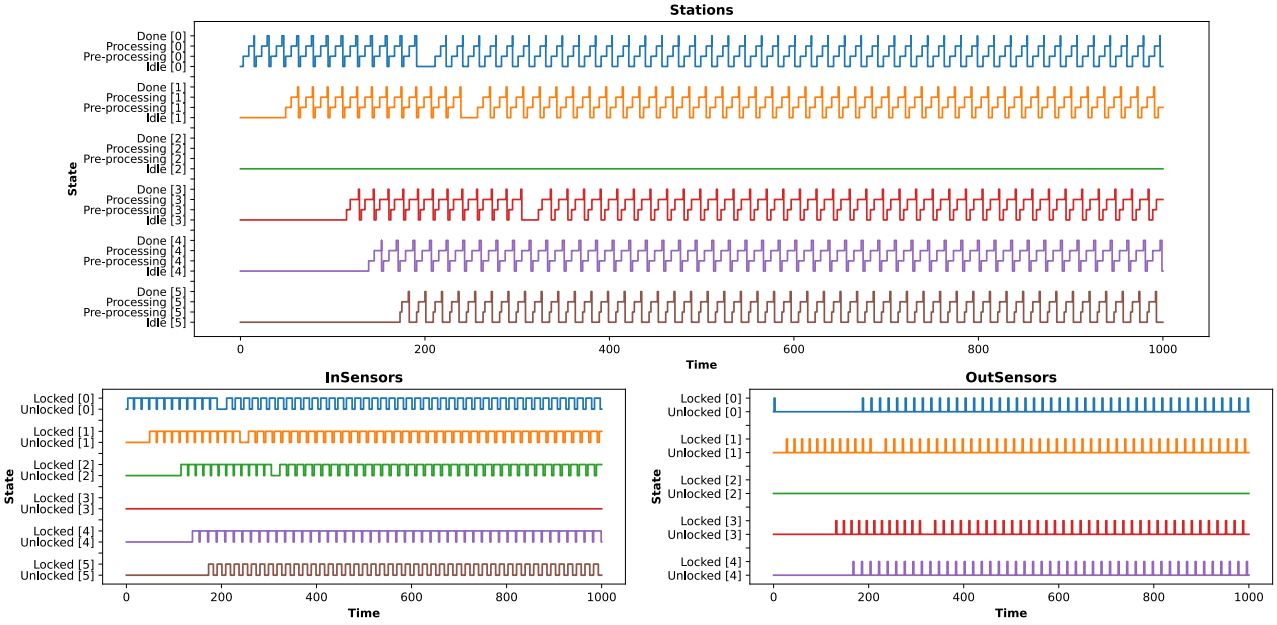


Figure 11: the results of the simulation over 1000 time instants.

Results All the properties are *verified*. If we take a closer look to the graphics, and we compare them with the ones of the first scenario, we can notice that the scheduling policy in which we divide the workpieces on the two different branches (so the one of the first scenario) is slightly more efficient than this one (station 0 here processes 56 pieces in 1000 time instants with respect to the 58 ones of the first scenario), which may be expected since, usually, balancing the workload is the best solution.

5.4 Scenario 4

5.5 Scenario 5 (stochastic)

5.6 Scenario 6 (stochastic)

6 Conclusions