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

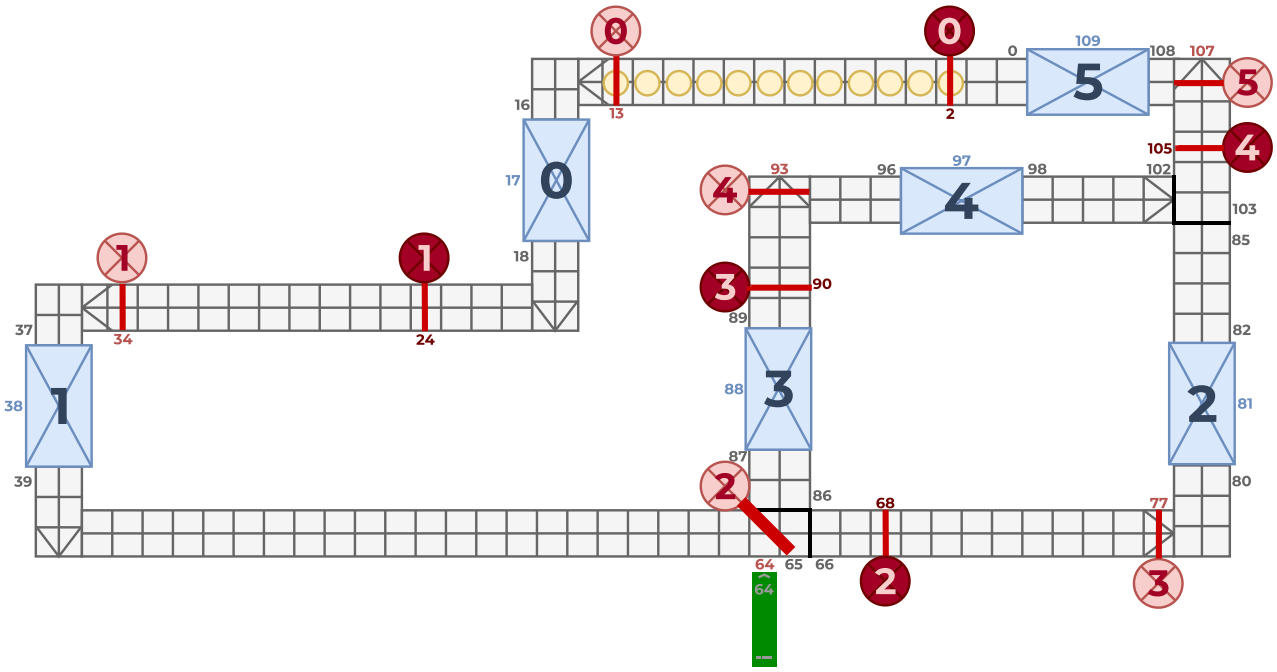


Figure 1: the production plant we modelled.

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 In order to make the entire description of our project as much clear as posssible, we enhanced the original scheme with the IDs we assigned to stations and laser sensors, and with the positions of the slots that compose the conveyor belt. Note that we decided to consider the stations as particular components placed above a normal slot; therefore we have two numbers defining each station: one refers to the position of the slot, while the other the ID of the station.

1.2 General Overview

The model of our system is made of 6 different components which interact between them 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 manager of the conveyor belt, in charge of moving around all the workpieces. In our model it's also the one in charge of preventing workpieces from entering a station, based on the information gathered by the laser sensors. Finally, it handles 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, once they get into them. We implemented a single template for them, which is instantiated as many times as needed in order to recreate the plant. As already explained, each station is assumed to be placed above a slot of the conveyor belt.

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` (represented as light red circles in Figure 1), while the ones guarding the queue and preventing the station right before it to release a workpiece are called `OutSensor` (represented as dark red circles). 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 where 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`)

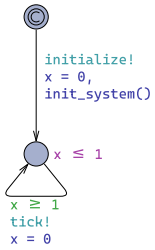


Figure 2: the Timed Automata of the initializer.

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

Initialization Initializes the system by placing the workpieces starting from the slot guarded by the first `InSensor` (index 0), and continuing 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 on which all the components of our plant, directly or indirectly synchronize (usually by means of the signals generated by the motor).

1.4 Motor (`Motor`)

As the name says, this is the meta component that manages all the signals used by the sensors, the conveyor belt and the stations, 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., check whether there is a workpiece right below them or not) and to perform some state-changing actions on all the stations (if needed).

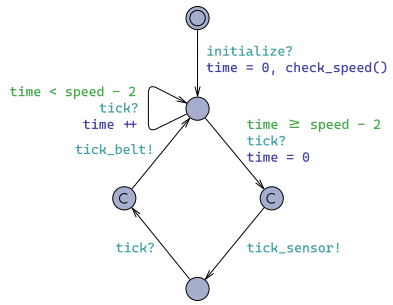


Figure 3: the Timed Automata of the motor.

1.5 Conveyor Belt (`ConveyorBelt`)

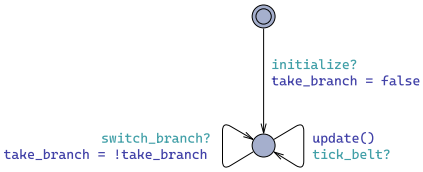


Figure 4: the Timed Automata of the conveyor belt.

It's the entity which handles 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 `InSensor` 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 of another one.

Conveyor belt branch and merge In order to be updated about which branch the next workpiece should take, we decided to keep the Conveyor Belt always ready to receive the signal `switch_branch?`. 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 (position 103), 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`)

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 representing a status in which the station can be.

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 entrance of the workpiece.

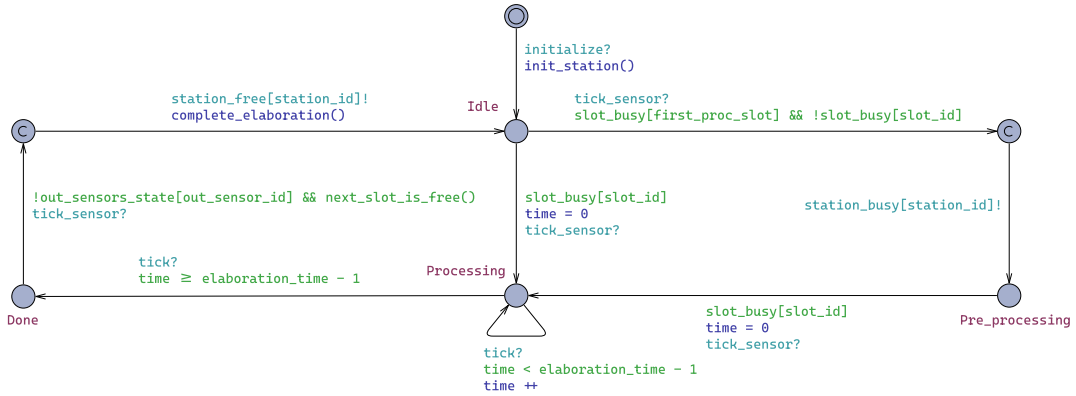


Figure 5: the Timed Automata of every station.

Processing Once the workpiece is entered, 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 queue of the following station, in order to know if the workpiece can be release or has to be held 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`)

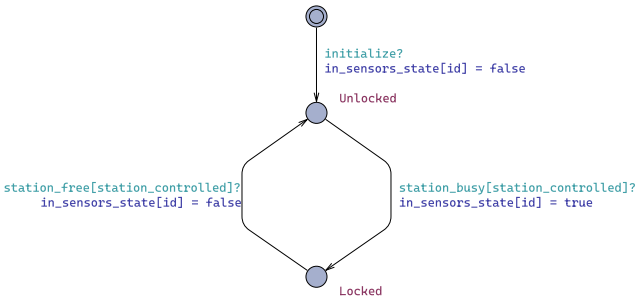


Figure 6: the Timed Automata of every input sensor.

These are the laser sensors that guard 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 guarded station, until it is free again.

1.8 Queue's guard (`OutSensor`)

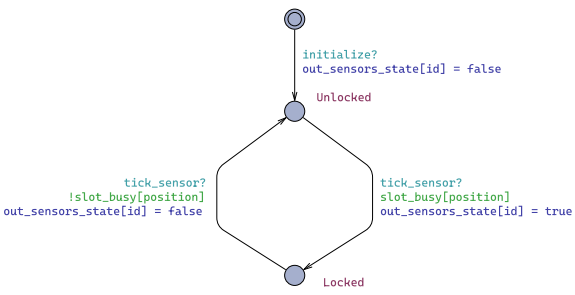
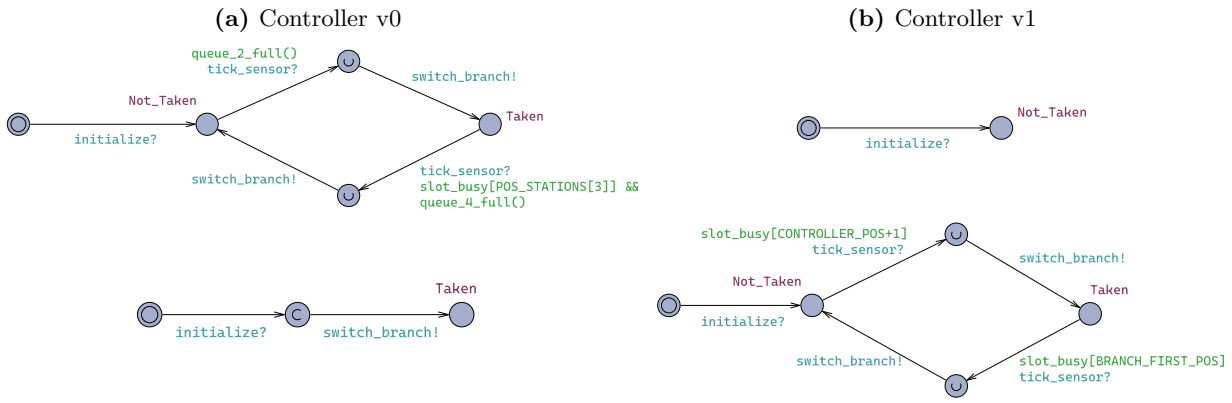


Figure 7: the Timed Automata of every output sensor.

Right before a station there is always an `InSensor` which guards its entrance, but it can be preceeded by another sensor, 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 prevents the preceeding station to release a workpiece until the queue is no longer 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 effective movement of the various workpieces is managed by the conveyor belt, these automata send a signal (`switch_branch`) to it each time the path followed by the workpieces has to be changed.

2 Design Decisions

2.1 Hypothesis

Slots' numbering We numbered the slots as shown in Figure 1. This helped us in the management of the conveyor belt's update function and implicitly allowed us to solve the problem of merging the two alterenative pathes at position 103. In the (possible) corner case in which both position 85 and position 102 are carrying a workpiece at the same time (i.e. they are simultaneously trying to place a workpiece on slot 103), we decided to apply the following policy: the first piece to flow is always the one coming from station 4. This is, of course, a simplification. A more complicated approach could have been taken, like randomly picking one of the two workpieces, but we decided to keep it as simple as we could.

Maximum number of circulating workpieces We decided to stick with the representation of Figure 1. So we have at most twelve workpieces circulating in the production plant, and they all are initially placed from position 13, proceeding backward. Please, notice that if the number of workpieces exceeds the position of the first `OutSensor` (which can be changed by project hypothesis), one of the properties isn't verified. However, this is normal since during the initialization of the plant we are already exceeding the maximum queue's length of the first one.

Time discretization This will be more clear in the next part, where we describe the main optimization steps we did, but, in brief, we discretized the time in order to have a better synchronization mechanism of the whole plant.

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 Proceeding in this way, we experienced huge verification times, due to the high number of entities and states that 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 the slots of the conveyor belt all together into a single logically 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 had a unique entity to manage the complexity derived from the movement of the various workpieces. This also works in synchronicity with the processing stations and the sensors, in order to block the pieces when needed, and with the flow controller, which communicates to the `ConveyorBelt` where to send the workpieces on the branch. By doing this we also improved significantly the verification time needed to check the properties, but we still had a huge memory consumption.

2.2.3 Automata simplification

Idea Since one of the things that was making the verification process heavy and slow, was the wide number of states traversed at each iteration, we decided to try to reduce the number of total states composing the entire system at the bare minimum.

Situation We removed some superfluous states from our model, and we splitted the flow controller into four different ones (one for each policy), in order to instantiate only the one that we were going to use during the simulation and verification of the plant. This allowed us to have slight improvements both in verification times and memory consumption.

2.2.4 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); since this fact was one of the biggest bottlenecks during the verification phase, we decided to reduce them to just a general one, and to keep all the various templates synchronized 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, while respecting the speed of the conveyor belt's movement; then 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 potentially take at any instance. 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

3.1 No two workpieces can be in the same position at the same time

Since we modelled the plant as a unique conveyor belt with the stations placed on some specific slots, this property embraces two different requirements of the production plant:

- *It never happens that a station holds more than 1 workpiece*;
- *It never happens that two pieces occupy the same conveyor belt slot.*

Note that the slots are modelled as an array of 110 Boolean values, in which `true` corresponds to the presence of a workpiece in that position.

In order to verify this property, we check that, after the plant has been initialized, the number of the workpieces remains constant and equal to its configured number, which can be represented in TCTL as follows:

$$\forall\Box \left(\underbrace{\texttt{initializer.run\_state}}_{\varphi_1} \implies \left(\underbrace{\sum_{i=0}^{109} \texttt{slot\_busy[i]} = \texttt{DISKS}}_{\varphi_2} \right) \right)$$

- φ_1: the system has been initialized;
- φ_2: the number of circulating workpieces in the production plant.

3.2 Queues never exceed their maximum length

We verified this by exploiting a custom function which, given the positions of a related pair (`InSensor`, `OutSensor`), we check that there is at least one free slot between the position of the `InSensor` and the position preceeding the one of the `OutSensor`. If the property is false it means that there is at most one extra piece in the queue. We verified this as follows:

$$\forall\Box \left(\forall i \in [0;4] \left(\underbrace{\texttt{outSensor(i).Locked}}_{\varphi_1} \implies \underbrace{\texttt{check\_queue(i, POS\_OUT\_SENSORS[i])}}_{\varphi_2} \right) \right)$$

- φ_1: the i-th `InSensor` is locked (the guarded station is not in `Idle`);
- φ_2: checks that the queue doesn't exceed its maximum length.

3.3 The plant never incurs in deadlock

In order to verify this, we checked both the automata part and the logic one. This means that we made sure to not incur in a deadlock in the transitions, and that all the pieces are continuing to circulate in the plant.

The property for the transitions is verified with $\forall\Box\,(\neg\texttt{deadlock})$, while the second one with:

$$\forall\Box \left(\neg \left(\underbrace{(\forall i \in [0;5]\,(\texttt{station(i).Processing} \vee \texttt{station(i).Done}))}_{\varphi_1} \wedge \underbrace{(\forall i \in [0;4]\,\texttt{outSensor(i).Locked})}_{\varphi_2} \wedge \underbrace{(\forall i \in [0;5]\,\texttt{inSensor(i).Locked})}_{\varphi_3} \right) \right)$$

- φ_1: all the stations are either blocked on `Processing` or `Done`;
- φ_2: all the `OutSensor` are preventing pieces from exiting the stations;
- φ_3: all the `InSensor` are blocking pieces from reaching the stations.

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).

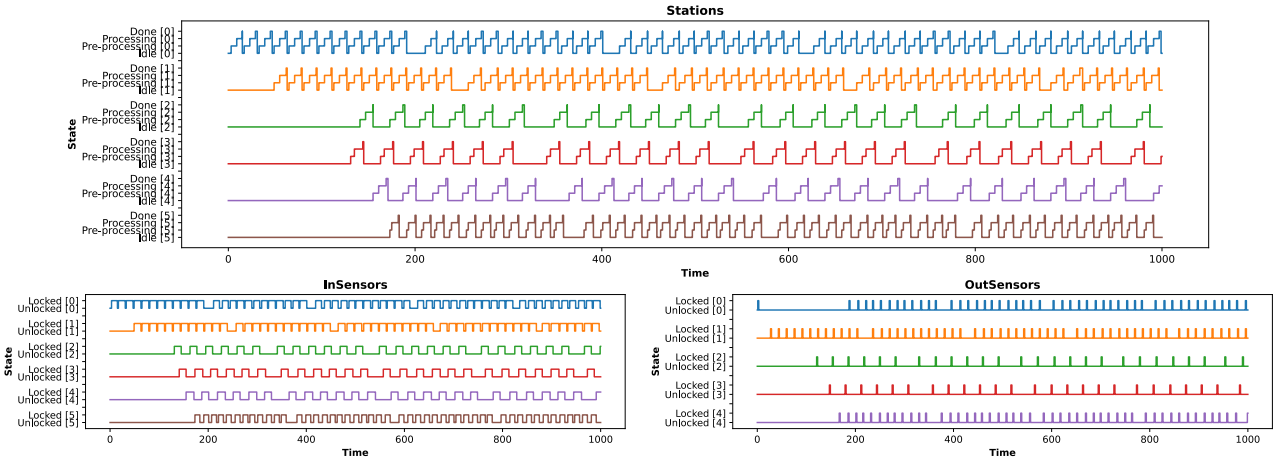


Figure 9: simulation over 1000 time instants of the first scenario.

Parameters These are the parameters used to obtain Figure 9.

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

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 These are the parameters used to obtain Figure 10.

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

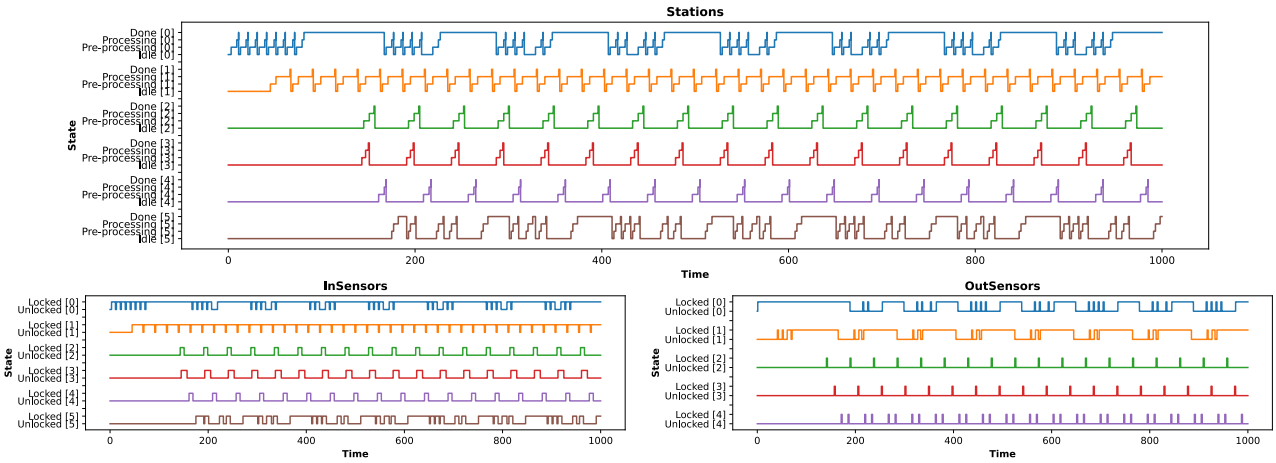


Figure 10: simulation over 1000 time instants of the second scenario.

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 to prevent this situation.

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 These are the parameters used to obtain Figure 11.

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

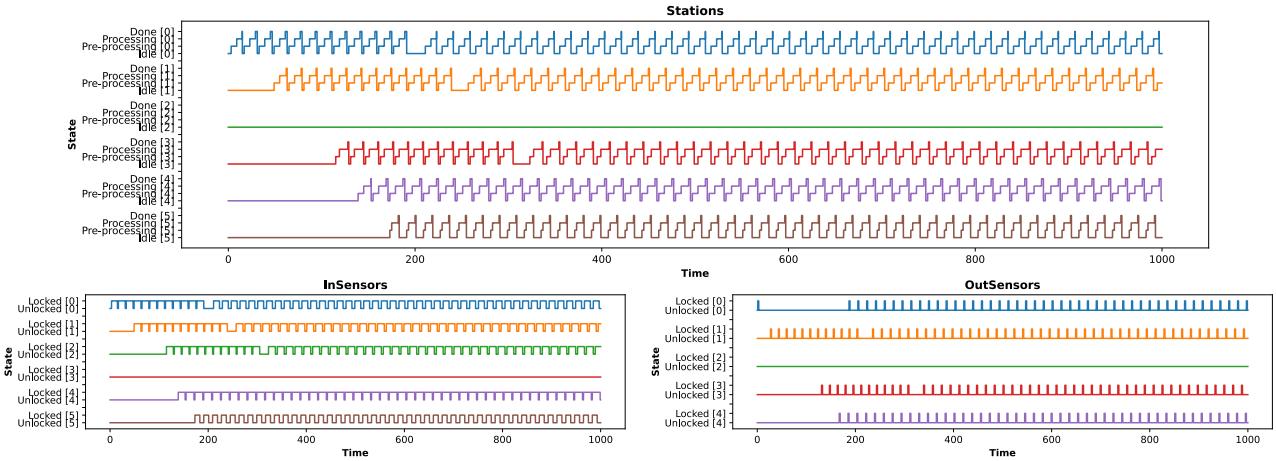


Figure 11: simulation over 1000 time instants of the third scenario.

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 (here, station 0 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: the stochastic case

This is an example of a stochastic plant in which the processing time of the stations varies as a Gaussian distribution around the average and the sensors can be faulty.

Parameters These are the parameters used to obtain Figure 12.

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

STD_DEV_STATIONS	IN_SENSORS_ERR	OUT_SENSORS_ERR
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0]	[1, 1, 1, 1, 1, 1]	[1, 1, 1, 1, 1, 1]

IN_SENSORS_RIGHT	OUT_SENSORS_RIGHT
[9999, 9999, 9999, 9999, 9999, 9999]	[9999, 9999, 9999, 9999, 9999, 9999]

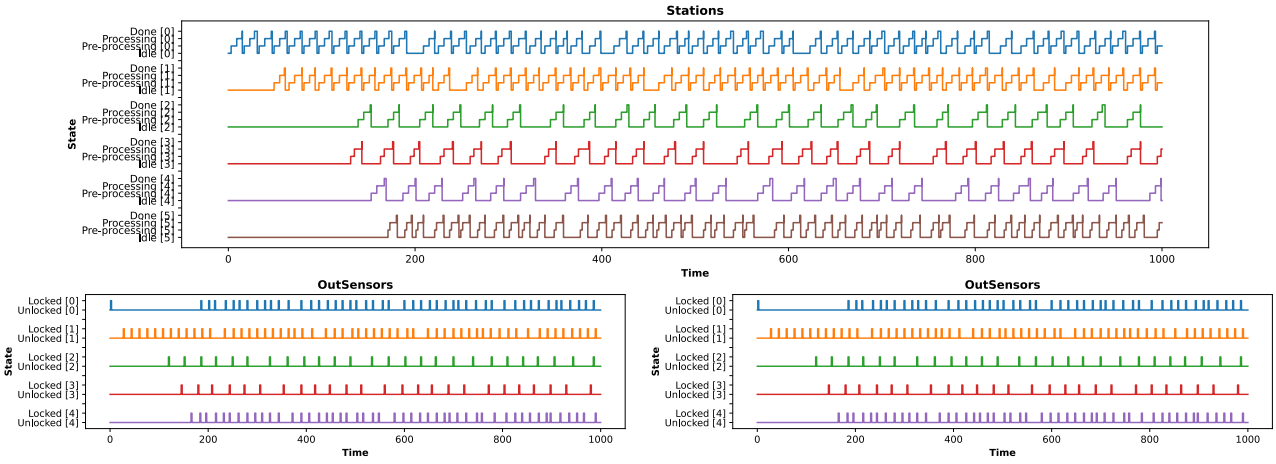


Figure 12: simulation over 1000 time instants of the fourth scenario in a stochastic environment.

Results All the properties are *verified* with a 95% confidence and with probability: $\geq 95\%$ (Property 1: No two workpieces can be in the same position at the same time), $84\% \pm 5\%$ (Property 2: Queues never exceed their maximum length) and $\geq 95\%$ (Property 3: The plant never incurs in deadlock). We can see that the plant is still working in a non-ideal scenario, processing correctly all the workpieces, without breaking physics laws. It may happen that some queues exceeds their maximum length because of faulty sensors.

6 Conclusions