POLITECNICO MILANO 1863

FORMAL DIGITAL TWIN OF A LEGO MINDSTORMS PRODUCTION PLANT

FORMAL METHODS FOR CONCURRENT AND REAL-TIME SYSTEMS
A.Y. 22/23

Prof. Pierluigi San Pietro,**Dr.Eng. Livia Lestingi**

Credits to Prof. Andrea Matta and Prof. Nicla Frigerio for providing footage and documentation of the Digital Twin Lab@DMec.

1 Introduction

A Digital Twin (DT) is the virtual replica of an actual Cyber-Physical System (CPS). DTs accurately replicate relevant behavioral aspects of a CPS. The purpose of a DT is to perform computations meant to improve the performance of the real CPS that would be resource-intensive if performed on the real system. Examples include the real-time synthesis of adaptation policies under uncertainty or scalability studies.

DTs are increasingly attracting attention within the manufacturing field. Production plants are, indeed, growingly complex due to the high demand for flexible and robust solutions. Furthermore, complex manufacturing systems are highly real-time critical since deadlocks, bottlenecks, and failures can incur efficiency and economic losses.

As engineers, you are in charge of developing the formal DT of a Lego Mindstorms[1] production plant. The work has two main outputs: an Automata-based model of the plant and a formal verification experimental campaign highlighting relevant features of the system.

The document is structured as follows:

Section 2 explains the mandatory entities to be modeled, how they are supposed to synchronize and their main characteristics.

Section 3 explains the mandatory properties to be verified.

Section 4 provides instructions on how to deliver your project.

2 Model

Production plants consist of different components connected through a conveyor belt that transports workpieces. The specific plant constituting the subject of the project is shown in Fig. 2[2].

The upcoming subsections introduce the mandatory features to be formally modeled.

2.1 Conveyor Belt

Workpieces flow through the plant on conveyor belts (a sample segment is shown in Fig. 1). Conveyor belt movement is one-directional, forcing pieces to flow through stations in a fixed sequence. For simplicity, we assume that all segments move at the same speed (which is, however, customizable). It is not necessary to explicitly model the length of the belt slot (e.g., 5cm or 1m): the translational speed can be expressed as [slots/(time unit)]. The number of slots for each segment and the arrangement of segments is fixed.

The only exception to the fixed processing sequence is the flow controller component (the green bar in Fig. 2). When the controller is "open" (i.e., in the position represented in Fig. 2), it lets pieces flow into station 5. If the controller closes while a piece is passing by, it pushes the piece into the alternative branch of the belt (i.e., through stations 3 and 4). The policy with which the controller opens or closes (thus, one branch is chosen rather than the

[1] Watch it in action here.

[2] Footage of the real plant deployed at the Digital Twin Lab is available here (password: **FM2223**, expires on Aug, 1st.)

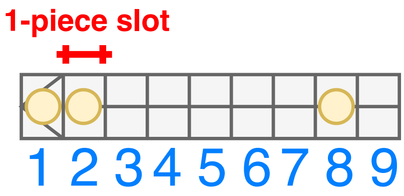

Figure 1: Diagram capturing a sample conveyor belt segment with numbered slots moving right to left. Each slot contains at most one workpiece. The piece in slot 2 cannot move if the piece in 1 does not leave the segment, whereas the piece in 8 is free to move forward.

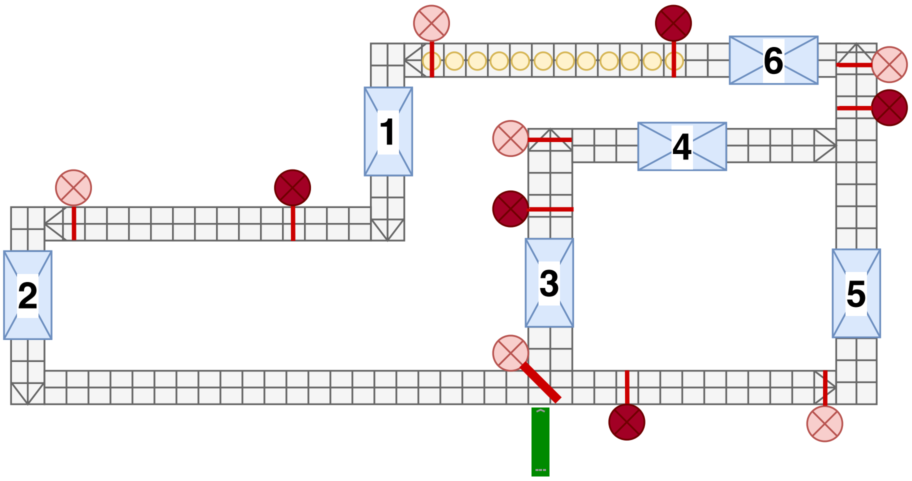

Figure 2: Diagram capturing the initial setup of the plant with pieces represented in yellow. Blue rectangles are processing stations. Red circles are sensors: lighter sensor guard the input to stations while darker sensors mark whether the queue has reached the maximum length. The green bar determines which branch of the belt each piece follows.

other) is customizable: it can be a static rule (i.e., the controller is *always open* or it switches for one out of three pieces) or depend on the state of the plant.

All pieces start queued up on the segment incoming station 1 and flow through the plant in loops. When a piece reaches station 1 again, a new processing loop begins.

2.2 Processing Station

Each station implements a processing phase of the workpiece. Neither specifying nor modeling the *specific* processing task of each station (e.g., finishing, packaging, or coating) is required. The position of a station is assumed to be *fixed* (i.e., it is not customizable).

Each station can process **at most one** piece at a time. Therefore, if the station is already busy, no new piece can enter.

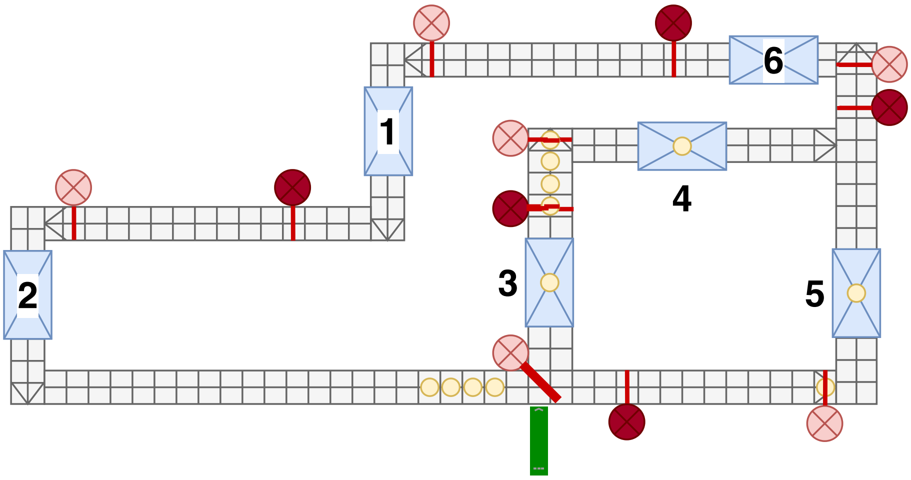

Figure 3: Diagram capturing the operational plant with circulating pieces.

A station holds a piece for a certain amount of time units representing the duration of the processing task. The duration may vary significantly between different stations since they realize different processing tasks.

Stochastic Version: Assume that the processing time for a single station is stochastic (e.g., the time spent within the station by different pieces is not deterministic), specifically, **normally** distributed [2] (mean and variance are assumed to be customizable).

It is not possible for a piece to flow through a station without being processed (hence, when a piece is positioned within a station, the envisaged delay must always elapse).

2.3 Laser-based Sensor

The plant is instrumented with laser-based sensors that detect the passage of a piece. There is always a sensor monitoring the **entrance** to a station. If such sensor detects the presence of a piece and the station is already busy (see, for example, station 5 in Fig. 3), the piece is mechanically blocked from entering the station. Explicitly modeling the bar blocking the piece is not necessary (we assume that the delay between the laser detecting the piece and the bar moving is negligible).

If multiple pieces are locked waiting for the station to become available again, they will form a **queue**. For each station, there may be a second laser-based sensor (placed farther away from the entrance) that marks the maximum length of a queue allowed by the station downstream. When this second sensor detects a piece (thus, the queue has reached the maximum length), the station upstream does not release any piece to prevent the queue from overflowing. For example, in Fig. 3, the two sensors upstream compared to station 4 both detect the presence of a piece; therefore, station 3 cannot release its piece until station 4 is free again and the queue can resume flowing.

Sensors guarding the entrance to a station are assumed to be *fixed* both in number and position. The position of sensors guarding the end of the queue (thus, the maximum length of the queue) is customizable. Furthermore, the presence of a maximum queue length is also optional (for example, station 3 does not have one besides the maximum capacity of the segments themselves).

Stochastic Version: Every time the conveyor belt moves, there is a certain probability that a sensor **malfunctions** (i.e., it detects a piece when it is not present or vice-versa). The probability weight of an error may vary between different sensors (irrespective of their role) and is assumed to be customizable.

3 Properties

The project will be carried out using the Uppaal[3] tool [1]. The verification experimental campaign must be carried out in two phases.

3.1 Phase 1: Exhaustive Model Checking

You are required to formally model the plant in Fig. 2 used as the running example without the stochastic features. The system must be modeled as a Network of **Timed Automata** (NTA) through the Uppaal GUI. Unless explicitly stated, you decide what to model as a TA feature and what via a variable for each described entity.

You are required to formally express in **TCTL** logic and verify the following properties:

1. it never happens that a station holds more than 1 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.

Verifying additional properties is welcome (indeed, encouraged) but these are to be considered mandatory.

Given the features presented in Section 2, the plant can have different configurations based on:

- the conveyor belt translational speed;
- the processing time of each station;
- the number of pieces circulating in the plant;
- the maximum length (if any) of the queue upstream of each station;
- the branch-switching policy.

Analyze and report on **at least three** plant configurations that highlight different behavioral aspects of the plant. Configurations must not be trivial (e.g., only one piece circulating or negligible processing time for all stations).

[3] Uppaal.org ↗

3.2 Phase 2: Statistical Model Checking

You are required to extend the model of the plant in Fig. 2 used for Phase 1 with the stochastic features [2].

You are required to formally express using Uppaal's `Pr` operator the **probability** of Phase 1 properties holding within specific time bounds. The time-bound used for verification must also be properly tuned to avoid trivial results (e.g., 0% deadlock probability within 1 time unit). Verifying additional properties (e.g., the estimated loop duration for a single piece to support the time-bound tuning task) is welcome also for Phase 2.

Given the stochastic features presented in Section 2, the plant can have different configurations based on:

- the processing time distribution of each station;
- sensors' failure probabilities.

Analyze and report on **at least three** non-trivial plant configurations (these can be stochastic versions of one or more configurations analyzed during Phase 1), highlighting relevant behavioral aspects of the plant.

4 Delivery Instructions

It is **mandatory** to deliver:

- a .pdf report (max 10 pages excluding front cover and bibliography, no constraint on the template) describing:
  - the model (emphasis on **critical** modeling choices you have made);
  - the system configurations you have chosen;
  - verification results;
- the .xml Uppaal model. Make sure:
  - it is the same as what you present in the report
  - it is test-ready (i.e., it includes the queries you ran for the experiments and system configurations are easily selectable).

The deadline for the delivery, also for students participating in the exchange program with UIC, is **July 20** (**June 22** if you want the grade registered with the first call).

To submit your work, send an email to livia.lestingi@polimi.it with the report, Uppaal files, and potential supplementary material to be evaluated.

Bibliography

[1] Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: Formal Methods for the Design of Real-Time Systems. Lecture Notes in Computer Science, vol. 3185, pp. 200–236. Springer (2004), https://doi.org/10.1007/978-3-540-30080-9_7

[2] David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: Uppaal SMC tutorial. Int. J. Softw. Tools Technol. Transf. **17**(4), 397–415 (2015), https://doi.org/10.1007/s10009-014-0361-y