POLITECNICO
MILANO 1863

Model Checking of a Lego Mindstorms Production Plant

Formal Methods for Concurrent and Real-Time Systems

Author
Christian Confalonieri

Professors
Pierluigi San Pietro
Livia Lestingi

2022/2023

Contents

1 Introduction

1.1 Lego Mindstorms Production Plant

The system under consideration is a Lego Mindstorms Production Plant, which is a simplified replica of an actual production plant constructed entirely using Lego pieces.
Its primary function is to transport pieces along **conveyor belts**.
Sensors detect these pieces, which are then temporarily held in designated spaces that simulate **processing stations**.
Pieces circulate within the plant indefinitely, continuously transitioning between different conveyor belts without a predetermined end.

1.2 Objective

The objective of the project is to develop a digital twin of this system.
This replication will be accomplished using the **Uppaal** model checking program (v5.0.0).
Through this program, the system needs to be modeled by defining templates composed of **timed automata** and their associated functions (written in a simplified version of C++).
During and after the modeling phase, it is essential to define and verify properties using the verifier.

2 Design and Modeling

2.1 Production Plant Layout

The formal digital twin of the Lego Mindstorms production plant has been modeled based on the following diagram:

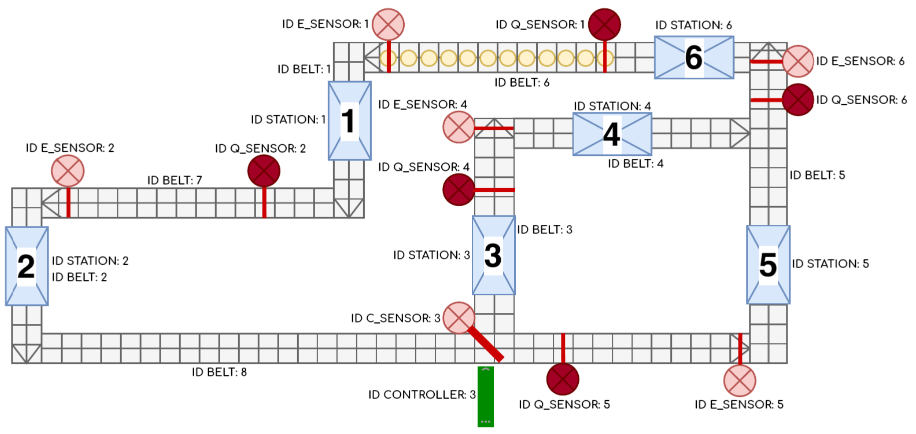


Figure 1: Initial configuration of the Lego Mindstorms Production Plant

In the initial setup, as shown in the diagram, the plant consists of six key elements:

1. **Pieces** to be transported and processed. They are represented by blue circles.
2. **Conveyor belts** on which the pieces are transported.
3. **Processing stations** where the workpieces are processed. They are represented by blue rectangles.
4. **Entrance sensors** used to monitor the incoming pieces at a station. They are represented with a lighter color.

5. **Queue sensors** employed to monitor the queue and detect when it reaches its maximum capacity. They are represented with a darker color.

6. **Flow controllers** that determine the branch of the belt that each workpiece follows. They are represented by a green bar.

2.2 Design Assumptions

Some details are not specified, so assumptions and choices have to be made in modeling. These are the choices that have been taken:

- Processing **stations occupy a "slot" of space**, which means that a piece cannot move within the station, but can only enter, stay inside the station and leave.

- **Sensors are vertical**, this means that the queue sensors will only be able to perceive whether there is a piece in their position and not whether the entire queue is filled.

- The processing **station is equipped with sensors** that allow it to determine the presence of a piece (or more precisely, the conveyor belt, through these unmodeled sensors, knows when to activate the station).

- The **movement of the flow controller is instantaneous**: pieces are sent to the designated belt in negligible time.

- **Pieces cannot be pushed/moved by other pieces**. For example in the case of the flow controller, a piece will not be able to push and make room for itself in the opposite belt if the last position is already occupied (the flow controller has the strength to move only one piece at a time, not two or more). In general, if a piece finds its way blocked by a queue, it cannot simply push the other pieces forward.

- The text specifies that if an entrance sensor detects a piece and the processing station is busy at that time, the piece and all successive pieces are mechanically blocked. In general, there are several slots between the station and the entrance, so **pieces can get stuck in between**. However, this does not cause any problems because, as mentioned above, the pieces cannot push or pass each other.

- The plant controller activates only the conveyor belts on which pieces are placed. In this way, **empty conveyor belts do not move pointlessly** (this in reality can be done, for example, through a weight sensor placed in the conveyor belts). In general, active conveyor belts move at the same speed, while inactive ones are stopped.

2.3 Global Declarations

2.3.1 System Parameters

Parameter	Description
`TAU`	Time-bound used for verification
`STATIONS`	Number of processing stations + 1
`BELTS`	Number of conveyor belts + 1
`beltDimension[8]`	Number of slots for each belt with id - 1
`entranceBelts[6]`	Id of the belt corresponding to entrance sensor with id - 1
`entrancePositions[6]`	Number of the slot in which the entrance sensor is placed
`queueBelts[6]`	Id of the belt corresponding to queue sensor with id - 1
`queuePositions[6]`	Number of the slot in which the queue sensor is placed
`stationPositions[6]`	Number of the slot in which the processing station is placed

2.3.2 Variables and Channels

Variable	Description
`belt<1-6>[beltDimension[]]`	Belt <1-6> (for each slot: 1 if occupied, 0 otherwise)
`entranceDetected[6]`	Array in which entrance sensor responses are saved
`queueDetected[6]`	Array in which queue sensor responses are saved
`stationBusy[6]`	Array in which processing stations states are saved

Channel	Description
`startSensor[STATIONS]`	Channel used by the conveyor belt to request a sensor scan
`startController`	Channel used by the conveyor belt to request activation of the flow controller
`startStation[STATIONS]`	Channel used by the conveyor belt to request activation of the processing station
`startBelt[BELTS]`	Channel used the by plant controller to request activation of conveyor belts
`waitBelt`	Channel used by the plant controller to wait for the conveyor belts
`stopBelt`	Channel used by the plant controller to request deactivation of conveyor belts
`updateFirst`	Channel used by the plant controller to request update of the last conveyor belt positions

2.4 Templates

During the modeling phase, a significant effort was made to keep automata consistent. This meant making sure that each automaton tried to copy the main functions of the real entities they represent. Furthermore, I tried to create **a template for each entity involved**, aiming to have a digital twin that closely resembles the real one.

2.4.1 Sensors

Entrance and Queue Sensors

```
EntranceSensor(const int id, const int reliability_rate)
QueueSensor(const int id, const int reliability_rate)
```

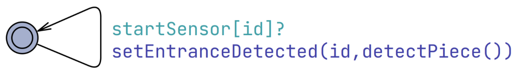


Figure 2: Entrance Sensor TA

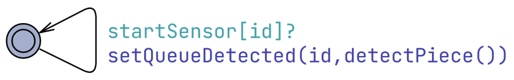


Figure 3: Queue Sensor TA

Timed automata for the entrance and queue sensors are quite simple and have a single function that detects whether a piece is on the corresponding slot or not.
In the stochastic version of the model, the perception of the piece could yield an opposite result based on a predefined probability weight (`reliability_rate`).
When the sensor detects a piece, it updates its position in the respective boolean array by setting it to true (`entranceDetected[]`/`queueDetected[]`).

Controller Sensor

`Controller_Sensor(const int id, const int reliability_rate)`

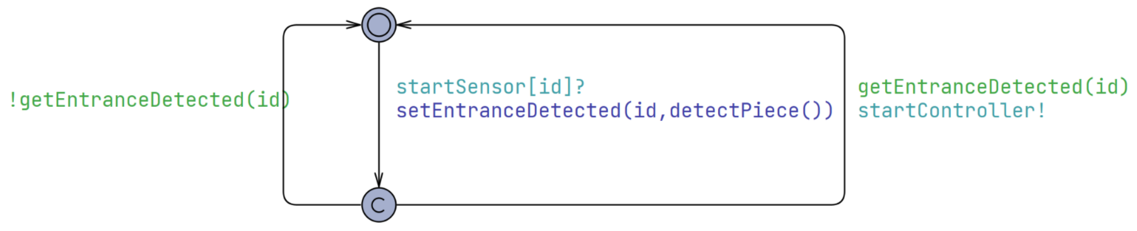


Figure 4: Controller Sensor TA

The controller sensor serves as a unique type of **entrance sensor** since it is responsible for activating the flow controller as well.
Regarding its ability to sense pieces in its slot, it functions exactly like an entrance sensor, including the stochastic version. However, it has an additional role: when it detects a piece, it also triggers the activation of the corresponding flow controller.

2.4.2 Flow Controller

`Flow_Controller(const int id)`

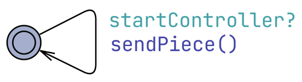


Figure 5: Flow Controller TA

The timed automaton of the flow controller may seem straightforward because I decided to include much of the logic inside the `sendPiece()` function.
As the name suggests, the `sendPiece()` function is responsible for sending the piece to the opposite conveyor belt, but this action only occurs if specific conditions are met (e.g. in one of the three scenarios analyzed, the controller behaves differently from the standard case).

2.4.3 Processing Station

`Processing_Station(const int id, const int average_time)`

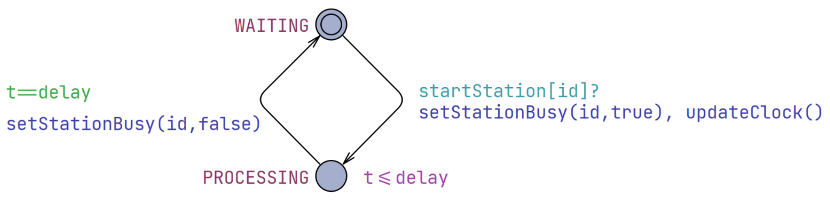


Figure 6: Processing Station TA

The processing station spends most of its time in the WAITING position. When it is triggered by its conveyor belt, it updates its state in the `stationBusy[]` array and starts processing the piece. The processing time varies for each station and, in the stochastic version, it follows a normal distribution. Once the processing time (`delay`) is reached, the processing station updates its state again in the `stationBusy[]` array and goes back to the WAITING position.

2.4.4 Conveyor Belts

To avoid having many duplicated and unused locations and to have more order, it was decided to divide the conveyor belts in 4 subcategories:

1. Conveyor Belt #1 (id: 1,2,4): containing only the processing station
2. Conveyor Belt #2 (id: 7): containing only general sensors (entrance and queue)
3. Conveyor Belt #3 (id: 3,5,6): containing both the processing station and general sensors (entrance and queue)
4. Conveyor Belt #4 (id: 8): containing only the controller sensor and general sensors (entrance and queue)

All the conveyor belts operate similarly, but they activate different entities based on their type.
In general, each timed automaton remains in a waiting state until the plant controller activates it. Once it receives the activation command `startBelt[]`, it initiates the starting sequence for its sensors and station, if present, in a sequential manner.
After that, it transitions to a new waiting state and communicates this status to the plant controller using the `waitBelt` command.
When the conveyor belt receives the signal from the plant controller that it can finish its operation, it performs a step by moving the various pieces one position forward. However, the last slots are updated differently; incoming pieces are received from other conveyor belts when the `updateFirst` command is invoked. This is repeated at each cycle of the timed automaton.

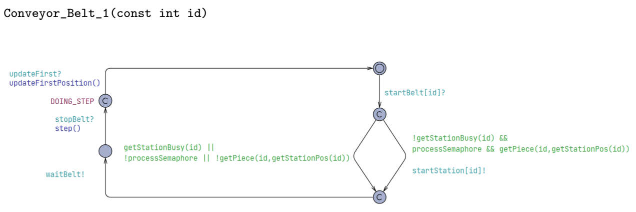


Figure 7: Conveyor Belt #1 TA

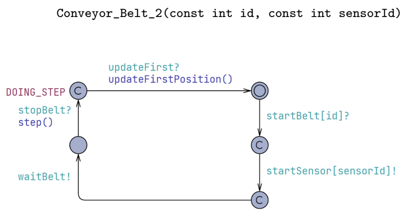


Figure 8: Conveyor Belt #2 TA

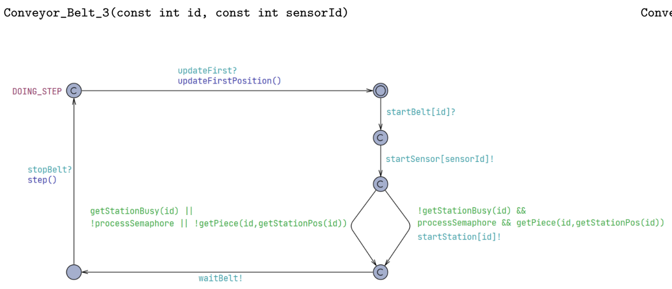


Figure 9: Conveyor Belt #3 TA

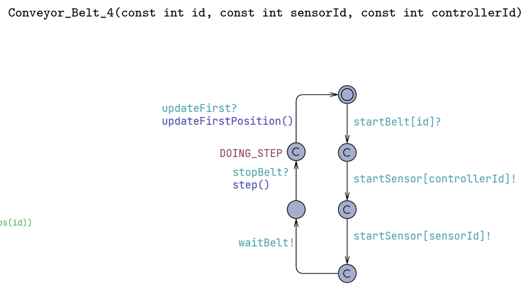


Figure 10: Conveyor Belt #4 TA

2.4.5 Plant Controller

`Plant_Controller(const int step_time)`

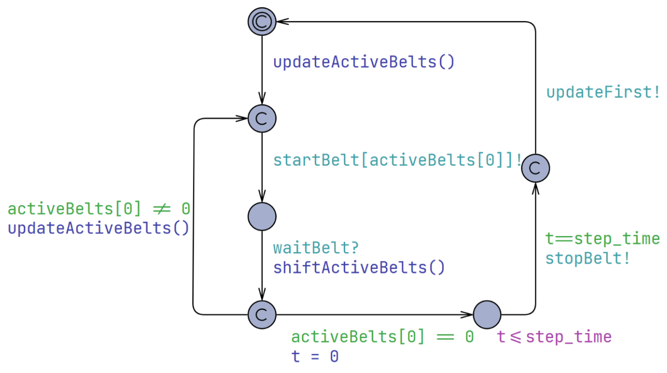


Figure 11: Plant Controller TA

This timed automaton is responsible for the overall management of the entire production plant. Its main tasks include managing, activating, and deactivating the various conveyor belts. As stated in the assumptions section, the belts are activated or deactivated depending on the presence of pieces. Additionally, this automaton is responsible for determining the speed of the belts, which specifically involves setting the time required for each step.

3 Analysis and Results

3.1 Properties

The verifier was used to check and confirm all the essential/mandatory properties as well as other interesting and useful ones added to gain a better understanding of the behavior of the designed production plant. This process helped ensure that the plant operates as intended and meets the desired specifications. All properties have been verified through the verifier (even those implicitly verified). For greater cleanliness of the text, the syntax of these properties has been omitted in this report but can be found within the .xml file.

3.1.1 Exhaustive Model Checking

Mandatory Properties

1. *It never happens that a station holds more than 1 piece.*
   This is implicitly verified by the use of boolean arrays for belt representation. The design and implementation of the conveyor belts using boolean arrays ensure that each station can only hold a single piece at any given time.

2. *It never happens that two pieces occupy the same belt slot.*
   In the same way also this property is implicitly verified.

3. *No queue ever exceeds the maximum allowed length.*

4. *The plant never incurs in deadlock.*

Additional Properties

5. *The production plant never has duplicate or missing pieces.*
   This in some ways replaces the first two mandatory properties, in fact through its verification it is certain that there are no functions or displacements that duplicate or make pieces disappear. This property was verified on a production plant with 12 pieces.

6. *All empty and waiting stations will eventually process a piece.*
   This ensures that no processing station will remain idle indefinitely, and all stations will have the opportunity to process a piece. This property is essential for the smooth and efficient operation of the production plant, as it guarantees that the processing capacity of each station will be utilized, and there won't be any bottlenecks in the production process.

7. *All stations that are processing a piece will eventually release it.*
   This ensures that no processing station will hold a piece indefinitely, and all pieces will eventually continue their movement through the production plant.
   This property is crucial for maintaining a steady flow of production.

8. *If a belt has 0 pieces, it will eventually have at least 1 piece.*
   This ensures that no conveyor belt will remain empty indefinitely, and over time, at least one piece will be present on it.
   This property is vital for maintaining a continuous and active production process. By verifying this property, it can be assured that all conveyor belts will continuously move pieces through the production plant, avoiding any scenarios where conveyor belts are left without any pieces, which could lead to inefficiencies or production delays.

9. *If a belt has at least 1 piece, it will eventually have 0 pieces.*
   This ensures that no conveyor belt will continuously hold a piece indefinitely, and over time, all pieces on the conveyor belt will be processed and moved away.
   This property is crucial for maintaining a balanced and well-functioning production process. It prevents any accumulation of pieces on the belts, which could lead to blockages or disruptions in the production flow.

3.1.2 Statistical Model Checking

In statistical model checking, all mandatory properties were verified, and additional properties were reduced/compressed into:

5. *The production plant never has duplicate or missing pieces.*

6. *Eventually all belts will receive at least 1 piece.*

7. *Eventually all stations begin to work.*

Parameters

The SMC default parameters have been used in all scenarios, with a time bound of 10000. On the right is shown a table summarizing the parameters used; further changes are communicated along with the scenarios.

Parameter	Value
$\pm\delta$	0.01
α, β, ϵ	0.05
u0	0.9
u1	1.1
Trace resolution	4000
Discretization step	0.01
TAU (τ)	10000

In the next scenarios I will use the respective numbers (listed earlier) to refer to the various properties (exhaustive and statistical).

3.1.3 Settings

To switch from the exhaustive model to the statistical model, perform the following steps:

- `Entrance_Sensor`, `Queue_Sensor`, `Controller_Sensor`: uncomment/comment the corresponding "`x`" variable.
- `Processing_Station`: uncomment/comment the corresponding "`delay`" variable and "`updateClock()`" function.

3.2 First Scenario - Default

The first scenario presented is the **standard/default situation** where the production plant works with general settings and behavior. It's designed to handle normal production needs without any special customizations or unusual situations.
The settings for this scenario are:

- **Branch switching policy:** the flow controller is activated once every 3 pieces detected (to have this behavior, the variable "`acceptance`" must be set to 2 in the declariations of the flow controller).
- **Number of queue slots:** initial configuration (to enable this option uncomment the line "`queuePositions[6] = {12, 11, -1, 3, 9, 3};`" in the global declarations, and comment out the other one with the same name).

3.2.1 Exhaustive Model Checking

Mandatory Property

1	●
2	●
3	●
4	●

Additional Property

5	●
6	●
7	●
8	●
9	●

The model was implemented and modified in parallel with the use of the simulator and verifier, so it is correct and expected that all properties of the default case are satisfied.

3.2.2 Statistical Model Checking

Mandatory Property

1	$\geq$ 0.950056	●
2	$\geq$ 0.950056	●
3	$\geq$ 0.950056	●
4	$\geq$ 0.950056	●

Additional Property

5	$\geq$ 0.950056	●
6	$\geq$ 0.950056	●
7	$\geq$ 0.950056	●

Also in the stocasting version, the model was implemented and modified in parallel with the use of the simulator and verifier, so it is correct and expected that all properties of the default case are satisfied.

3.3 Second Scenario

This second scenario was designed to try to **clog the conveyor belt number 3** and thus study the behavior of the production plant.
Because of the way the model was designed, it is expected that some of the pieces will not be able to access that belt and will therefore proceed to the conveyor belt number 5.

The settings for this scenario are:

- **Branch switching policy:** the flow controller is activated for each piece detected.
  (to have this behavior, the variable "`acceptance`" must be set to 0 in the declariations of the flow controller).
- **Number of queue slots:** initial configuration (to enable this option uncomment the line "`queuePositions[6] = {12, 11, -1, 3, 9, 3};`" in the global declarations, and comment out the other one with the same name).

3.3.1 Exhaustive Model Checking

Mandatory Property

1	🟢
2	🟢
3	🟢
4	🟢

Additional Property

5	🟢
6	🔴
7	🟢
8	🟢
9	🟢

From the properties analyzed, it becomes evident that strange behavior occurs, as at some point, the verification of the second additional property is no longer valid. Specifically, upon closer examination of the verifier, it reveals that the station associated with conveyor belt number 5 will cease processing pieces at a certain stage of the production plant cycle. This situation arises due to the pieces being extensively spread out across the conveyor belts, resulting in the elimination of any clogging issues in conveyor belt 3. All this can be seen even better in the stochastic version.

3.3.2 Statistical Model Checking

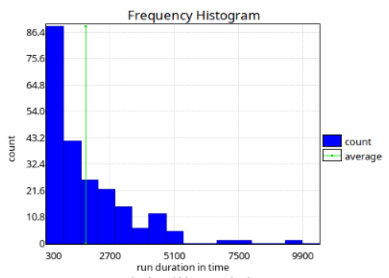
Mandatory Property

1	≥ 0.950056	🟢
2	≥ 0.950056	🟢
3	0.921013 ± 0.0417609	🟢
4	≥ 0.950056	🟢

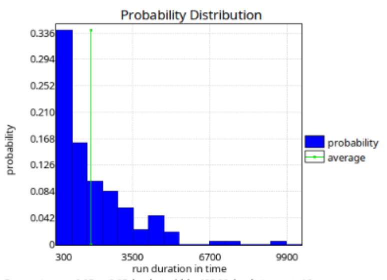
Additional Property

5		≥ 0.950056	🟢
6		≥ 0.950056	🟢
7	Eventually all stations except 5 start	≥ 0.950056	🟢
	Eventually station 5 starts	0.838919 ± 0.0459328	

Already from these results we can see that the third property (no queue ever exceeds the maximum allowed length) is not verified in all cases. This happens precisely because of the assumptions made earlier (between a sensor and a busy station there could be pieces), in fact because of the congestion created in conveyor belt 3 it will at some point lead to exceeding the queues (probably those of belt number 3 and 5).
The property we are most interested in, however, is number 7; in fact, we can see that specifically station 5 might stop processing pieces in some cases. Investigating further through the graphs provided by the verifier we see a behavior that confirms this: at some point in the production phase station number 5 will stop receiving pieces.



```
Parameters: α=0.05, ε=0.05, bucket width=635.33, bucket count=16
Runs: 261 in total, 220 (84.291%) displayed, 41 (15.709%) remaining
Span of displayed sample: [350, 9880]
Mean estimate of displayed sample: 1759 ± 195.6 (95% CI)
```



```
Parameters: α=0.05, ε=0.05, bucket width=635.33, bucket count=16
Runs: 261 in total, 220 (84.291%) displayed, 41 (15.709%) remaining
Span of displayed sample: [350, 9880]
Mean estimate of displayed sample: 1759 ± 195.6 (95% CI)
```

3.4 Third Scenario

This scenario analyzes the case where the **queues are 1 slot long**, this means that the entrance and queue sensor are in the same position. Verifying this case is useful in order to delineate a fixed rule for setting the queues.
The settings for this scenario are:

- **Branch switching policy:** the flow controller is activated once every 3 pieces detected (to have this behavior, the variable "`acceptance`" must be set to 2 in the declariations of the flow controller).
- **Number of queue slots:** 1 (to enable this option uncomment the line "`queuePositions[6] = {1, 1, -1, 0, 0, 1};`" in the global declarations, and comment out the other one with the same name).

3.4.1 Exhaustive Model Checking

Mandatory Property

1	🟢
2	🟢
3	🔴
4	🟢

Additional Property

5	🟢
6	🔴
7	🔴
8	🔴
9	🔴

Through the verifier we can see that at least 4 of the additional properties are not verified (in addition to the mandatory one related to queues). This highlights the importance of having queues with a minimum length of two in the production plant's model, as any deviation from this requirement could lead to unexpected and erroneous behavior.

3.4.2 Statistical Model Checking

Mandatory Property

1	≥ 0.950056	🟢
2	≥ 0.950056	🟢
3	≤ 0.0499441	🟢
4	≥ 0.950056	🟢

Additional Property

5	≥ 0.950056	🟢
6	≥ 0.950056	🟢
7	≥ 0.950056	🟢

It is interesting to see that no major problems seem to appear in the additional properties of the stochastic version; it turns out that only the third mandatory property is not verified (as expected). Since both models are not verified and erroneous behavior is found, however, it does not make sense to allow queues of length 1 (unless substantial changes are made during the modeling phase).

For all scenarios, in rare cases, an ***"invalid invariant"*** *error may come up during statistical model checking, in that case repeat the properties check several times until successful.*

4 Conclusions

To summarize, the model of the production plant was designed to be as complete as possible. I tried to represent every entity involved and implement all essential functions, with the goal of achieving a model that closely resembled the real system. In addition, a significant effort was made to optimize the model so that the verifier could check it in a reasonable time, even with a large number of pieces (12, according to the initial proposed configuration).