Università
degli Studi
di Padova

University of Padua

Department of Mathematics “Tullio Levi-Civita”

Formal Methods for Cyber-Physical Systems

ASSIGNMENT 2

Luca Monetti (Mat. 2199440)

Introduction and objectives

The primary objective of this second assignment is the design and implementation of a Supervisory Control System for a virtual production plant utilizing the Eclipse ESCET toolset with the CIF modeling language.

The system simulates a factory environment consisting of three machines, three depots and two rovers with logistical and maintenance operations. The project aims to synthesize a controller that guarantees the correct behavior of the system.

The roles and configurations of each entity will be described in the *Plant architecture* section.

Structure of the report

The present report documents the design and implementation process of the Supervisory Control System.

The report is structured as follows:

- **Section 2: Plant architecture**: Contains a description of all the physical entities.
- **Section 3: Graphical interface**: Illustration of the visual interface of the plant.
- **Section 4: Control Requirements**: Formalization of the six control policies with an explanation of the actual implementation.
- **Section 5: ToolDef**: Explanation of the automated script used to synthesize the supervisor and the controlled-system.

Plant architecture

Each component of the plant has a specific role in the production process:

- **Machines**: There are three machines (M_1, M_2, M_3) designed for processing raw materials into finished products. They need to take input from the Blue Rover and deposit the finished workpiece into the corresponding Depot. If a failure occurs during operation, the machine breaks and halts until repaired.
- **Depots**: The plant includes three depots (D_1, D_2, D_3) for storing the workitems located in the same cell of the corresponding machine.
- **Rovers**: Two rovers (BR, OR) that can freely move within the plant. Each movement consumes battery power. The Blue Robot is a logistical unit, capable of carrying two workitems of each type and transferring them between the machines. The Orange Robot is a maintenance unit, responsible for repairing the machines when they break down.

Machine modeling

The machines (M_1, M_2, M_3) are the main processing units of the system. Each unit receives a workpiece from the *Blue Rover*, executes a transformation process and either outputs the result to the corresponding *Depot* or breaks down and halts until it is repaired by the *Orange Rover*.

Each machine is modeled as a finite automaton with the following states:

- **Idle**: The machine is ready to process workitems. This is the initial and marked state.
- **Working**: The machine has received a workitem and is processing it.
- **Broken**: The machine is out of order and needs the presence of the *Orange Rover* to perform a repair.

The possible events are categorized by their controllability:

- **Controllable Events**:
  - `Mx_start`
  - `Mx_repaired`
- **Uncontrollable Events**:
  - `Mx_finish_success`
  - `Mx_broken`

Depot modeling

The depots (D_1, D_2, D_3) are the storage of the system, acting as intermediate buffers between the machines and the final target.

Each depot is modeled as an extended state machine that utilizes a discrete integer variable to manage its internal state:
- `disc int[0..2] quantity`: Tracks the number of workitems stored within the depot.

The automaton consists of a single initial and marked state and it is synchronized with the corresponding machine and the Blue Rover. The `quantity` variable is updated as follows:
- **Increment**: The `quantity` increases when is less than two and the machine completes a processing.
- **Decrement**: The `quantity` decreases when is greather than zero and the Blue Rover picks up a workpiece.

The possible events are categorized by their controllability:

- **Controllable Events**:
  - `BR_take_x`
- **Uncontrollable Events**:
  - `Mx_finish_success`

Where $x \in [1, 3]$.

Rovers modeling

The rovers (BR, OR) share the navigation and energy functionalities but also have a different specif role:
- The *Blue Rover* manages workitems logistics
- The *Orange Rover* handles the system maintenance.

Both rovers are modeled as extended state machines whose internal state is defined by three discrete variables:
- `disc int[0..8] energy`: Represents the rover's battery level, which decreases with each movement and can be recharged at the Charging Stations.
- `disc int[1..6] x`: Represents the rover's current x-coordinate within the plant.
- `disc int[1..4] y`: Represents the rover's current y-coordinate within the plant.

The automaton consist of a single initial and marked state and each variable is modified according to specific rules:
- **Directional movement**: Each rover can move between cells using directional events. Each event check if the reached cell is inside the boundaries, there is enough energy left and the move is *valid* (explained on *Requirement Formalization* section).
- **Charging**: Each rover can charge when is positioned at one of the two Charging station, this action resets the rover's energy to its maximum capacity.

The possible events are categorized by their controllability:

- **Controllable Events**:
  - `[col]R_move_up`
  - `[col]R_move_down`
  - `[col]R_move_left`
  - `[col]R_move_right`
  - `[col]R_charge`

Where `[col]` can be either `O` or `B`.

Blue Rover (BR)
The Blue Rover (BR) is responsible for transporting workitems between depots and machines. It can carry up to 2 workitems of each type.

This model is characterized by 4 discrete variable that tracks the number of workitems of each type currently carried by the rover:
- `disc int[0..2] wi0`: Number of workitems of type 0 carried by the rover
- `disc int[0..2] wi1`: Number of workitems of type 1 carried by the rover.
- `disc int[0..2] wi2`: Number of workitems of type
- `disc int[0..2] wi3`: Number of workitems of type 3 carried by the rover.

The automaton is synchronized with each machine model. Each workitems variable is updated as follows:
- **Increment**: The variable `wix` where $x \in [0, 3]$ increase when the rover take a workpiece from a depots or the source. The rover must be in the same cell.
- **Decrement**: The variable `wix` where $x \in [0, 3]$ decrease when a machine start its processing or the rover deliver the final product to the target. The rover must be in the same cell.

It has eight additional controllable events:

- **Controllable Events**:
  - `BR_take_x`
  - `BR_deliver_3`
  - `My_start`

Where $x \in [0, 3]$ and $y \in [1, 3]$.

Orange Rover (OR)
The Orange Rover (OR) is responsible for repairing broken machines. Must reach the broken machine to repair it, allowing the production to resume.

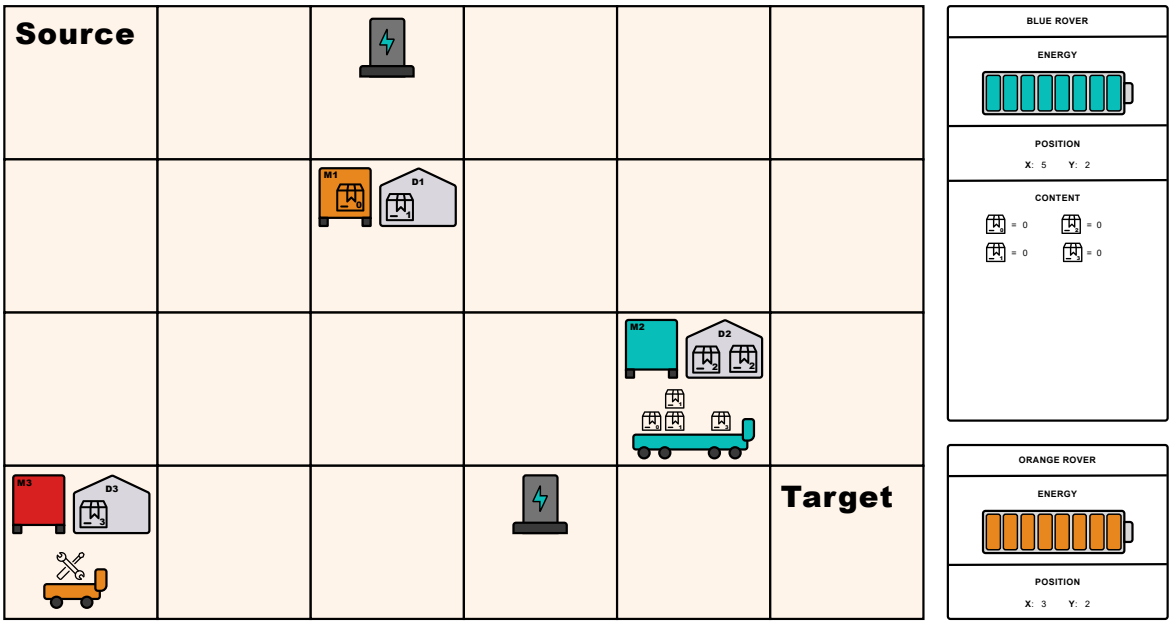
It has three additional events:

- **Controllable event**:
  - `Mx_repair`

Where $x \in [1, 3]$.

Graphic Interface

To facilitate the supervision of the plant the project includes a dynamic interface that provides a visual representation of the rovers states.



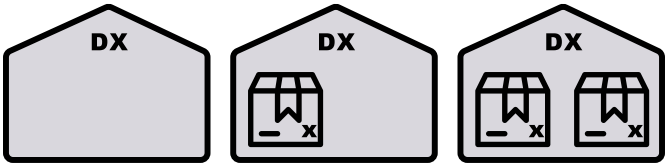
Each element in the graphic interface is dynamically updated based on the current state of the plant components.

Machines

Each machine is represented by a rectangle that changes color based on its state:

- **Idle**: Light Blue
- **Working**: Orange
- **Broken**: Red

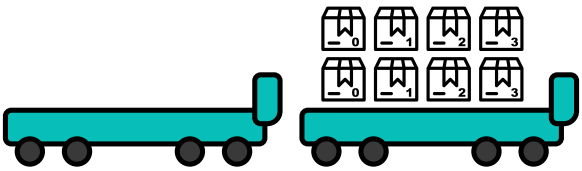
Depots



Each depot represent the number of workitems stored using small icons inside the rectangle:

- **Empty Depot**: No icons
- **Partially Full Depot**: One icons
- **Full Depot**: Two icons

Blue Rover



The Blue Rover is represented by a blue rover icon that moves within the plant. The actual positioning of the rover within the SVG interface is achieved thought a dynamic coordinate transformation using the following translation formula:

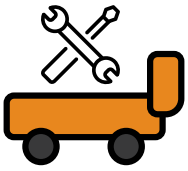
```
svgout id "B-Rover" attr "transform" value
  fmt("translate(%d, %d)", 52 + 207 * (BlueRover.x - 1), 135 + 207 * (4 - BlueRover.y))
```

The parameters used are some calibration constants to align the model with the visual grid.

To provide a clear indication of the rover's actual payload the interface utilizes conditional visibility for each workitem icons:

```
svgout id "Item-00" attr "display" value if BlueRover.wi0 >= 1 : "inline" else "none" end;
svgout id "Item-01" attr "display" value if BlueRover.wi0 = 2 : "inline" else "none" end;
```

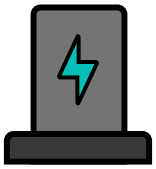
Orange Rover



The Orange Rover is represented by an orange rover icon that moves within the plant according to its coordinates following the same logic of the Blue Rover.

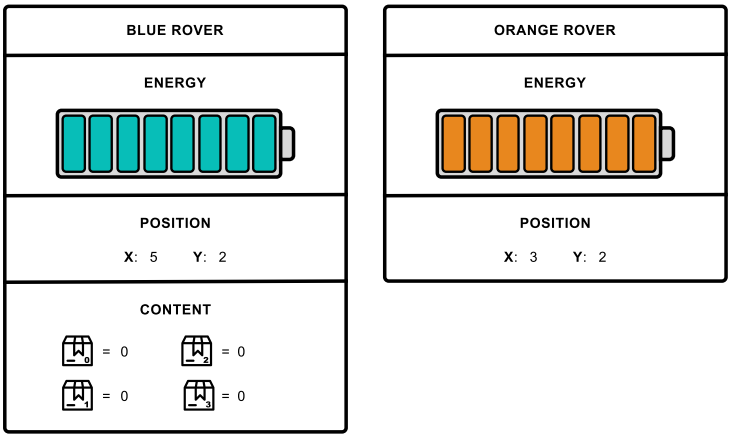
```
svgout id "O-Rover" attr "transform" value
  fmt("translate(%d, %d)", 92 + 207 * (OrangeRover.x - 1), 144 + 207 * (4 - OrangeRover.y));
```

Charging Stations



The Charging Stations are represented by a gray charging station icon located at fixed coordinates within the grid. They have no real state and are used only as reference points for the rovers to recharge their batteries.

Rover status dashboard



On the right side of the interface is located a status dashboard that gets dynamically updated to show the rovers' internal state.

Taking the *Blue Rover* as an example:

- **Energy Level**: A simple battery that indicates the current energy level of the rover, with a maximum value of 8.

```
// Energy Update
svgout id "BR-L1" attr "display" value if BlueRover.energy >= 1 : "inline" else "none" end;
svgout id "BR-L2" attr "display" value if BlueRover.energy >= 2 : "inline" else "none" end;
...
svgout id "BR-L8" attr "display" value if BlueRover.energy = 8 : "inline" else "none" end;
```

- **Position**: The current coordinates of the rover within the plant grid.

```
// Position Update
svgout id "BR-X-Pos" text value BlueRover.x;
svgout id "BR-Y-Pos" text value BlueRover.y;
```

- **Carried Workitems**: A visual representation of the workitems currently carried by the Blue Rover, with a maximum of 2 icons for each type of workitem.

```
// Workitems Update
svgout id "BR-0-Label" text value BlueRover.wi0;
...
svgout id "BR-3-Label" text value BlueRover.wi3;
```

Requirements

In this section all the control policies that the supervisory controller must enforce to ensure the correct behavior of the plant are explained and fomalized.

Requirement 1

"No rover runs out of battery on tiles that are not charging stations"

Battery preservation is a critical requirement for the rovers operating within the plant. If a rover depletes its battery while on a tile that is not a charging station, it would become immobilized. For this reason this control policy is enforced at plant level.

Before moving the rover needs to know if it will be able to reach a Chargin Station from that cell with its remaining energy. This can be done by checking if the Manhattan distance from at least one of the two charging stations after the move is less than or equal to the remaining energy.

```
// functions.cif
func int[0..10] distance(int[0..7] x1; int[0..5] y1; int[1..6] x2; int[1..4] y2):
    return abs(x1 - x2) + abs(y1 - y2);
end

func bool validMove(int[0..7] x; int[0..5] y; int[-1..7] energy):
    return
      distance(x, y, Charger1_x, Charger1_y) <= energy or
      distance(x, y, Charger2_x, Charger2_y) <= energy;
end

// Example of usage
edge BR_move_up
  when y <= 3 and energy >= 1 and validMove(x, y + 1, energy - 1)
  do y := y + 1, energy := energy- 1;
```

Requirement 2

"The sums of the units of energy of the two batteries is always >= 1"

At any moment the combined energy levels of the two rovers must be at least 1 unit. In order to prevent a situation where both rovers are completely out of energy, the supervisor checks the resulting sum of their energy levels before allowing any move.

```
requirement R2:
    location: initial; marked;
        edge BR_move_up, BR_move_down, BR_move_left, BR_move_right,
             OR_move_up, OR_move_down, OR_move_left, OR_move_right
             when BlueRover.energy + OrangeRover.energy - 1 >= 1;
end
```

The guard `BlueRover.energy + OrangeRover.energy - 1 >= 1` ensures that **after** any movement event, the total energy of both rovers remains at least 1 unit.

In combination with the previous requirement we know that the rover with energy level zero will be exactly on one of the two Charging Stations.

Requirement 3

"The maximum number of workpieces that the blue rover can carry is 4"

At any moment the Blue Rover can carry at most 4 workitems in total, regardless of their type. This control policy is enforced using a requirement that checks the sum of all workitems currently carried by the rover before allowing any `take` event.

```
requirement R3:
    location: initial; marked;
        edge BR_take_0, BR_take_1, BR_take_2, BR_take_3
            when BlueRover.wi0 + BlueRover.wi1 + BlueRover.wi2 + BlueRover.wi3 < 4;
end
```

Requirement 4

"Each rover can charge provided its battery is not already full"

To prevent unnecessary charging actions, each rover is allowed to charge only when its battery is not already full. This control policy is enforced using a requirement invariant that checks the current energy level of the rover before allowing it to perform the charging action.

```
requirement invariant R4A: BR_charge needs BlueRover.energy < 8;
requirement invariant R4B: OR_charge needs OrangeRover.energy < 8;
```

Requirement 5

"Rovers do not collide"

Two rovers must never occupy the same position within the plant at the same time. This control policy is enforced using a requirement that checks if the two rovers are on adjacent tiles, if so the movement that would cause a collision is prevented.

```
requirement R5:
    location: initial; marked;
        edge BR_move_up, OR_move_down
          when (BlueRover.y + 1 != OrangeRover.y and BlueRover.x = OrangeRover.x) or
               OrangeRover.x != BlueRover.x;

        edge BR_move_right, OR_move_left
          when (BlueRover.x + 1 != OrangeRover.x and BlueRover.y = OrangeRover.y) or
               OrangeRover.y != BlueRover.y;

        edge BR_move_down, OR_move_up
          when (BlueRover.y - 1 != OrangeRover.y and BlueRover.x = OrangeRover.x) or
               OrangeRover.x != BlueRover.x;

        edge BR_move_left, OR_move_right
          when (BlueRover.x - 1 != OrangeRover.x and BlueRover.y = OrangeRover.y) or
               OrangeRover.y != BlueRover.y;
end
```

Requirement 6

"If D3 is full, then D1 can store at most one workpiece"

If Depot D3 contains 2 workpieces, then Depot D1 is only allowed to store at most one workitem. This control policy is enforced using a requirement invariant that checks the quantity of workitems in both depots before allowing Machine M1 to start another process. This is necessary since the `M1_start` event is controllable while the `M1_finish_success` event is uncontrollable.

```
requirement invariant R6: M1_start needs D1.quantity < 1 or D3.quantity < 2;
```

Tooldef

The final stage of the project involved the automation of the supervisor generation process using a `tooldef` file that specifies the necessary configurations.

The `tooldef` file is structured into three main section:

1. **Environment preparation**: If the `generated` directory does not exist, it is created to store the generated CIF files.
2. **Supervisor Synthesis**: The `cifdatasynth` command is used to generate the supervisor based on the plant requirements specified in the `plant/requirements.cif` file. The output is saved in the `generated/supervisor.cif` file.
3. **Plant Merging**: The `cifmerge` command is used to combine the original plant model with the synthesized supervisor, resulting in a controlled system saved in the `generated/controlled-system.cif` file.

```
from "lib:cif" import *;

if not exists("generated"):
    mkdir("generated");
end

cifdatasynth(
    "plant/requirements.cif",
    "-o generated/supervisor.cif",
    "-n supervisor"
);

cifmerge(
    "plant/plant.cif",
    "generated/supervisor.cif",
     "-o generated/controlled-system.cif"
);
```