Intelligent Systems Assignment 2

Christoph Royer [12004184]

June 27, 2025

1 Description Logic

1.1 Semantics

1.1.1

$(\exists \text{over.White})^I$
$= \{x \in \Delta^I | \exists y \in \Delta^I : (x, y) \in \text{over}^I \wedge y \in \text{White}^I\}$
$= \{x \in \Delta^I | \exists y \in \Delta^I : (x, y) \in \{(\Box, \clubsuit), (\star, \bigcirc), (\spadesuit, \diamondsuit), (\clubsuit, \heartsuit), (\bigcirc, \blacksquare), (\diamondsuit, \triangle)\} \wedge y \in \{\Box, \bigcirc, \diamondsuit, \heartsuit, \triangle\}\}$
$= \{\star, \spadesuit, \clubsuit, \diamondsuit\}$

$(\exists \text{over.Black})^I$
$= \{x \in \Delta^I | \exists y \in \Delta^I : (x, y) \in \text{over}^I \wedge y \in \text{Black}^I\}$
$= \{x \in \Delta^I | \exists y \in \Delta^I : (x, y) \in \{(\Box, \clubsuit), (\star, \bigcirc), (\spadesuit, \diamondsuit), (\clubsuit, \heartsuit), (\bigcirc, \blacksquare), (\diamondsuit, \triangle)\} \wedge y \in \{\star, \spadesuit, \clubsuit, \blacksquare\}\}$
$= \{\Box, \bigcirc\}$

$$\begin{aligned}
(\text{White} \sqcap \text{Suit} \sqcap \exists \text{over.White} &\sqsubseteq \neg\exists \text{over.Black})^I \\
(\text{White} \sqcap \text{Suit} \sqcap \exists \text{over.White})^I &\subseteq (\neg\exists \text{over.Black})^I \\
\text{White}^I \cap \text{Suit}^I \cap (\exists \text{over.White})^I &\subseteq (\neg\exists \text{over.Black})^I \\
\{\Box, \bigcirc, \diamondsuit, \heartsuit, \triangle\} \cap \{\spadesuit, \clubsuit, \diamondsuit, \heartsuit\} \cap \{\star, \spadesuit, \clubsuit, \diamondsuit\} &\subseteq \Delta^I \setminus \{\Box, \bigcirc\} \\
\{\diamondsuit\} &\subseteq \{\Box, \star, \spadesuit, \clubsuit, \bigcirc, \diamondsuit, \heartsuit, \blacksquare, \triangle\} \setminus \{\Box, \bigcirc\} \\
\{\diamondsuit\} &\subseteq \{\star, \spadesuit, \clubsuit, \diamondsuit, \heartsuit, \blacksquare, \triangle\} \\
&\text{Satisfies.}
\end{aligned}$$

1.1.2

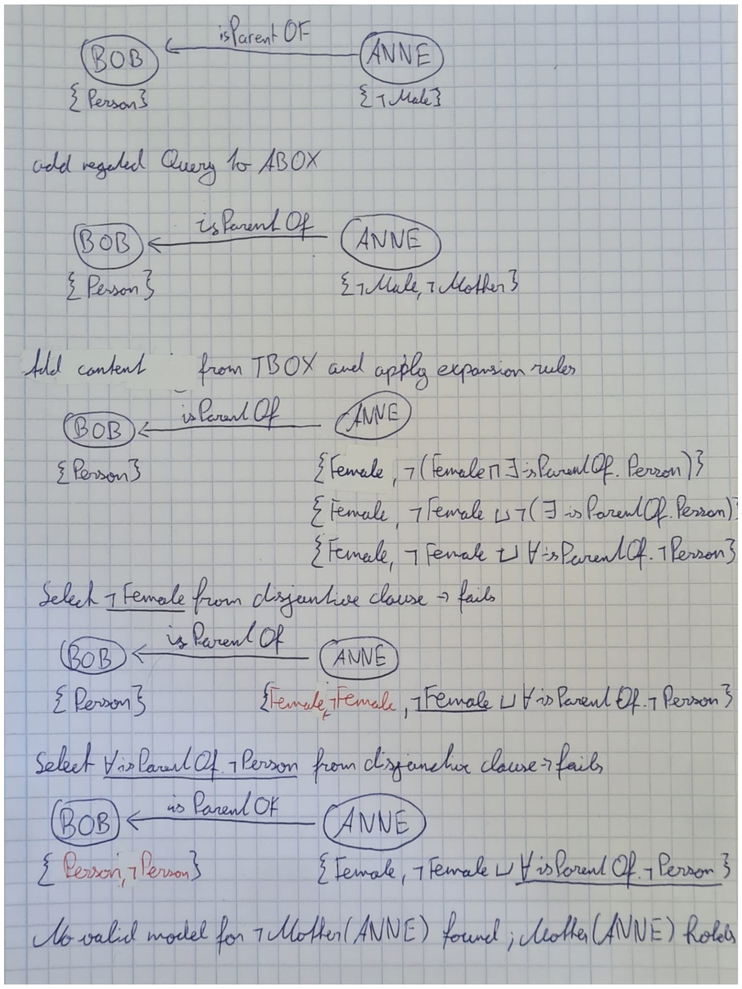
$$\begin{aligned}
&(\forall \text{left}.\top)^I \\
&= \{x \in \Delta^I | \forall y \in \Delta^I : (x,y) \in \text{left}^I \to y \in \top^I\} \\
&= \{x \in \Delta^I | \forall y \in \Delta^I : (x,y) \in \text{left}^I \to y \in \Delta^I\} \\
&= \{x \in \Delta^I | \top\} \\
&= \Delta^I
\end{aligned}$$

$$\begin{aligned}
(\exists \text{over.Black} &\equiv (\neg\text{Suit} \sqcap \forall\text{left}.\top) \sqcap \neg\text{Black})^I \\
(\exists \text{over.Black})^I &= ((\neg\text{Suit} \sqcap \forall\text{left}.\top) \sqcap \neg\text{Black})^I \\
\{\square, \bigcirc\} &= (\neg\text{Suit} \sqcap \forall\text{left}.\top)^I \cap \Delta^I \backslash \text{Black}^I \\
\{\square, \bigcirc\} &= (\neg\text{Suit})^I \cap (\forall\text{left}.\top)^I \cap \{\square, \bigcirc, \diamondsuit, \heartsuit, \triangle\} \\
\{\square, \bigcirc\} &= \Delta^I \backslash \{\spadesuit, \clubsuit, \diamondsuit, \heartsuit\} \cap (\forall\text{left}.\top)^I \cap \{\square, \bigcirc, \diamondsuit, \heartsuit, \triangle\} \\
\{\square, \bigcirc\} &= \{\square, \star, \bigcirc, \blacksquare, \triangle\} \cap \Delta^I \cap \{\square, \bigcirc, \diamondsuit, \heartsuit, \triangle\} \\
\{\square, \bigcirc\} &= \{\square, \bigcirc, \triangle\}
\end{aligned}$$

Does not satisfy.

1.2 Reasoning

$$\begin{aligned}
\text{TBOX :} & \text{Male} \equiv \neg\text{Female} \\
& \text{Mother} \equiv \text{Female} \sqcap \exists \text{isParentOf.Person} \\
& \text{Father} \equiv \text{Male} \sqcap \exists \text{isParentOf.Person} \\
& \text{Parent} \equiv \text{Mother} \sqcup \text{Father} \\
\text{ABOX :} & \text{Person(BOB)} \\
& \text{isParentOf(ANNE, BOB)} \\
& \neg\text{Male(ANNE)}
\end{aligned}$$



$$\begin{tikzcd} \boxed{\text{BOB}} & \boxed{\text{ANNE}} \arrow[l, "\text{isParent OF}"'] \end{tikzcd}$$

{Person} {¬Male}

add negated Query to ABOX

$$\begin{tikzcd} \boxed{\text{BOB}} & \boxed{\text{ANNE}} \arrow[l, "\text{isParent Of}"'] \end{tikzcd}$$

{Person} {¬Male, ¬Mother}

Add content from TBOX and apply expansion rules

$$\begin{tikzcd} \boxed{\text{BOB}} & \boxed{\text{ANNE}} \arrow[l, "\text{isParent Of}"'] \end{tikzcd}$$

{Person}

$\{\text{Female}, \neg(\text{Female} \sqcap \exists \text{isParentOf. Person})\}$

$\{\text{Female}, \neg\text{Female} \sqcup \neg(\exists \text{ isParentOf. Person})\}$

$\{\text{Female}, \neg\text{Female} \sqcup \forall \text{isParentOf.} \neg\text{Person}\}$

Select $\underline{\neg\text{Female}}$ from disjuntive clause → fails

$$\begin{tikzcd} \boxed{\text{BOB}} & \boxed{\text{ANNE}} \arrow[l, "\text{is Parent Of}"'] \end{tikzcd}$$

{Person} $\{\text{Female}, \neg\text{Female}, \underline{\neg\text{Female}} \sqcup \forall \text{isParentOf.} \neg\text{Person}\}$

Select $\underline{\forall \text{isParentOf.} \neg\text{Person}}$ from disjunctive clause → fails

$$\begin{tikzcd} \boxed{\text{BOB}} & \boxed{\text{ANNE}} \arrow[l, "\text{is Parent Of}"'] \end{tikzcd}$$

$\{\text{Person}, \neg\text{Person}\}$ $\{\text{Female}, \neg\text{Female} \sqcup \underline{\forall \text{isParentOf.} \neg\text{Person}}\}$

No valid model for ¬Mother(ANNE) found; Mother(ANNE) holds

1.3 Modeling and Reasoning

The finished ontology (all three inventories combined) can be found in *task1c.owl*.

The queries for recipes that are possible with a given inventory are also saved within the file, and are defined as follows:

```
Recipe and (requires only (isInInventory value Inventory1))
Recipe and (requires only (isInInventory value Inventory2))
Recipe and (requires only (isInInventory value Inventory3))
Recipe and (requires only (isInInventory value Inventory4))
```

Each inventory is modeled as an Individual, and an Ingredient, tool, or Appliance is linked as being within an inventory by the *isInInventory* property.

The tool *Whisk* and *Mixer* are marked as equivalent and can be used interchangeably.

2 Constraint Programming

2.1 Railway

The corresponding constraint program can be found in *task2a.mzn*

The tiles are constrained to be connected to neighbors (vertically and horizontally), to not be connected to the edges, to only place the allowed tiles, and so that they all form one continuous track.

The output of the program is as follows:

```
board =
[| RB, BL, RB, LR, LR, BL
 | TB, TR, CC, BL, RB, LT
 | TB, RB, LT, TR, CC, BL
 | TB, TR, BL, RB, LT, TB
 | TR, LR, LT, TR, LR, LT
 |];
tiles_continuous =
[| true, true, true, true, true, true
 | true, true, true, true, true, true
 | true, true, true, true, true, true
 | true, true, true, true, true, true
 | true, true, true, true, true, true
 |];
```

Figure 1 shows a visualization of the generated track.

2.2 Agents and Tasks

The corresponding constraint program can be found in *task2b.mzn*

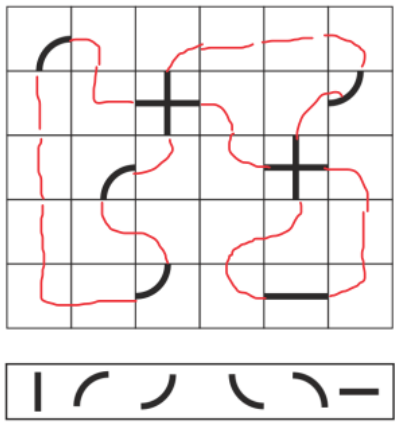


Figure 1: The solution for the railway problem

Each task is assigned an agent, with the constraint that the sum of tasks for an agent cannot exceed the agent's capacity.

The output of the program is as follows:

```
agent_resources_used = [121, 101, 56, 37];
task_assignment = [4, 1, 2, 3, 2, 1];
```

3 Planning

3.1 PDDL

The PDDL domain and problem can be found at *task3a-domain.pddl* and *task3a-problem.pddl* respectively.

The domain file models the way the robot can move along connected locations, and that it can pick up packages as long as its capacity is not exceeded, and put packages down that it currently carries.

The problem file describes the concrete layout of locations, the capacity of the robot and the weight of the packages. It also gives the start and target locations for the robot and all the packages.

The output of the solver is as follows:

```
0.000: (move robot a d)  [0.001]
0.001: (move robot d f)  [0.001]
0.002: (pickup robot package1 f)  [0.001]
0.003: (move robot f h)  [0.001]
0.004: (putdown robot package1 h)  [0.001]
0.005: (move robot h g)  [0.001]
0.006: (pickup robot package3 g)  [0.001]
```

```
0.007: (move robot g h)  [0.001]
0.008: (move robot h f)  [0.001]
0.009: (move robot f d)  [0.001]
0.010: (move robot d b)  [0.001]
0.011: (putdown robot package3 b)  [0.001]
0.012: (move robot b d)  [0.001]
0.013: (move robot d e)  [0.001]
0.014: (pickup robot package2 e)  [0.001]
0.015: (move robot e d)  [0.001]
0.016: (move robot d f)  [0.001]
0.017: (move robot f h)  [0.001]
0.018: (putdown robot package2 h)  [0.001]
0.019: (move robot h f)  [0.001]
0.020: (move robot f d)  [0.001]
0.021: (move robot d a)  [0.001]
```

Since no weight is given to movement, the heuristic solver determines that a lower load is better, and thus prefers to move up and down twice instead of carrying packages 1 and 2 at the same time.

It was tested by altering the problem that the robot can also carry two packages, the solver simply chooses not to in this instance.

3.2 MiniZinc

The corresponding constraint program can be found in *task3b.mzn*

The states for the robot position, package position, robot carrying a package, load and action are all encoded in arrays to signify time passing.

The *Gecode* solver does not handle this problem well, I had quick execution times with the *OR Tools CP-SAT 9.12.4544*.

The output is as follows:

```
stepsNeeded = 16;
robotLocation = [A, D, E, E, D, F, F, H, H, G, G, H, F, D, B, A, A, A,
        A, A, A, A, A, A, A];
packageLocation =
[| F, E, G
 | F, E, G
 | F, E, G
 | F, E, G
 | F, D, G
 | F, D, G
 | F, D, G
 | F, D, G
 | H, H, G
 | H, H, G
 | H, H, G
 | H, H, G
 | H, H, F
 | H, H, D
 | H, H, B
 | H, H, B
 | H, H, A
 | H, H, A
 | H, H, A
 | H, H, A
 | H, H, A
 | H, H, A
 | H, H, A
 | H, H, A
 | H, H, A
 |];
robotCarries =
[| false, false, false
 | false, false, false
 | false, false, false
 | false,  true, false
 | false,  true, false
 | false,  true, false
 |  true,  true, false
 |  true,  true, false
 | false, false, false
 | false, false, false
 | false, false,  true
 | false, false,  true
 | false, false,  true
 | false, false,  true
 | false, false,  true
 | false, false,  true
 | false, false, false
 | false, false, false
 | false, false, false
 | false, false, false
 | false, false, false
 | false, false, false
 | false, false, false
 | false, false, false
 | false, false, false
 |];
actions = [Move, Move, Pickup, Move, Move, Pickup, Move, Putdown, Move, Pickup,
        Move, Move, Move, Move, Move, Putdown, Done, Done, Done, Done, Done,
        Done, Done, Done, Done];
load = [0, 0, 0, 10, 10, 10, 30, 30, 0, 0, 50, 50, 50, 50, 50, 50, 0, 0, 0, 0,
        0, 0, 0, 0, 0];
```

Since I encoded a minimum number of steps as a bonus, the solver takes a different approach. As a consequence, the robot carries packages 1 and 2 at the same time, drops them off at location H and then proceeds to carry package 3 back to location B before returning to location A.