SAARLAND UNIVERSITY ARTIFICIAL INTELLIGENCE
Dr. Álvaro Torralba and Prof. Wolfgang Wahlster
Dr. Cosmina Croitoru, Daniel Gnad, Marcel Steinmetz
Yannick Körber, Michael Barz
Christian Bohnenberger, Sophian Guidara, Alexander Rath,
El Khansa Rekik, Julia Wichlacz, Anna Wilhelm

Practical Sheet 4.

Solutions due Tuesday, **July 3**, 23:59, in the Moodle system.

In this sheet you are going to model the RushHour game[1] as a planning task in PDDL. Initially, there is a number of cars placed on a grid. Each car occupies several tiles in the grid (see Figure 1 showing a possible initial configuration).

Figure 1: The RushHour game.

The goal is to get the red car out of the grid, which you can do by moving it to the end (rightmost extreme) of the row in which it is placed initially. Cars are placed either horizontally or vertically and can only move in that direction (i.e., cars placed horizontally only move left and right, vertically placed cars can only move up and down). Cars can only move one square in each step, and the position the car moves to has to be "clear".
Modeling a planning task consists of two parts: you need a PDDL file for the domain description and another PDDL file for each problem instance. We provide template files called `domain1.pddl`, `instance1.pddl`, and `domain2.pddl`, `instance2.pddl`, which already define the necessary predicates and actions for the model.

[1]`https://en.wikipedia.org/wiki/Rush_Hour_(puzzle)`

Your task is to fill in the semantics of the actions, and complete the initial state and goal definition. **You are not allowed to introduce new actions or predicates.** Descriptions of predicates and actions are shown in the below tables. Some PDDL examples are available here:

- `http://fai.cs.uni-saarland.de/hoffmann/PlanningForDummies.zip`
- `http://planning.domains/`

You can test your model by running FF[2] as follows: `ff -o domain.pddl -f instance.pddl`. FF is installed in the VM.

IMPORTANT Please add comments to your models. We will subtract points if you do not put any comments into your models, or the comments provided are insufficient or hard to follow.

Submission instructions Put all domain and problem files into an archive called "name1-name2-name3.zip", where "name1", "name2", "name3" are the family names of all authors. Additionally, add a file called "authors.txt" to the archive that contains one line per author, detailing the full name and matriculation number. Please do **not** add any subfolders to your archive. To upload the archive to the Moodle system, go to the corresponding assignment, click "Add submission", and upload it. Note that only one author per group needs to do the submission.

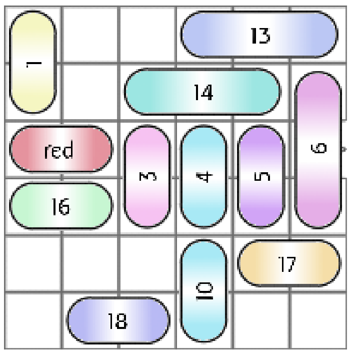

Figure 2: Initial State to be modeled.

[2]`https://fai.cs.uni-saarland.de/hoffmann/ff.html`

Exercise 5. (7 Points)

Write a PDDL model of the game that restricts the length of cars to 2 (small) or 3 (large) tiles. There are two types of objects, *cars* and *positions*. The grid is represented as a set of positions. The *adjacent* predicate is used to indicate which positions are next to each other. For example, "(ADJACENT loc1-1 loc2-1 loc3-1)" indicates that these three positions are in line (from left to right). There is also a fact "(ADJACENT loc3-1 loc2-1 loc1-1)" that connects them from right to left and similarly for any combination of three positions that are connected in a line (row or column). The predicate takes three locations because that way one can infer the orientation (i.e., you know that the three locations are either in the same row or the same column).
The defined predicates are:

Predicate	Description
`(adjacent ?p1 ?p2 ?p3 - position)`	Indicates that `?p1`, `?p2` and `?p3` are in line.
`(small ?c  - car)`	Indicates that car `?c` has length 2.
`(large ?c  - car)`	Indicates that car `?c` has length 3.
`(isOccupied ?p  - position)`	Indicates that there is a car at position `??p`.
`(containsCar ?p  - position ?c - car)`	Indicates that car `?c` is at position `?p`.

The actions that you need to define are:

Action	Description
`(move-small ?c - car ?p1 ?p2 ?p3 - position)`	Moves a small car placed on `?p1` and `?p2` to `?p3` (and `?p2`).
`(move-large ?c - car ?p1 ?p2 ?p3 ?p4 - position)`	Moves a small car placed on `?p1`, `?p2`, and `?p3` to `?p4` (and `?p2`, `?p3`).

Your task is to define the preconditions and effects of these actions, as well as to model the initial state and goal that encodes the game state depicted in Figure 2, where the goal is to put the red car in the rightmost position of the grid.

Exercise 6. (13 Points)

Model a more general version of the game that accepts cars of arbitrary length (≥ 2). The grid is represented in terms of `?x` and `?y` coordinates, where x and y are natural numbers between 1 and the width/height of the grid. We have two types of object: *car* and *num*. Note that *num* objects are no different than other objects in PDDL, so you cannot directly use any numeric operations on them. Instead, you need to use the predicate *plusone*, which is specified in the initial state in a similar way as the adjacency predicate in Exercise 5. Position 1-1 is the top left corner, and increasing the x coordinate means moving to the right whereas increasing the y coordinates means moving down.
The position of cars is given by their head and tail **(horizontal cars face to the right and vertical cars face to the bottom of the grid, so in increasing x/y direction)**. The defined predicates are:

Predicate	Description
`(plusone ?n1 ?n2 - num)`	Indicates that `?n2` = ?n1 + 1.
`(carTail ?c - car ?x ?y - num)`	Indicates that the tail of car `?c` is at position x-y.
`(carHead ?c - car ?x ?y - num)`	Indicates that the head of car `?c` is at position x-y.
`(isOccupied ?x ?y - num)`	Indicates that there is a car at position x-y.

The actions that you need to define are:

Action	Description
`(move-up ?c - car ?x ?y1 ?y2 ?y3 ?y4 - num)`	Moves a car whose head and tail are on x-y4 and x-y2 respectively, to a new head x-y3 and tail x-y1.
`(move-down ?c - car ?x ?y1 ?y2 ?y3 ?y4 - num)`	Moves a car whose head and tail are on x-y3 and x-y1 respectively, to a new head x-y4 and tail x-y2.
`(move-right ?c - car ?y ?x1 ?x2 ?x3 ?x4 - num)`	Moves a car whose head and tail are on x3-y and x1-y respectively, to a new head x4-y and tail x2-y.
`(move-left ?c - car ?y ?x1 ?x2 ?x3 ?x4 - num)`	Moves a car whose head and tail are on x4-y and x2-y respectively, to a new head x3-y and tail x1-y.

In the problem file, as before, you have to complete the initial state definition as given in Figure 2 and the goal, having the red car in the rightmost position of the grid.