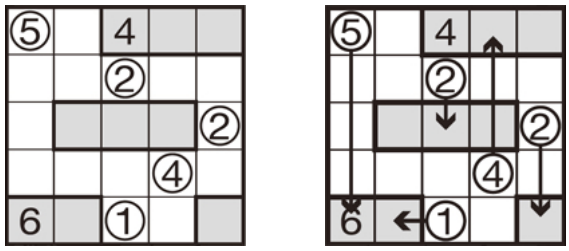
Mestrado (Integrado) em Engenharia Informática
Universidade Nova de Lisboa

Knowledge Representation and Reasoning Systems
Second Project (due on December 22)

Group 1

The first task of this project is to solve the Japanese grid puzzle Yosenabe[1] using ASP. To illustrate the rules of the game, consider a grid as follows:



Rules of the game:

1. Move all circles, vertically or horizontally, so they enter the gray areas.
2. Movements of a circle are shown by an arrow, with the tip of the arrow in a grey cell. The arrows do not bend, and do not cross other white circles or lines of other arrows, they go straight to their grey area.
3. The number in a gray area must be equal to the total sum of the numbers of the circles which enter the area. Empty gray areas may have any associated total sum, but at least one circle (arrow tip in the right image) must enter a gray area.

An instance of a grid is represented as follows (using the example grid above):

```
cell(1,1).      cell(1,2). ...   cell(5,4).      cell(5,5).      number(1,5,5).
area(1,1,1).    area(2,1,1).                    goal(1,6).      number(3,1,1).
area(2,3,2).    area(3,3,2).    area(4,3,2).                    number(3,4,2).
area(3,5,3).    area(4,5,3).    area(5,5,3).    goal(3,4).      number(4,2,4).
area(5,1,4).                                                    number(5,3,2).
```

Note that the first two arguments in facts over `area/3` and `number/3` provide grid cells, and the third an area identifier and a number to move, respectively. If an area is associated with a goal number, its identifier is reused as first argument in a fact over `goal/2`, and the goal is given by the second.

The moves in a solution shall be provided by atoms over the predicate `target/4` in a stable model, expressing a move in terms of coordinates of the initial cell as well as the cell to which a number is moved. Along with the instruction `#show target/4.`, a correct solution in file `yosenabe.lp` yields the following:

```
clingo yosenabe.lp instances/instance01.lp 0
clingo version 5.4.0
Reading from yosenabe.lp ...
Solving...
Answer:  1
target(1,5,1,1) target(3,1,2,1) target(3,4,3,3) target(4,2,4,5) target(5,3,5,1)
SATISFIABLE
```

[1] `https://www.janko.at/Raetsel/Yosenabe/` – You may want to change to English on the top-left corner.

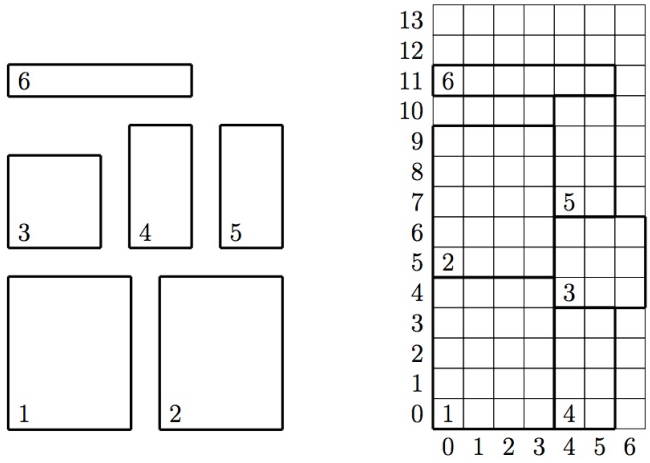
Write an ASP encoding such that, given an instance, the answer sets of the encoding and the instance correspond to the solutions of the problem represented by the instance.

In the `yosenabe.zip` archive, you can find fourteen example instances. You have to submit a file named `yosenabe.lp`, included as template in `yosenabe.zip`, that contains only one `#show` statement (`#show target/4.`), so that in the output only the atoms over predicate `target/4` are shown.

Hints Develop your encoding incrementally and check the additions. If in doubt whether rules are correct, the "–text" option of gringo/clingo allows for investigating the instantiation, for which small instances are advisable.

Group 2

Strip-packing is a combinatorial optimization problem. In the 2D case, a number of given rectangles need to be arranged without overlaps in a strip of fixed width such that the required height is minimal. For instance, the following six rectangles may be arranged in a strip of width 7 as follows:



Observe that the rectangles, labeled 1,...,6, are arranged without overlaps in the strip on the right. Their positions can be identified by the horizontal x- and the vertical y-coordinates of lower left corners. E.g., the respective x-position for rectangle 2 is 0, and its y-position is 5; given the width 4 and height 5, rectangle 2 then occupies cells up to column 3 and row 9. In view of strip width 7, the last column that can be occupied by any rectangle is referred to by x-coordinate 6. Similarly, we assume that an upper bound limits the number of available rows, where the bound 14 predefines y-coordinate 13 for the last row available above. However, the displayed arrangement of rectangles is optimal in occupying 12 rows only, while the rows referred to by y-coordinates 12 and 13 remain unused. The general objective is minimizing the required height of a strip, determined by the maximum y-position plus height over all rectangles.

An instance specifies parameters of a strip and rectangles in terms of facts as follows:

```
maxheight(14).          width(7).
rectangle(1,4,5).       rectangle(2,4,5).       rectangle(3,3,3).
rectangle(4,2,4).       rectangle(5,2,4).       rectangle(6,6,1).
```

Facts over `maxheight/1` and `width/1` provide an upper bound on the number of rows along with the width of a strip. The first argument in facts over `rectangle/3` are (integer) IDs for rectangles, each associated with the width and height determined by the second and third argument, respectively. The x- and y-coordinates of rectangles' lower left corners in an (optimal) solution shall be provided by atoms over the predicate `position/3` in a stable model, indicating either axis by constant "x" or "y", followed by the ID of a rectangle and the coordinate of a column or row, respectively. Along with the instruction `#show position/3.`, a correct solution in file `packing.lp` yields clingo behavior like the following (though intermediate optimization solutions may be presented depending on the encoding and the instance):

```
clingo packing.lp instances/instance01.lp
clingo version 5.4.0
Reading from packing.lp
Solving...
Answer:  1
position(x,1,0) position(y,1,0) position(x,2,0) position(y,2,5)
position(x,3,4) position(y,3,4) position(x,4,4) position(y,4,0)
position(x,5,4) position(y,5,7) position(x,6,0) position(y,6,11)
Optimization:  12
OPTIMUM FOUND
```

Write an ASP encoding such that, given an instance, a final optimal answer set of the encoding and the instance is presented, i.e., there must not be any other solution of lower height, while several optimal solutions may exist.

In the `packing.zip` archive, you can find twelve example instances. You have to submit a file named `packing.lp`, included as template in `packing.zip`, that contains only one `#show` statement (`#show position/3.`), so that in the output, only the atoms over predicate `position/3` are shown.

Hints Develop your encoding incrementally and check the additions. If in doubt whether rules are correct, the "–text" option of gringo/clingo allows for investigating the instantiation, for which small instances are advisable.

- Revising the solution w.r.t. an efficient encoding is highly recommended;
- The search space can be significantly pruned by breaking symmetries (forbidding redundant solutions), e.g., based on the IDs of rectangles of equal width and height (check, e.g., instance11);
- Deriving and explicitly stating some non-trivial yet valid lower bound on the required height may reduce the combinatorics of optimality proofs (check, e.g., instance12);
- some part of the solution may not be combinatorical, in which case it can be solved in, e.g., Python to improve efficiency - if you choose to do so please also submit the respective files.

Deliverables

You should send a single zip file labelled with your student numbers (individually or in groups of two students) to `mkn@fct.unl.pt`. The file should contain:

- One file for each ASP program developed, labelled appropriately.
- Any additional test files you may have used (besides the ones given), labelled appropriately.
- Any relevant material in Python (or similar) you may have (see above).
- A report (single PDF file) which should include the commented code of all ASP programs used (with references to the corresponding files), explaining the encoding optimizations applied, as well as any supplementary material.

Finally, note that different groups have to submit different solutions; in case of plagiarism all groups involved will fail the project.