Einführung in Wissensbasierte Systeme
WS 2023/24, 4.0 VU, 192.023
ASP Project

You have two turn-in possibilities for this project. The procedure is as follows: your solutions will be tested with automatic test cases after the first turn-in deadline and tentative points will be made available in **TUWEL**. You may turn-in (repeatedly) until the second turn-in deadline two weeks later, where the total points for your project are calculated as follows (p_i are the achieved points for the i-th turn-in, $i \in \{1, 2\}$):

- if you deliver your project at both turn-in 1 and 2, you get the maximum of the points and the weighted mean:

$$\max\left\{p_1, \quad 0.8 \cdot p_2, \quad \frac{p_1 + 0.8 \cdot p_2}{1.8}\right\};$$

- if you only deliver your project at turn-in 1: p_1;
- if you only deliver your project at turn-in 2: $0.8 \cdot p_2$; and
- 0, otherwise.

The deadline for the first turn-in is **Wednesday, December 06, 23:55**. You can submit multiple times, please submit early and do not wait until the last moment. The deadline for the second turn-in is **Wednesday, December 20, 23:55**.

This project deals with encoding problems in terms of ASP-Core-2 programs and using Clingo as solving tool (see `https://potassco.org/clingo` to download the system). The project is divided into Exercise 1 and 2, both of which give up to 6 points. The 6 points in Exercise 2 are distributed between its two problems as follows: Problem A gives up to 4 points, whereas Problem B gives up to 2 points. Thus, the maximum score amounts to **12 points**. In order **to pass** this project, you are required to attain at least **7 points**. There are no minimum points required on a specific problem. Detailed information on how to submit your project is given in Section 4. Make sure that you have named all files and all predicates in your encodings according to the specification. Failing to comply with the naming requirements might result in losing all points of the corresponding problem.

There will be a **dedicated online QA session** for the ASP project. The details can be found in **TUWEL**. Furthermore, for general questions on the assignment, please consult the **TUWEL** forum. Questions that reveal (part of) your solution should be discussed privately during the tutor sessions (see the time slots listed in **TUWEL**) or sent via email to:

`ewbs-2023w@kr.tuwien.ac.at`

---

1 Exercise 1: Attack on Rooks and Bishops

Given a $n \times n$ chessboard, c rook pieces, and d bishop pieces, place these pieces in the chessboard so that they do not attack each other.

Remember that in the game of chess:

- the rook can move/attack any number of squares horizontally (left or right), or vertically (up or down); and
- the bishop can move/attack any number of squares diagonally.

For clarity, see Figures 1 and 2 taken from Chess.com.

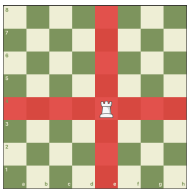

Figure 1: Rook

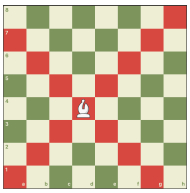

Figure 2: Bishop

The output of the ASP program must be given via:

- the binary predicate `rook(I,J)`, where `I` is a row and `J` is a column specifying the placement of a rook; and
- the binary predicate `bishop(I,J)`, where `I` is a row and `J` is a column specifying the placement of a bishop.

1.1 Example instance

Consider the following example instance:

```
% n x n chessboard, where n=8.
size(8).
% c rook pieces, where c=5.
n_rooks(5).
% d bishop pieces, where d=6.
n_bishops(6).
```

A possible solution is depicted in Figure 3.

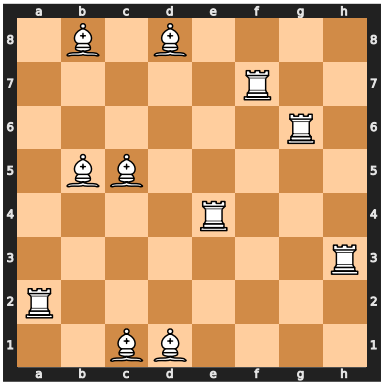

Figure 3: Possible solution for $n = 8$, $d = 6$, and $c = 5$.

The expected output (input facts omitted) of the corresponding ASP encoding is:

```
rook(2,1) rook(4,5) rook(7,6) rook(6,7) rook(3,8)
bishop(5,2) bishop(8,2) bishop(1,3) bishop(5,3)
bishop(1,4) bishop(8,4)
```

2 Exercise 2: Farm Location Problem

There is a set of farms $F = \{1, \ldots, n\}$ that supply products to a set of markets $M = \{1, \ldots, m\}$ for resale under the following conditions:

- each farm $j \in F$ produces a maximum number p_j of products it can supply;
- each farm $j \in F$ has an initial opening cost o_j for starting it;
- each market $i \in M$ has a request for r_i units of product;
- each market $i \in M$ has a maximum number s_i of farms that can supply it; and
- each *unit* of product supplied to a market $i \in M$ by a farm $j \in F$ incurs a transportation cost $c_{i,j}$.

2.1 Problem A: Feasibility

Write an ASP program that finds a feasible supply of products from farms to markets. The following constraints must be satisfied:

- the total quantity of products supplied by a farm $j \in F$ cannot exceed its maximum production p_j;

- the total quantity of products supplied to a market $i \in M$ must be exactly equal to its request r_i;
- products can be transported only from open farms;
- a farm is considered open if-and-only-if it supplies products to some market;
- the total number of farms supplying a market $i \in M$ cannot exceed the limit specified by s_i.

The output of the ASP program must be given via:

- the unary predicate `f_open(J)`, where `J` is an open farm; and
- the ternary predicate `transport_prod(J, I, Q)`, where `Q` is the quantity of products transported from a farm `J` to a market `I`.

Note that we expect a single atom `transport_prod(j,i,q)` for the supply of `q` products of farm `j` to market `i`. Giving multiple `transport_prod` atoms for the same pair of farm and market in your answer sets, like for example, `transport_prod(j,i,q1)` and `transport_prod(j,i,q2)`, will confuse our automated tests.

2.1.1 Example instance

Consider the example instance:

```
f_prod(1, 20;
       2, 20;
       3, 20).
% Meaning that all the farms
% can produce at most 20 units

% Quantity of products requested by markets
% and maximum number of different farms as suppliers.
m_req(1, 2, 2;
      2, 15, 1;
      3, 2, 1;
      4, 21, 2).
% Market 2 has a necessity of r_2=15 product units
% and accept a maximum of s_2=1 farm as supplier

% Opening cost for farms.
f_open_cost(1, 4;
            2, 10;
            3, 20).
% Farm 1 has an opening cost of 4
```

```
% Cost per unit of transporting products from each farm to each
% market.
transport_cost(1, 1, 4;
               2, 1, 2;
               3, 1, 5;
               4, 1, 2;

               1, 2, 2;
               2, 2, 5;
               3, 2, 3;
               4, 2, 2;

               1, 3, 3;
               2, 3, 4;
               3, 3, 5;
               4, 3, 3).
% The third row says that the connection from farm 1 to market 3
% costs 5.
```

Then the following is a possible solution:

```
f_open(1) f_open(2)
transport_prod(1,2,15)
transport_prod(1,4,5)
transport_prod(2,1,2)
transport_prod(2,3,2)
transport_prod(2,4,16)
```

2.2 Problem B: Optimization

Suppose a feasible solution to Problem A is given as follows: the integer matrix $x_{i,j}$ denotes the quantity of products supplied to a market $i \in M$ by a farm $j \in F$ and the binary vector $y_j \in \{0, 1\}$ denotes whether farm $j \in F$ is open. Write a new ASP program which not only solves Problem A (you can copy the rules), but also adds an objective function f to be minimized. The objective f is obtained as the sum of the transportation costs f_{TC} and the fixed opening costs f_O. Note that a farm is considered open if and only if it supplies at least one market with a strictly positive quantity of products. More formally:

$$f = f_{TC} + f_O$$

where

$$f_{TC} = \sum_{\substack{i \in M, \\ j \in F}} x_{i,j} \cdot c_{i,j}$$

and

$$f_O = \sum_{j \in F} y_j \cdot o_j$$

In our input format the transportation cost is provided by a ternary predicate `transport_cost(I, J, C)`, where `C` is the transportation cost from market `I` to farm `J`. The opening cost `C` of farm `J` is given by `f_open_cost(J, C)`.

2.2.1 Example instance

The solution to the Problem A is also optimal. The total cost amounts to 96.

3 Hints and final remarks

- You can find the example instances discussed here in **TUWEL**.
- The order of predicates in Clingo's output is irrelevant.
- Your ASP programs must be completely self-contained.
- You are allowed to introduce any number of additional predicates within your ASP program.
- Your encoding must not output the same combination of output-predicates when asking the solver for multiple models. This can happen if you use auxiliary predicates internally that make the models be different, although they look equal when only showing the output-predicates (e.g., by using `#show`).
- It might be useful for you to restrict Clingo's output to some specific predicates using the command `#show`. For example, you can write:

  ```
  #show bishop/2.
  ```

  in your example instances to print only those atoms over the `bishop` predicate with arity 2.[1] **However, make sure that you use `#show` only in your example instances and not in your actual ASP programs!**
- Use Clingo v5.4.0 or later for testing your encodings. Refer to the following links for Clingo installation instructions on Windows, Linux and macOS:
  - `https://potassco.org/clingo/`
  - `https://youtu.be/H7NmClCXONY?si=HMHd1FpXFpZg_Imq`
- The ASP-Core-2 specification available at:
  `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.03c.pdf`,
  and the Clingo's guide at:
  `https://github.com/potassco/guide/releases/download/v2.2.0/guide.pdf`
  might be useful.

[1] Note that `#show` is a Clingo specific keyword and is not defined by the ASP-Core-2 standard.

4 Submission Information

Submit your solution by uploading a ZIP file to the **TUWEL** platform ensuring that it contains the following files:

- `chess.lp` for Exercise 1,
- `feasibility.lp` for Exercise 2: Problem A, and
- `optimization.lp` for Exercise 2: Problem B.

Every encoding should be self-contained, so it can be tested as follows

```
clingo <encoding> <instance>
```

where `<encoding>` is either `chess.lp`, `feasibility.lp`, or `optimization.lp` and `<encoding>` is an instance file, like for example, `toy_instance_chess.lp`.

In particular, this means that `optimization.lp` must also include the feasibility constraints specified in Section 2.1.

Make sure to properly rename your ASP programs as described above, and that your ZIP file contains no subfolders.