Answer Set Programming Modeling Competition: 20 July 2014, Vienna, Austria

A team consists of one to three members, using one computer. The number of correct submissions (the last submitted correct one for a particular problem) within 90 minutes is used as the first ranking criterion. Incorrect submissions cost nothing. For each correct submission, the time it took to submit it (starting from some arbitrary time point) is added up: this total is used as a tie breaker in case teams end up with the same number of correct solutions. If that doesn't break all ties, the organizers can use any other criterion like performance, space usage, coin tossing, etc. Efficiency of your programs is not important, but if your program fails to finish in a reasonable time, it is considered incorrect.

Submitting is done by putting your (presumed) solution on a USB stick, not hidden in a subdirectory. The name of the file that contains your submitted solution must be the same as given in parentheses in the header of a task description: so, for the first problem, you make a file named `function.asp`. Programs must be written in ASP-Core-2 (version 2.03b[1]), pragmatically taken to be processible with *clingo* (version 4.3.0[2]). If we can't read your stick/file, bad luck to you. The input to your program is always in the form of facts; see examples in task descriptions. On any input, your program must specify (optimal) solutions, determined by the output predicates given in task descriptions. Arbitrary auxiliary predicates can be defined in addition but will be ignored in correctness evaluation.

Keep in mind: correctness matters, submission time matters, task order doesn't matter. Good luck!

1 Function Fun (`function.asp`)

Consider a function f on non-negative integers n:

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 & \text{if } n = 1 \\ 3 & \text{if } n = 2 \\ 2*f(n-1) - f(n-3) & \text{if } n > 2 \end{cases}$$

For instance, the first few outcomes of f are as follows:

$$\begin{array}{rcl} f(0) & = & 1 \\ f(1) & = & 2 \\ f(2) & = & 3 \\ f(3) & = & 5 \;=\; 2*f(2) - f(0) \\ f(4) & = & 8 \;=\; 2*f(3) - f(1) \\ f(5) & = & 13 \;=\; 2*f(4) - f(2) \\ f(6) & = & 21 \;=\; 2*f(5) - f(3) \\ f(7) & = & 34 \;=\; 2*f(6) - f(4) \\ & \vdots & \end{array}$$

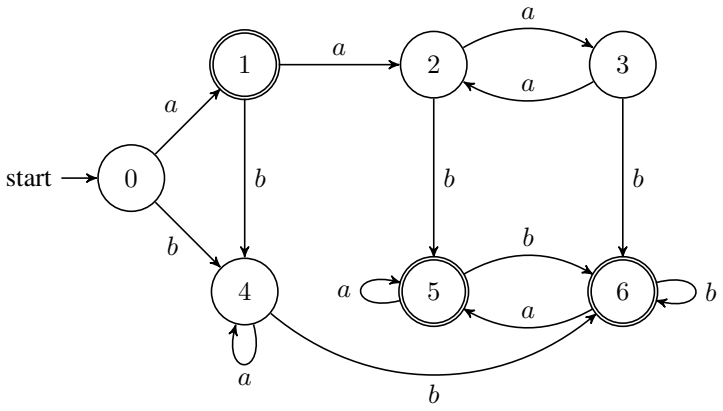
We represent an argument n of interest by a fact '`input(`n`).`', such as '`input(7).`' in instance file `instance07.asp`. Along with the instruction '`#show output/1.`', a correct solution in file `function.asp` yields the following *clingo* behavior:

```
$ clingo-4.3.0 instance07.asp function.asp
clingo version 4.3.0
Reading from instance07.asp ...
Solving...
Answer: 1
output(34)
SATISFIABLE
```

[1] `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.03b.pdf`
[2] `http://sourceforge.net/projects/potassco/files/clingo/4.3.0/`

2 Automata Reduction (`automaton.asp`)

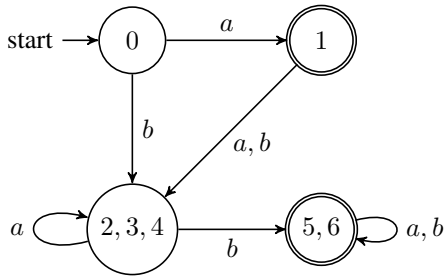
A Deterministic Finite Automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, consisting of a finite set Q of states, an alphabet Σ, a total transition function $\delta : Q \times \Sigma \to Q$, a start state $q_0 \in Q$, and a set $F \subseteq Q$ of accept states. Given any word $w \in \Sigma^*$, starting from q_0, the DFA processes w symbol by symbol and performs state transitions according to δ. If the final state reached after processing w belongs to F, w is accepted, and rejected otherwise. DFAs can be nicely visualized as directed graphs with labeled arcs, like the following:



Every DFA has a unique minimal equivalent DFA, which can be obtained by first determining distinguishable states, and then merging classes of indistinguishable into one state each. In more detail:

1. Any accept state $q \in F$ must be distinguished from every non-accept state $q' \in Q \setminus F$.
2. Any two states $q, q' \in Q$ such that, for some symbol $\sigma \in \Sigma$, $\delta(q, \sigma)$ and $\delta(q', \sigma)$ are distinguishable must be distinguished as well.

For the DFA depicted above, the accept states 1, 5, and 6 must be distinguished from 0, 2, 3, and 4. Then, 0 must also be distinguished from 2, 3, and 4 because $\delta(0, b) = 4$ is distinguishable from $\delta(2, b) = 5$ and $\delta(3, b) = \delta(4, b) = 6$. Likewise, 1 must be distinguished from 5 and 6 given that $\delta(1, a) = 2$ (or $\delta(1, b) = 4$) is distinguishable from $\delta(5, a) = \delta(6, a) = 5$ (or $\delta(5, b) = \delta(6, b) = 6$). The members of the remaining classes $\{0\}$, $\{1\}$, $\{2, 3, 4\}$, and $\{5, 6\}$ are indistinguishable, and each class can be merged into a single state to obtain the following minimal equivalent DFA:



In ASP, we represent the states Q, accept states $F \subseteq Q$, and the alphabet Σ of a DFA by facts like:

```
state(0). state(1). state(2). state(3). state(4). state(5). state(6).
accept(1). accept(5). accept(6).
symbol(a). symbol(b).
```

We implicitly assume 0 to be the start state q_0, and the transition function δ is given by facts as follows:

```
delta(0,a,1). delta(2,a,3). delta(4,a,4). delta(6,a,5).
delta(0,b,4). delta(2,b,5). delta(4,b,6). delta(6,b,6).
delta(1,a,2). delta(3,a,2). delta(5,a,5).
delta(1,b,4). delta(3,b,6). delta(5,b,6).
```

An answer set shall represent the unique minimal equivalent DFA by atoms of the form `merge(`q`,`q'`)`, where q is the lexicographically smallest state (according to *clingo*'s term order) of a non-singleton class and q' ranges over all lexicographically greater states of the class. For example, when the facts above are given in file `instance03.asp` (and '`#show merge/2.`' is used), a correct solution leads to the following (explicit) representation of the non-singleton classes $\{2,3,4\}$ and $\{5,6\}$:

```
$ clingo-4.3.0 instance03.asp automaton.asp
clingo version 4.3.0
Reading from instance03.asp ...
Solving...
Answer: 1
merge(2,3) merge(2,4) merge(5,6)
SATISFIABLE
```

3 Meeting Point (`meeting.asp`)

Consider a grid puzzle as shown below on the left-hand side:

The task is to construct non-crossing and non-overlapping paths that start from given numbered grid cells, having coordinates (1,1), (2,1), and (3,3) in this example, and meet each other in a single cell. The paths can make horizontal and vertical transitions between neighboring cells but must not pass numbered grid cells (except for their origins). Most importantly, for each path, the number in its origin cell prescribes the number of turns, that is, alternation between horizontal and vertical transition or vice versa, the path has to make. A corresponding solution is displayed on the right-hand side above. Observe that the paths from cell (1,1) and (2,1) make one turn each at cell (1,2) or (3,1), respectively, while the path from (3,3) makes no turn on its way to the common meeting point (3,2).

In facts, we represent the cells of a quadratic grid like the one on the left-hand side above along with numbered cells as follows:

```
cell(1,3). cell(2,3). cell(3,3).
cell(1,2). cell(2,2). cell(3,2).
cell(1,1). cell(2,1). cell(3,1).
number(1,1,1). number(2,1,1). number(3,3,0).
```

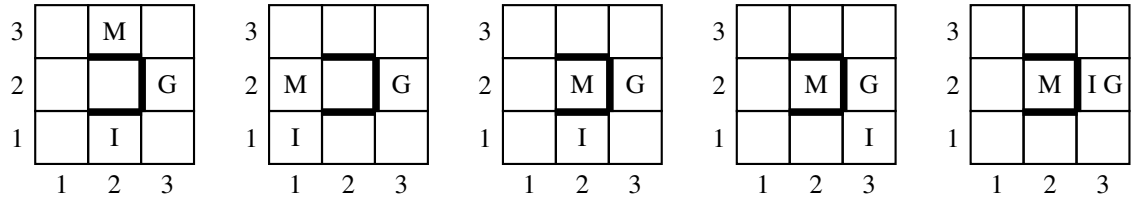
A solution is represented in terms of transitions of all paths, given by instances of the predicate `link/4`. For instance, given the facts in file `instance01.asp` (along with '`#show link/4.`'), the paths on the right-hand side above shall be calculated by *clingo* like this:

```
$ clingo-4.3.0 instance01.asp meeting.asp
clingo version 4.3.0
Reading from instance01.asp ...
Solving...
Answer: 1
link(1,1,1,2) link(1,2,2,2) link(2,2,3,2) \
link(2,1,3,1) link(3,1,3,2) link(3,3,3,2)
SATISFIABLE
```

Finally, note that an answer obtained by adding the atoms `link(1,3,2,3)` and/or `link(2,3,1,3)`, referring to blank cells not used by the three paths, would still represent the same solution. Although such redundant atoms should be avoided in general, we won't regard their accidental inclusion as incorrectness.

4 Cretan Labyrinth (`minotaur.asp`)

This grid puzzle, best described by example, is a task for heroes:



On the very left, imagine a grid-shaped labyrinth with walls between the cells (2,2) and (2,1), (2,2) and (2,3), as well as (2,2) and (3,2). The hero, denoted by "I" at cell (2,1), must reach the goal cell indicated by "G" at (3,2). The task is dangerous because a hungry minotaur, denoted by "M" at cell (2,3), is hunting the hero and can move two steps in horizontal or vertical direction at the same time the hero needs for one step. Fortunately, the minotaur isn't as smart as the hero and always tries to approach towards her in horizontal direction, and in vertical direction if the former is not possible, due to being at the same height already or being blocked by a wall. Accordingly, when the hero makes a first step to the left, the minotaur moves left and then downwards, resulting in the second situation where both face each other at cells (1,1) and (1,2). Now the hero returns to cell (2,1), and since a step to the right is feasible, the minotaur opportunistically moves to cell (2,2), from where it cannot approach further due to the wall in-between the cells. Thus, the minotaur is trapped, while the hero can safely reach the goal cell at (3,2) by making two more steps. Note that the minotaur must not reach the cell with the hero, also if the hero is at the goal cell. As labyrinths can be tricky, it may sometimes be helpful that the hero stays at a cell in order to lure the minotaur into a trap.

In facts, we represent a rectangular grid with walls like the one on the left-hand side above along with cells of the hero, the goal, and the minotaur as well as a (maximum) number of steps as follows:

```
cell(1,3). cell(2,3). cell(3,3).
cell(1,2). cell(2,2). cell(3,2).
cell(1,1). cell(2,1). cell(3,1).
wall(2,1,2,2). wall(2,2,2,3). wall(2,2,3,2).
wall(2,2,2,1). wall(2,3,2,2). wall(3,2,2,2).
init(2,1). goal(3,2). mino(2,3).
maxsteps(4).
```

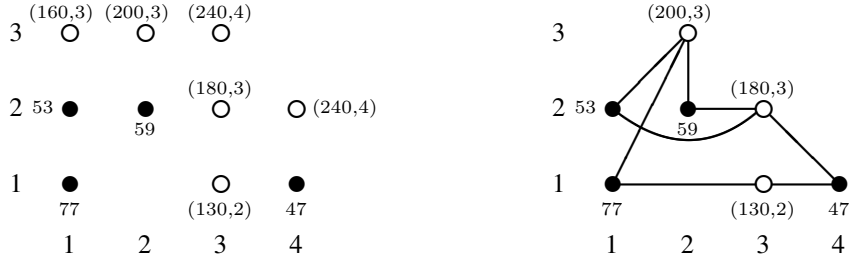
Note that instances of the predicate `wall/4` are symmetric, e.g., the fact '`wall(2,1,2,2).`' is accompanied by '`wall(2,2,2,1).`'

A solution shall be expressed in terms of atoms of the form `at(`x`,`y`,`i`)`, providing the cells of the hero for $0 \leq i \leq n$, where n is given by a fact '`maxsteps(`n`).`' Clearly, at step 0, the hero must be at the initial cell (x_0,y_0) determined by '`init(`x_0`,`y_0`).`', and the goal cell (x_n,y_n) specified by '`goal(`x_n`,`y_n`).`' must be reached at the last step n. The cells at successive steps must stay the same or be horizontally or vertically adjacent and not be separated by a wall. The minotaur's movements are predetermined by the hero's cells at steps $1 \leq i \leq n$, and the minotaur must not catch the hero on her way. For instance, given the facts in file `instance00.asp` (along with '`#show at/3.`'), a correct solution yields the following *clingo* behavior:

```
$ clingo-4.3.0 instance00.asp minotaur.asp
clingo version 4.3.0
Reading from instance00.asp ...
Solving...
Answer: 1
at(2,1,0) at(1,1,1) at(2,1,2) at(3,1,3) at(3,2,4)
SATISFIABLE
```

5 Fiber Network (`network.asp`)

In the final task, we turn our attention to the construction of an optimal fiber network connecting customers to gateways. For illustration, consider the following scenario:



The filled circles on the left-hand side stand for customers or leaf nodes, respectively, each of which produces a certain load, e.g., 77 in case of the leaf at coordinates (1,1). The white circles represent possible gateways, limited to some maximum load by customers as well as a capacity of customers to be connected. For instance, the maximum admissible load for the possible gateway at coordinates (3,1) is 130, and its capacity is limited to 2 leaves. The task is to connect every leaf to two gateways such that the load and cardinality restrictions of each gateway are not exceeded. A solution using three gateways at (2,3), (3,1), and (3,2) is shown on the right-hand side above. The gateway at (2,3) receives the combined load $77 + 53 + 59 = 189$ from its 3 connected leaves. Similarly, the 2 leaves connected to (3,1) amount to a load of $77 + 47 = 124$, and the loads of the 3 leaves connected to (3,2) sum up to $53 + 59 + 47 = 159$. Observe that accumulated loads stay within the limits of the gateways and that the numbers of connected leaves do not exceed the gateways' capacities.

In order to guarantee reliable access, the maximum number of leaves that are disconnected when two arbitrary gateways fail simultaneously is of interest. For the network on the right-hand side above, the two gateways at (2,3) and (3,2) are most critical because the leaves at (1,2) and (2,2) become disconnected when both gateways fail. Hence, the maximum number of leaves that can become disconnected in case of a double-fault is 2; this number can be subject to a design restriction. In addition, the number of gateways to use is a design parameter, which is 3 in the above case, and leaves must be connected to gateways only. For instance, the possible gateway at coordinates (1,3) on the left is unused in the network on the right.

Importantly, the total length of connections between leaves and their gateways is subject to minimization, where we use the measure $(x-x')^2+(y-y')^2$ for the distance between a leaf at (x,y) connected to a gateway at (x',y'). In this way, the connections of the leaf at (1,1) to the gateways at (2,3) and (3,1) amount to the distance measures $1 + 4 = 5$ and 4. One can check that the sum of distances over all connections is 20, providing an estimate of the global network quality. As it turns out, the network on the right-hand side above is optimal, that is, no network of strictly better quality or smaller sum of distances, respectively, can be constructed within the given resource limits.

We represent an instance like the one on the left-hand side above by facts as follows:

```
leaf(1,1,77). leaf(1,2,53). leaf(2,2,59). leaf(4,1,47).
node(1,3,160,3). node(2,3,200,3). node(3,1,130,2).
node(3,2,180,3). node(3,3,240,4). node(4,2,240,4).
gateways(3). limit(2).
```

Facts of the form '`leaf(`x`,`y`,`l`).`' provide the coordinates (x,y) of leaves along with associated load l. Similarly, facts '`node(`x`,`y`,`b`,`c`).`' specify the maximum combined load b and maximum number c of connected leaves for possible gateways with coordinates (x,y). The number n of gateways to use is determined by a fact of the form '`gateways(`n`).`' Moreover, the maximum number m of disconnected leaves in case of arbitrary double-faults is restricted via a fact '`limit(`m`).`'

A solution is represented by atoms of the form `gateway(`x`,`y`)`, providing the coordinates (x,y) of used gateways, as well as `wire(`x`,`y`,`x'`,`y'`)` indicating that a leaf at (x,y) is connected to a gateway at (x',y'). Recall that any leaf must be connected to two gateways and that the sum of distances over connections should be globally minimal for the solution represented by the last answer set calculated

by *clingo*. Given the above facts in file `instance00.asp` (along with '`#show gateway/2.`' and '`#show wire/4.`'), an optimal solution shall be calculated, e.g., like this:

```
$ clingo-4.3.0 instance00.asp network.asp
clingo version 4.3.0
Reading from instance00.asp ...
Solving...
Answer: 1
wire(1,1,1,3) wire(1,1,4,2) wire(1,2,3,3) wire(1,2,4,2) \
wire(2,2,3,3) wire(2,2,4,2) wire(4,1,1,3) wire(4,1,4,2) \
gateway(1,3) gateway(3,3) gateway(4,2)
Optimization: [?]
Answer: 2
wire(1,1,2,3) wire(1,1,4,2) wire(1,2,3,1) wire(1,2,4,2) \
wire(2,2,2,3) wire(2,2,4,2) wire(4,1,3,1) wire(4,1,4,2) \
gateway(2,3) gateway(3,1) gateway(4,2)
Optimization: [?]
Answer: 3
wire(1,1,1,3) wire(1,1,4,2) wire(1,2,1,3) wire(1,2,4,2) \
wire(2,2,3,1) wire(2,2,4,2) wire(4,1,3,1) wire(4,1,4,2) \
gateway(1,3) gateway(3,1) gateway(4,2)
Optimization: [?]
Answer: 4
wire(1,1,3,1) wire(1,1,3,3) wire(1,2,3,2) wire(1,2,3,3) \
wire(2,2,3,2) wire(2,2,3,3) wire(4,1,3,1) wire(4,1,3,2) \
gateway(3,1) gateway(3,2) gateway(3,3)
Optimization: [?]
Answer: 5
wire(1,1,2,3) wire(1,1,3,2) wire(1,2,2,3) wire(1,2,3,2) \
wire(2,2,2,3) wire(2,2,4,2) wire(4,1,3,2) wire(4,1,4,2) \
gateway(2,3) gateway(3,2) gateway(4,2)
Optimization: [?]
Answer: 6
wire(1,1,2,3) wire(1,1,3,1) wire(1,2,2,3) wire(1,2,3,2) \
wire(2,2,2,3) wire(2,2,3,2) wire(4,1,3,1) wire(4,1,3,2) \
gateway(2,3) gateway(3,1) gateway(3,2)
Optimization: [?]
OPTIMUM FOUND
```