

Lab 3: Answer Set Programming

Answer Set Programming (ASP) is a declarative programming language based on logic. This lab session introduces various well-known problems for you to model in ASP.

Deliverable The core of the lab assignment is in Sections 2 and 3. If you finish early, feel free to tackle the problems in Section 4.

Submit one .lp file for each problem with the naming conventions mentioned in their respective sections. Also submit a short PDF report containing a brief explanation of the intuition behind each implementation, and your answers to the questions, for the exercises that have questions.

1 Quick reminders about ASP

This section serves as a quick reminder of some of the elements that you've seen in class, and introduces a few additional functions and syntax elements that you will require to complete the exercises.

Basics In what follows, some examples are presented to recall the basic syntax of ASP.

- **Facts:** `edge(a,b).` states that there is an edge from vertex `a` to vertex `b`.
- **Rules:** `path(X,Y) :- edge(X,Y).` states that if there is an edge from `X` to `Y`, then there is a path from `X` to `Y`.
- **Choice rules:** `{selected(X)} :- vertex(X).` allows each vertex to be optionally selected.
- **Constraints:** `:- selected(X), selected(Y), edge(X,Y).` forbids adjacent vertices from both being selected.

Aggregates Aggregates in ASP have the form `f{e} o t` where

- `f` is an aggregation function (`#count`, `#sum`, `#min`, `#max`, etc)
- `e` is an aggregate element, of the form `t1, ..., tm : phi`, where `ti` are terms and `phi` is a conjunction of literals.
- `o` is a comparison operator (`≤`, `≥`, `=`, etc.)
- `t` is a (ground) term.

Examples:

- `#count{X : selected(X)} = 3`: ensure that exactly 3 vertices are selected
- `#sum{W,X : edge(X,Y), weight(X,Y,W)} <= 10`: select edges incident to X with total weight at most 10

Others

- **Anonymous variables:** Use the underscore `_` when you don't need to reference a specific value.

Example: `p(X) :- q(X,_)`. will make `p(X)` true when there exists some element `Y` such that `q(X,Y)` is true.

- **Displaying predicates:** Use `#show` to control which predicates are displayed in the output. This is particularly useful to hide auxiliary predicates used during computation.

Example: `#show selected/1`. will only show atoms of the form `selected(X)` that are true in the solution found.

Software Through this lab, we will use Clingo as a solver for ASP programs. It can be installed on most platforms through your favorite package manager. ASP program files usually end with `.lp`.

```
clingo your_file.lp
```

Installation instructions You can find the general installation instructions at the following link: <https://github.com/potassco/clingo/releases/>

Recommended You can use Clingo through its Python package. It can be installed using either Conda or Miniconda (see the link above), or by using PyPI (`pip install clingo`).

If you choose this way, we provide file `run_clingo.py`, so that you can run ASP files directly ¹:

```
run_clingo.py file.lp
```

Ubuntu

```
sudo apt-get install gringo
```

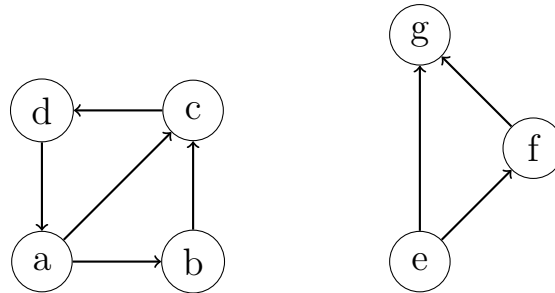
Windows

Download and install the file at the following location: <https://github.com/potassco/clingo/releases/download/v5.4.0/clingo-5.4.0-win64.zip>

¹Thanks to Oliver Jørgensen for providing the file

2 Preliminaries: Graph Representation

This section allows you to practice the basic syntax and some of the common patterns in ASP, through basic reasoning on graphs. The code below models the (non-connected) graph depicted below.



```

edge(a,b). edge(b,c). edge(c,d). edge(d,a). edge(a,c).
edge(e,f). edge(f,g). edge(e,g).
  
```

Tasks Complete the following exercises in file `1_graphs_intro.lp`:

1. **Orientation removal:** The graph above is directed. To make it undirected, we could add a fact `edge(b,a).` for each fact `edge(a,b).` in the original definition, but doing so manually is tedious and can be automated.
Write rules to make the graph provided above undirected.
2. **Vertices extraction:** Write a rule that identifies all vertices of the graph.
3. **Inner nodes:** Inner nodes are vertices that have both incoming and outgoing edges. Write a rule that identifies them.
4. **Leaf nodes:** Leaf nodes are vertices that have at most one incident edge. Write a rule that identifies them.
5. **Reachability:** Define a predicate `reachable` that encodes the transitive closure of `edge`. In other words, `reachable(a,b).` should be true iff there exists a path from `a` to `b`.
6. **Connected components:** Write a set of rules that counts the number of connected components in the graph.

Hints: Define a predicate `component(X,M)` such that every vertex `X` is associated to its connected component `M`, which is the smallest vertex in the component.

3 Problems on Graphs

This section offers various well-known problems for you to encode in ASP. All of them take a graph as input, except for the first one. For some of these problems, you will have to reuse some of the techniques seen in the previous section. **Note that all graphs in this section are undirected.**

3.1 Hitting Set Problem

Let X be a collection of elements, and $\Sigma \subseteq 2^X$ a collection of subsets of X . The hitting set problem consists in finding a subset $S \subseteq X$ of elements, which intersects every set of Σ .

Tasks Complete the following two exercises in file `2_1_hitting_set.lp`:

1. Implement the **decision version**: determine if a hitting set of size at most k exists.
2. Implement the **optimization version**: find the minimum hitting set.

Hint:

- `#minimize{1, X : p(X)}` can be used to indicate to the solver to minimize the number of constants satisfying p .

Test instance:

$$X = \{1, 2, 3, 4, 5, 6\}$$

$$\Sigma = \{\{1, 2, 5\}, \{2, 3, 5\}, \{3, 4, 6\}, \{1, 4\}\}$$

```
subset(s1, 1). subset(s1, 2). subset(s1, 5).
subset(s2, 2). subset(s2, 3). subset(s2, 5).
subset(s3, 3). subset(s3, 4). subset(s3, 6).
subset(s4, 1). subset(s4, 4).
```

3.2 Dominating Set Problem

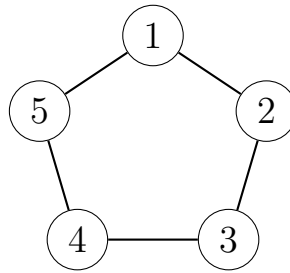
In a graph, a dominating set is a subset of vertices such that every vertex is either in the dominating set or adjacent to at least one vertex in the dominating set. More formally, for a graph $G = (V, E)$, a dominating set D is a subset of vertices $D \subseteq V$ such that for every vertex $u \in V \setminus D$, there exists $v \in D$ with $\{u, v\} \in E$.

```
edge(1,2). edge(2,3). edge(3,4). edge(4,5). edge(5,1).
```

Tasks Complete file `2_2_dominating_set.lp`:

1. Implement the **decision version**: determine if a dominating set of size at most k exists.
2. Implement the **optimization version**: find the minimum dominating set.

Test instance:



```
edge(1,2). edge(2,3). edge(3,4). edge(4,5). edge(5,1).
```

3.3 Bipartite Graph Testing

A graph is bipartite if its vertices can be colored with two colors such that no two adjacent vertices have the same color.

Tasks Complete file `2_3_bipartite.lp`:

1. Write the definition of a bipartite graph more formally, starting from the definition above.
2. Implement a script that checks if a graph is bipartite.
3. *(Optional)* If your script allows it, extend it so that it can check if a graph is n -colorable.

Test instances:

```
% Bipartite graph
edge(1,2). edge(2,3). edge(3,4). edge(4,1).

% Non-bipartite graph
edge(1,2). edge(2,3). edge(3,1).
```

3.4 Spanning Tree

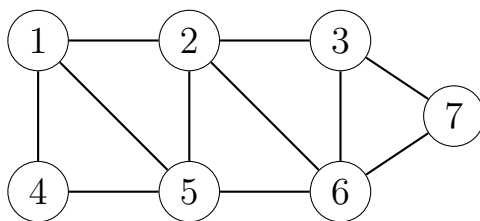
A spanning tree of a connected graph $G(V, E)$ is a subgraph T of G that is a tree, and whose vertices are exactly the ones of G .

Tasks Complete file `2_4_spanning_tree.lp`:

1. Write an ASP script that extracts a spanning tree from a given graph.

Hint: A spanning tree is a subgraph of $|V| - 1$ edges such that all nodes are reachable from any other.

Test instance:



```
edge(1,2). edge(2,3). edge(1,4). edge(2,5). edge(3,6). edge(4,5).
edge(5,6). edge(1,5). edge(2,6). edge(3,7). edge(6,7).
```

4 Advanced Problems

The following problems offer a more general landscape of the capabilities of ASP, and are less guided. However, they can all be solved with the tools and techniques used in the previous sections.

Feel free to tackle the problems of your choice, regardless of the order.

4.1 CNF Satisfiability

The Boolean satisfiability problem (SAT) is a seminal problem in computer science, both from a theoretical and a practical point of view. It consists in finding whether a propositional logic formula, provided in conjunctive normal form, admits a model (i.e. a satisfying assignment).

Tasks In a file named `3_1_sat.lp`, implement a program that solves the Boolean satisfiability problem:

- When a model for the input formula exists, a satisfying assignment should be output.
- When no model for the input formula exists, the program should be unsatisfiable.

Test instances:

Satisfiable formula

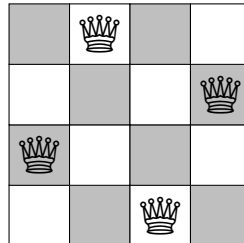
$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

Unsatisfiable formula

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_3) \wedge (x_3)$$

4.2 n -Queens Problem

The n -queens problem consists in placing n queens on an $n \times n$ chessboard, such that no two queens attack each other. Otherwise said, no two queens should be on the same row, column, or diagonal. This problem was also introduced in the lecture on Constraint Satisfaction Problems (CSP), but the encoding of the problem in ASP might require different ideas.



Tasks In a file named `3_2_n_queens.lp`, implement a program that, given an integer n as input, solves the n -queens problem:

- Most computers should have no problem finding an assignment for $n = 8$, and even higher values of n .
- You can pass to Clingo the number of solutions that you want it to find. When given 0, Clingo will output all solutions (ex: `clingo 3_2_n_queens.lp 0`). For $n = 8$, you should find 92 models.

4.3 Scottish Club Problem

There exists in Scotland a very exclusive club that follows these rules:

- Every non-Scottish member wears red socks
- Every member either wears a kilt or doesn't wear red socks
- Married members don't go out on Sunday
- A member goes out on Sunday if and only if they are Scottish
- Every member who wears a kilt is Scottish and married
- Every Scottish member wears a kilt

Tasks In a file named `3_3_scottish.lp`, implement this set of rules:

1. Find all possible sets of characteristics of a member of this club.
2. Discuss the result.