Lecture Seven: Answer Set Programming - Problem Solving

301315 Knowledge Representation and Reasoning

ASP Solvers

So far, we have learnt ASP programming with respect to query answering: We write a ASP program to represent the underlying knowledge base of the domain, then we compute ASP answer sets in order to answer various queries, i.e., $\Pi \models q$.

Example

Π:

$edge(a, b).$
$edge(b, c).$
$link(X, Y) \leftarrow edge(X, Y).$
$link(X, Z) \leftarrow link(X, Y), edge(Y, Z).$

We would like to know whether $\Pi \models link(a, c)$?

ASP Solvers

- ASP is beyond query answering.
- ASP Paradigm - use ASP to solve computational problems by reducing them to finding answer sets of ASP programs.
- In principle, any NP-complete problem can be solved this way using ASP without disjunction.
- With disjunction, we can solve more-complex problems.

ASP Solvers

- *Clingo* at `https://potassco.sourceforge.net/`
- *DLV* at `https://www.dlvsystem.com/`
- Both systems have their advantages

Finding Hamiltonian Cycles - What is a Hamiltonian Cycle

What is a Hamiltonian cycle?

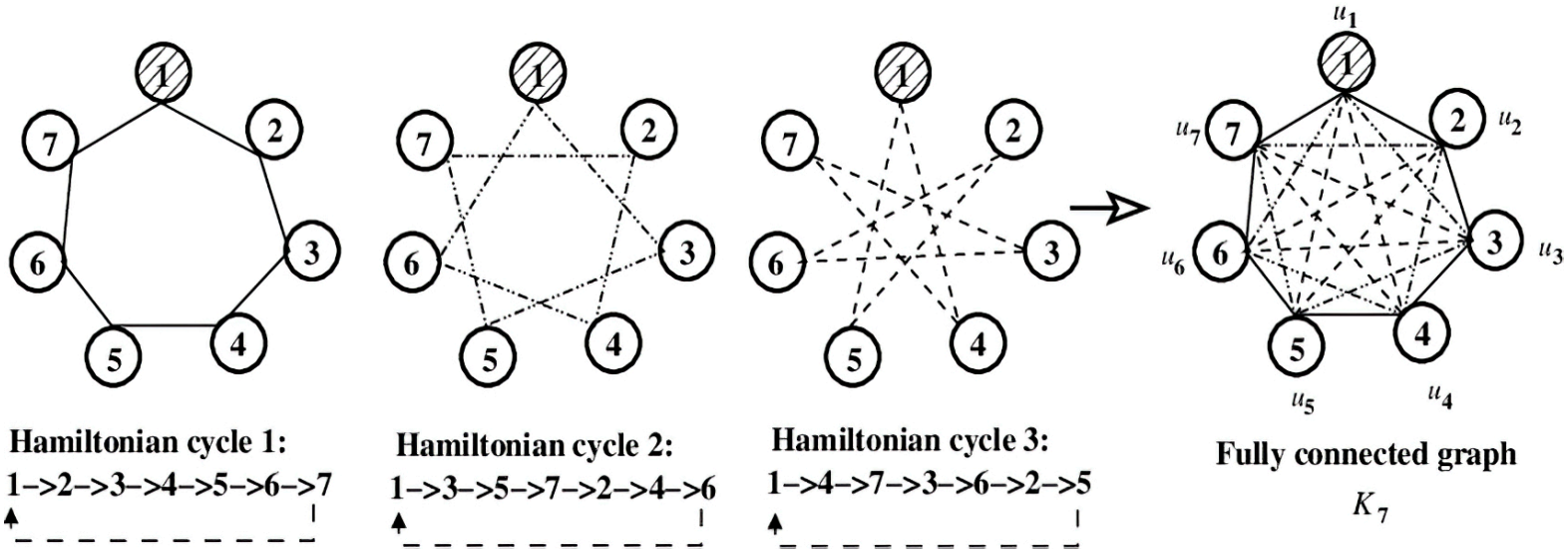


Figure: Hamiltonian Cycles

Finding Hamiltonian Cycles - What is a Hamiltonian Cycle

- What: Given a directed graph G and an initial vertex v_0, and a path from v_0 to v_0 that enters each vertex exactly once.
- Why: Applications to numerous problems including processor allocation and delivery scheduling.
- How: Construct an ASP program whose answer sets correspond to Hamiltonian cycles of graphs G.

Finding Hamiltonian Cycles - Representation

- graph: vertices and edges as we've seen before.
- cycle: collection of statements of the form

  $$in(v_0, v_1), \cdots, in(v_k, v_0)$$

  $in(V_1, V2_)$ states that "the edge from vertex V1 to vertex V_2 is in a given Hamiltonian cycle."
- If we can describe when $in(V_1, V_2)$ is true, we will have a program that computes Hamiltonian cycles.

Finding Hamiltonian Cycles - Defining the Cycle

Three conditions make a set of atoms of the from $in(V_1, V_2)$ a Hamiltonian cycle; the collection of atoms:

1. leaves each vertex at most once;
2. enters each vertex at most once;
3. enters every vertex of the graph.

Finding Hamiltonian Cycles - Defining the Cycle

Conditions 1 & 2:

```
%% Only one way out of a node:
-in(V, V2) :- in(V, V1),
              V1 != V2.

%% Only one way into a node:
-in(V2, V) :- in(V1, V),
              V1 != V2.
```

Finding Hamiltonian Cycles - Defining the Cycle

Condition 3: To define that the path must enter every vertex of the graph, we define relation *reached*(V), which holds if the path enters vertex V on its way from the initial vertex:

```
 reached(V2) :- init(V1),
                in(V1,V2).
 reached(V2) :- reached(V1),
                in(V1,V2).
-reached(V) :- not reached(V).

%% It is impossible that a vertex is not reached:
:- -reached(V).
```

Finding Hamiltonian Cycles - Generation

Now that we have defined a set of atoms that constitutes a Hamiltonian cycle, we must generate candidate paths which our rules will test:

```
in(V1,V2) | -in(V1,V2) :- edge(V1,V2).
```

states that every given edge is either in the path or is not.

Finding Hamiltonian Cycles - A New Way for Problem Solving

- We have just seen a new way to look at an old problem.
- Focus is on defining an appropriate encoding of the definition of the problem vs. data structure and algorithm.
- In this case, the declarative solution is:
  - shorter
  - easier to implement
  - more transparent
  - more reliable

  Open question: What are the limits of applicability of the declarative method?

Choice Rules - Definition

A useful extension of ASP syntax, the choice rule allows us to generate answer sets of various cardinalities, based on previously defined predicates.

A choice rule has two forms:

$$n1\ OP1\ \{p(X) : q(X)\}\ OP2\ n2 :- body.$$
$$n1\ OP1\ \{p(c1); \cdots ; p(ck)\}\ OP2\ n2 :- body.$$

where the OPs are relations $<$, $>$, $!=$, $<=$, or $>=$. We do not worry about what mixtures of these operators mean and just consider the standard $<=$ which is the operator *Clingo* assumes you mean if you omit the OPs.

Choice Rules - Definition

Details

$$n1\ OP1\ \{p(X) : q(X)\}\ OP2\ n2 :- body.$$

Here expression $\{p(X) : q(X)\}$ means that if $q(X)$ is in the answer set, then $p(X)$ has to be in the answer set too.

$$n1\ OP1\ \{p(c1); \cdots ; p(ck)\}\ OP2\ n2 :- body.$$

Expression $\{p(c1); \cdots ; p(ck)\}$ means that choice is from atoms $p(c1), \cdots , p(ck)$.

Choice Rules - Examples

Example

- Program

```
q(a).
0 {p(X):q(X)} 1 % the same as
                % 0 <= {p(X): q(X)} <= 1
```

  has two answer sets {q(a)}, {q(a),p(a)}.

- Program

```
q(b).
0 {p(a);p(b)} 1 % the same as
                % 0 <= {p(a);p(b)} <= 1
```

  has three answer sets {q(b)}, {q(b),p(a)}, {q(b),p(b)}.

Choice Rules - Generation with Choice Rules

- Recall Finding Hamiltonian Cycles program. The following choice rule is sufficient to find solutions:

  ```
  ^^I{in(V1,V2):edge(V1,V2)}.
  ```

- Note that we just look for a sequence of atoms from the obtained answer set: $in(v_0, v_1), \cdots, in(v_n, v_0)$, which runs all nodes of the graph to form a Hamiltonian cycle.

Choice Rules - Generation with Choice Rules

10 min classroom exercise

We complete the program of Finding Hamiltonian Cycles by providing the following initial facts:

```
edge(a,b). edge(b,c). edge(c,d). edge(d,e).
edge(e,a). edge(a,e). edge(d,a). edge(c,e).
init(a).
```

Run this program under *clingo*.

Solving a Puzzle - The Story

Vinny has been murdered, and Andy, Ben, and Cole are suspects. Andy says he did not do it. He says that Ben was the victim's friend but that Cole hated the victim. Ben says he was out of town the day of the murder, and besides he didn't even know the guy. Cole says he is innocent and he saw Andy and Ben with the victim just before the murder. Assuming that everyone - except possibly for the murderer - telling the truth, use ASP to solve the case.

Solving a Puzzle - Representation

- We need introduce proper predicates to represent the key relations in this puzzle:
  $says(\#person, \#statement, \#truth\_value)$
  $holds(\#statement)$
  $murderer(\#person)$
- We use functions to represent various statements:

$$\begin{aligned} \#statment = \quad \{ \quad & is\_murderer(\#person), \\ & hated(\#person, \#person), \\ & out\_of\_town(\#person), \\ & know(\#person, \#person), \\ & innocent(\#person), \\ & together(\#person, \#person) \\ \} & \end{aligned}$$

Solving a Puzzle - Representation

- The values of objects (constants):
  $\#person = \{andy, ben, cole, vinny\}$
  $\#turth\_value = \{0, 1\}$

Solving a Puzzle - Program

```
%% Andy says
says(andy,is_murderer(andy),0). %% He didn't do it.
says(andy,hated(cole,vinny),1). %% Cole hated Vinny.
says(andy,friends(ben,vinny),1). %% Ben and Vinny
                                 %% were friends.
%% Ben says:
says(ben,out_of_town(ben), 1). %% He was out of town
says(ben,know(ben, vinny), 0). %% He didn't know Vinny.

%% Cole says:
says(cole,innocent(cole),1). %% He is innocent.
says(cole,together(andy,vinny),1). %% He saw Andy
                                   %% and Ben with
says(cole,together(ben,vinny),1).  %% the victim.
```

Solving a Puzzle - Program

```
%% Everyone, except possibly for the murderer,
%% is telling the truth:
holds(S) :- says(P,S,1),
            -holds(is_murderer(P)).
-holds(S) :- says(P,S,0),
             -holds(is_murderer(P)).

%% Normally, people aren't murderers:
-holds(is_murderer(P)) :- not holds(is_murderer(P)).

%% Relation together is symmetric and transitive:
holds(together(A,B)) :- holds(together(B,A)).
holds(together(A,B)) :- holds(together(A,C)),
                        holds(together(C,B)).
```

Solving a Puzzle - Program

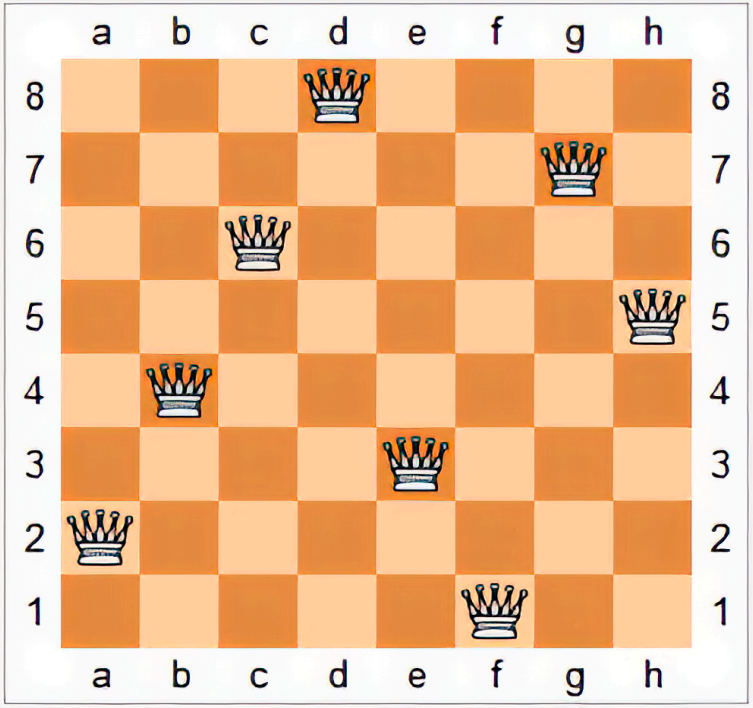
```
%% Relation friends is symmetric:
holds(friends(A,B)) :- holds(friends(B,A)).

%% Murderers are not innocent (constraint):
:- holds(innocent(P)),
   holds(is_murderer(P)).

%% A person cannot be together with
%% someone who is out of town (constraint):
:- holds(out_of_town(A)),
   holds(together(A,B)).

%% Friends know each other (constraint):
:- -holds(know(A,B)),
    holds(friends(A,B)).
```

Solving a Puzzle - Program

```
%% A person who was out of town
%% cannot be the murderer:
:- holds(is_murderer(P)),
   holds(out_of_town(P)).

%% For display:
murderer(P) :- holds(is_murderer(P)).

%% The murderer is either andy, ben or
%% cole (exclusively):
holds(is_murderer(andy)) | holds(is_murderer(ben)) |
holds(is_murderer(cole)).

%% We can replace the disjunction with
%% 1{holds(is_murderer(andy)); holds(is_murderer(ben));
%%   holds(is_murderer(cole))}1. % choice rule
```

The N Queens Problem

We have an $N \times N$ chessboard and we have to place N queens such that no two queens attack each other. That is, there is exactly one queen in each row and column and no two queens are along the same diagonal line.



One solution to the eight queens puzzle

The N Queens Problem

- Our general idea for solving N-queens problem is: we first need to *enumerate* the placing of N queens in the $N \times N$ chessboard and then *eliminate* those configurations where two queens may attack each other.
- Predicates:

  $queen(\#no)$ $\quad$ $row(\#no)$

  $at(\#queen\_no, \#row\_no, \#col\_no)$ $\quad$ $col(\#no)$

  $placed(\#queen\_no)$

The N Queens Problem

Program *nqueens.lp*

Part I: Initialization

```
%% Initializations. Note here n is a concrete
%% number, e.g., 8.
queen(1). queen(2). ... queen(n).
row(1). row(2). ... row(n).
col(1). col(2). ... col(n).
```

The N Queens Problem

Part II: Enumeration

```
%% Enumeration of all possible configurations
%% For each locations (X,Y) and each queen Q,
%% either is in location (X,Y) or not.
at(Q,X,Y) :- queen(Q),row(X),col(Y),not -at(Q,X,Y).
-at(Q,X,Y) :- queen(Q),row(X),col(Y),not at(Q,X,Y).
```

The N Queens Problem

```
%% For each queen Q, it is placed in at
%% least one location
placed(Q) :- queen(Q),row(X),col(Y ),at(Q,X,Y).
:- queen(Q),not placed(Q).

%% For each queen Q, it is placed in at
%% most one location
:- queen(Q),row(X),col(Y),row(U),col(Z),at(Q,X,Y),
   at(Q,U,Z), Y !=Z.
:- queen(Q),row(X),col(Y),row(V),col(Z),at(Q,X,Y),
   at(Q,V,Z), X !=V.
```

The N Queens Problem

```
%% No two queens are placed in the same location
:- queen(Q1),row(X),col(Y),queen(Q2),at(Q1,X,Y),
   at(Q2,X,Y), Q1 != Q2.
```

The N Queens Problem

Part III: Elimination

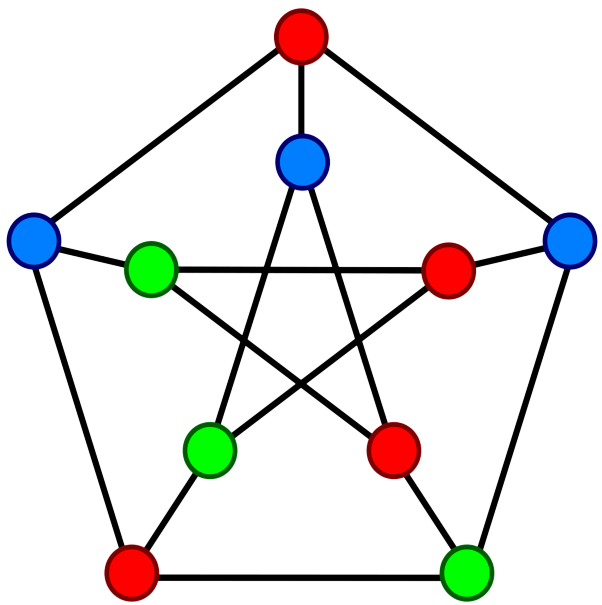
```
%% No two queens in the same row
:- queen(Q1),row(X),col(Y),at(Q1,X,Y),
   queen(Q2),row(X),col(Z),at(Q2,X,Z),
   Q1 != Q2.

%% No two queens in the same column
:- queen(Q1),row(X),col(Y),at(Q1,X,Y),
   queen(Q2),row(U),col(Y),at(Q2,U,Y),
   Q1 != Q2.
```

The N Queens Problem

```
%% No two queens in the same attach each
%% other diagonally
:- row(X),col(Y),row(U),col(V),queen(Q1),queen(Q2),
   at(Q1,X,Y),at(Q2,U,V),Q1 != Q2,
   |X-U| == |Y-V|.
```

K-Colorability

Given a positive integer K, and a graph G, we say G is K-colorable *if each vertex can be assigned one of the K colors so that no two vertices connected by an edge are assigned the same color. The decision problem is to find if a graph is K-colorable.*



K-Colorability

- Defining a graph $G = (V, E)$, where E is a set of edges and V is a set of vertices.
- Predicates:

vertex(#node)
color(#color)
another_color_of(#node,#color)
edge(#node, #node)
color_of(#node,#no)

K-Colorability

```
%% Initialization
vertex(v1). ... vertex(vx).
edge(v1,v2). ... edge(vp,vq).
color(c1). ... color(ck).

%% Assigning colors vertices
color_of(V,C) :- vertex(V),color(C),
                 not another_color_of(V,C).
another_color_of(V,C) :- vertex(V),color(C),
                         color(D),color_of(V,D),
                         C!=D.

%% Constraint
:- vertex(U),vertex(V),edge(U,V),color(C),
   color_of(U,C),color_of(V,C).
```

Tutorial and Lab Exercises

1. Write a *clingo* program encoding the following scenario:

   City C is warm if
   the temperature in C is T1,
   the temperature in Sydney is T2,
   and T1 > T2.

Suppose we have the following information:

City	Sydney	Tasmanian	Darwin	Melbourne
Temperature	28	18	35	22

Tutorial and Lab Exercises

2. If we run program *nqueens.lp* under *clingo* by setting $n = 8$, We will obtain multiple answer sets, where each answer set represents a solution of the 8-queens problem. However, by checking one answer set, we find there are too many atoms that we are not interested. In other words, most of the time, we only want to extract relevant atoms from the answer set to represent our solution.

   In *nqueens.lp* program, what predicate(s) does(do) represent a solution? Study "ASP User Guide", to add suitable statements into *nqueens.lp*, so that the answer set only contains those atoms representing the solution.

Tutorial and Lab Exercises

3. Write complete *clingo* programs for examples Hamiltonian Cycles problem, murderer puzzle, and K-colorability problem, and run your programs on *clingo*.