Table of Contents I

Reading

- Read Chapter 6, *Answer-Set Programming Paradigm*, in the KRR book.

The Answer-Set Programming Paradigm

- Let's go beyond answering queries.
- 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.

Some Popular ASP Solvers

- Clingo at `http://potassco.sourceforge.net/`
- DLV at `http://www.dlvsystem.com/`
- These two groups agreed on a standardization of ASP called ASP-Core-2. Still waiting on choice rules in DLV. Not sure where the Clingo folks are on weak constraints.
- DLV is the system that SPARC interfaces with and runs behind the scenes of the online SPARC webpage. SPARC can interface with Clingo as well, but in its current state, is not written to allow CR-rules in this mode. (Stay tuned.)
- Both systems have their advantages.

Examples: Finding Hamiltonian Cycles

- What: Given a directed graph G and an initial vertex v_0, find 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.

Representation

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

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

  $in(V_1, V_2)$ states that "the edge from vertex V_1 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.

Defining the Cycle

There are three conditions which 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.

Conditions 1 and 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.
```

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).
```

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.
See `http://pages.suddenlink.net/ykahl/s_hamgraph.txt` for program.

A New Way of Solving Problems

- You've just seen a new way to look at an old problem.
- Focuses is on finding 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: Syntactic Sugar That's Hard to Live Without

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);...;p(ck)} OP2 n2 :- body.
```

where the OPs are relations `<`, `>`, `!=`, `<=`, `or` `>=`. We won't 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: Examples

- Program

```
q(a).
0 {p(X):q(X)} 1
```

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

- Program

```
q(b).
0 {p(a);p(b)} 1
```

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

- Replacing the 1 by a 2 above gives the three answer sets plus {q(b),p(a),p(b)}.

Generation with Choice Rules

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

This rule is enough to find solutions. If we wanted to add negative information for some reason, we could add

```
-in(V1,V2) :- not in(V1,V2).
```

A Mystery Puzzle

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 — is telling the truth, use ASP to solve the case.

See `http://pages.suddenlink.net/ykahl/s_mystery.txt` for program.