Answer Set Programming

Coding the Problem rather than the Solution

Or

How to Cheat at Logic Puzzles

Intro

- Logic Programming
  - Prolog, anyone?
  - core.logic?
- Credit
  - This talk is based on one given by Marina De Vos and Willem van den Ende at a recent ACCU meeting.

Imperative programming languages

- “Traditional” languages
- "How" languages
- Step-by-step, explicit algorithm implementation
- Explicit control flow
- Programmer must design the algorithm
- Verify against a specification (somehow)

Declarative programming languages

- "What" languages
- Expresses the logic of computation not the control flow
- Define the problem and what constitutes a solution
- Language does the rest
- “The program is the specification"
- Functional is declarative?

Pros and Cons

- Advantages:
  - simple and quick to write
  - forces explicit requirements
  - no need to understand and code the algorithm

- Disadvantages:
  - can be slow - optimisation?
  - inscrutable: "42"
  - hard to learn
  - limited domain of application

Answer Set Programming (ASP)

- Derives from work in:
  - knowledge representation (KR)
  - logic programming,
  - non-monotonic logics
- Targeted towards difficult (NP-hard) search problems
- Related to:
  - boolean satisfiability (SAT)
  - constraint satisfaction problems (CSP)
- Language: AnsProlog*

ASP Paradigm

- Derived from Prolog via Datalog
- Semantics are different:
  - solutions are models not proofs
- Stable models, aka Answer Sets
- There may be 0, 1 or many answer sets

```
Problem                                              Model(s) M
--------> Encoding P --------> ASP Solver  -------->
Instance I                                           Solution(s)
```

1) encode the problem and instance as a logic program P
2) compute model(s) M of P using an ASP solver
3) extract a solution for I from M

AnsProlog* vs. Prolog

- ASP is pure declarative
  - no concessions to programmer, unlike Prolog
  - order of rules does not matter
  - order of terms in a rule not important
  - termination is not an issue
- ASP can be non-deterministic
  - can make "guesses"

ASP Architecture

- An ASP solver involves 2 stages:
  - Grounding
    - preprocess to remove the variables
    - intensive bottleneck
    - gives language expressiveness
    - e.g. Gringo, DLV, lParse
  - Model Search
    - solve for stable models (answer sets)
    - can be quite sophisticated
    - can use different algorithmic techniques (SAT,CSP,DLPP,DP)
    - e.g. Clasp, Smodels, DLV, Cmodels
- - 'Clingo' is Gringo and Clasp rolled into one

ASP Applications

- Planning
- Routing and scheduling
- Diagnosis
- Verification
- Optimisation
- Knowledge management
- Information integration
- …

Programming

- programs consist of rules of the form:
  Conclusion :- Condition
  < Head > < Body >
- rules are composed of atoms
- atoms start with lower case, variables with capitals
- head is a single atom
- body is a list of positive and negative atoms
- head on its own is a fact
- body on its own is an integrity constraint
- "negation by failure" is not classical negation
  - **not x**, is assumed true unless x is derived in the model
  - classical negation is **-x**
- choice rule **{a,b,c,…}** in the head
  - any combination of the atoms

Examples 1

a)

```
p :- q.
```

b)

```
p :- q.
q :- not r.
```

c)

```
p :- not q.
q :- not p.
```

Example 2

```
p :- q.
q :- not r.
{s,t} :- p.
:- s, not t.
```

Programming

- built-in arithmetic operators

  +, **-**, *****, **/**, \, **||**,**&**,**^**
- built-in integer comparison predicates

  <, **>**, **<=**, **>=**, **!=**, **==**
- assignments

  =, **:=**
- conditions **:** for domains
- interval/range **1..5** shorthand for brevity
- pooling **;** shorthand for brevity

Example 3

```
p(a ; b).
{q(X) : p(X)}.
```

Programming

- aggregates
  - cardinality constraint **lower{…}upper**
  - weighted sum **lower[atom1 = w1, atom2 = w2, …]upper**
  - square brackets (multiset), curly brackets (set)
- optimisation
  - **#minimize[atom1 = w1, atom2 = w2, …]**
  - **#maximize[atom1 = w1, atom2 = w2, …]**
- meta-statements
  - comments **%**
  - **#hide/#show**
  - **#const n=5**

Knapsack Problem

- http://xkcd.com/287/

Diners Problem

“Mr and Mrs Astor, Mr and Mrs Blake, Mr and Mrs Crane, and Mr and Mrs Davis were seated around a circular table. Mrs Astor was insulted by Mr Blake, who sat next to her on her left. Mr Blake was insulted by Mrs Crane, who sat opposite him across the centre of the table. Mrs Crane was insulted by the hostess, who was the only person to sit next to each one of a married couple. The hostess was insulted by the only person to sit next to each one of two men. Who insulted the hostess? Mrs. Davis is the hostess and she is seated at place 0.”

Incremental ASP

- 'iClingo' is an incremental version of Clingo
- iterates on special counter 1..
- alternately grounds and solves at each value of the counter
- uses meta-statements to partition the code
  – **#base** for static code that is processed only once
  – **#cumulative *const*** declares code which accumulates over the whole computation
  – **#volatile *const*** is for code which needs to be dismissed and recalculated on each step

Missionaries and Cannibals

- There are 3 missionaries and 3 cannibals on one side of a river.
- They all want to get across to the other side.
- There is a boat but it only carries 2 people.
- Anywhere the number of cannibals exceeds the number of missionaries, the missionaries get eaten.
- What sequence of boat journeys achieves the goal?