Answer Set Programming

- Answer Set Programs
- Stable Models Semantics
- Implementation Techniques

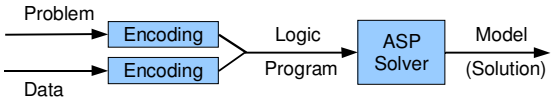
Example ASP: 3-Coloring

Problem: For a graph (*V*, *E*) find an assignment of one of 3 colors to each vertex such that no adjacent vertices share a color.

```
clrd(V,1) :- not clrd(V,2), not clrd(V,3), vtx(V).
clrd(V,2) :- not clrd(V,1), not clrd(V,3), vtx(V).
clrd(V,3) :- not clrd(V,1), not clrd(V,2), vtx(V).
:- edge(V,U), clrd(V,C), clrd(U,C).

vtx(a). vtx(b). vtx(c). edge(a,b). edge(a,c). ...
```


ASP in Practice



- Compact, easily maintainable representation
- Roots: logic programming
- Solutions = Answer sets to logic program

Some Applications

- Constraint satisfaction
- Planning, Routing
- Computer-aided verification
- Security analysis
- Configuration
- Diagnosis

ASP vs. Prolog

- Prolog not directly suitable for ASP
  - Models vs. proofs + answer substitiutions
  - Prolog not entirely declarative
- Stable model semantics: alternative semantics for negation-as-failure
- Existing ASP Systems: SMODELS, DLV and others

Stable Models: Example

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

The only stable model is {p, r}

- {p} is not a stable model (because it's not a model)
- {r, s} is not a stable model (because r included for no reason)

Stable Models: Definition

Consider a program P of ground clauses

$$A \leftarrow B_1, \ldots, B_m, \texttt{not}\ C_1, \ldots, \texttt{not}\ C_n \qquad (m \geq 0, n \geq 0)$$

Let S be a set of ground atoms.

- Reduct P^S :<=>
  - delete each clause with some not C_i s.th. $C_i \in S$
  - delete each not C_i s.th. $C_i \notin S$
- S stable model :<=> S = least-model(P^S)

Properties

- Programs can have multiple stable models.

```
p1 :- not q1.    q1 :- not p1.
 ⋮                ⋮
pn :- not qn.    qn :- not pn.
```

  This program has 2^n stable models

- Programs can have no stable models.

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

Properties (ctd)

- A stratified program has a unique stable model (the standard model).
- Checking whether a set of atoms is a stable model can be done in linear time.
- Deciding whether a program has a stable model is NP-complete.

Programs with Variables and Functions

- Semantics: Herbrand models
- Clause seen as shorthand for all its ground instances

```
clrd(V,1) :- not clrd(V,2), not clrd(V,3), vtx(V).
```

stands for

```
clrd(a,1) :- not clrd(a,2), not clrd(a,3), vtx(a).
clrd(b,1) :- not clrd(b,2), not clrd(b,3), vtx(b).
...
```

- Constraint

$\leftarrow B_1, ..., B_m,$ `not` $C_1, ...,$ `not` C_n

shorthand for ... $f \leftarrow B_1, ..., B_m,$ `not` $C_1, ...,$ `not` $C_n,$ `not` f

ASP Solver: Architecture

Two challenging tasks: handle complex data; search

Two-layer architecture

- Grounding handles complex data: A set of ground clauses is generated which preserves the models
- Model search uses special-purpose search procedures

Grounding: Domain Restrictions

- Domain-restricted programs guarantee decidability.
- Domain-restricted programs consist of two parts:
  1. Domain predicate definitions (stratified clause set), where each variable occurs in a positive domain predicate defined in an earlier stratum;
  2. Clauses where each variable occurs in a positive domain predicate in the body.
- The domain predicate definitions have a unique stable model, which is subset of every solution to the program.
- Only those ground instances of clauses need to be generated where the domain predicates in the body are true.

Example: Domain Predicate Definitions

```
col(1). col(2). col(3).
r(a,b). r(a,c). ...
d(U) :- r(V,U).
tr(V,U) :- r(V,U).
tr(V,U) :- r(V,Z), tr(Z,U), d(U).
edge(t(V), t(U)) :- tr(V,U), not tr(U,U), not tr(V,V).
vtx(V) :- edge(V,U).
vtx(U) :- edge(V,U).
```


Example: Domain-Restricted Clauses

```
clrd(V,1) :- not clrd(V,2), not clrd(V,3), vtx(V).
clrd(V,2) :- not clrd(V,1), not clrd(V,3), vtx(V).
clrd(V,3) :- not clrd(V,1), not clrd(V,2), vtx(V).
:- edge(V,U), col(C), clrd(V,C), clrd(U,C).
```


Example: Grounding

Suppose that the unique stable model for the definition of the domain predicate `vtx(V)` contains $\texttt{vtx}(v_1)$, ..., $\texttt{vtx}(v_n)$

Then for the clause

```
clrd(V,1) :- not clrd(V,2), not clrd(V,3), vtx(V).
```

grounding produces

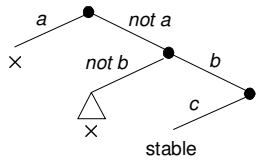
$\texttt{clrd}(v_1,1)$:- not $\texttt{clrd}(v_1,2)$, not $\texttt{clrd}(v_1,3)$.

...

$\texttt{clrd}(v_n,1)$:- not $\texttt{clrd}(v_n,2)$, not $\texttt{clrd}(v_n,3)$.

Search

- Backtracking over truth-values for atoms



- Each node consists of a model candidate (set of literals)
- Propagation rules are applied after each choice

Propagation Rules

- A propagation rule extends a model candidate by one or more new literals.
- Propagation rules need to be correct: If L is derived from model candidate A then L holds in every stable model compatible with A.

Propagation Rule "Upper Bound"

Consider program P and candidate model A

Let P' be all clauses in P

- whose body is not false under A
- without negative body literals

If $p \notin$ least-model (P') derive not p

P: `p2 :- p1, not q1.` `p1 :- p2, not q1.` `p2 :- not q2.`

A: {`q2`}

P': `p2 :- p1.` `p1 :- p2.`

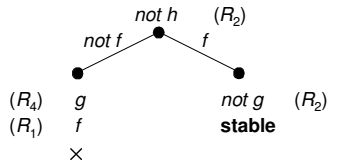
Derive: `not p1, not p2, not q1, not q2`

Local Propagation Rules

	Only clauses for q	A	Derive
(R_1)	$q \leftarrow p_1$, *not* p_2	p_1, *not* p_2	q
(R_2)	$q \leftarrow p_1$, *not* p_2 $q \leftarrow p_3$, *not* p_4	p_2, *not* p_3	*not* q
(R_3)	$q \leftarrow p_1$, *not* p_2	q	p_1, *not* p_2
(R_4)	$q \leftarrow p_1$, *not* p_2	*not* q, p_1	p_2

Example

```
f :- not g, not h
g :- not f, not h
f :- g
```



Lookahead

Given a program P and a candidate model A.

If, for a literal L, **propagate**(P, $A \cup \{L\}$) contains a conflict (some p together with *not* p) derive the complement of L.

Search Heuristics

Heuristics to select the next atom for splitting the search tree:

- an atom with the maximal number of occurrences in clauses of minimal size
- an atom with the maximal number of propagations after the split
- an atom with the smallest remaining search space after split + propagation