Answer Set Programming without Unstratified Negation

Ilkka Niemelä

Helsinki University of Technology
Department of Information and Computer Science
Ilkka.Niemela@tkk.fi

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Outline

- Unstratified negation and the stable model semantics
- The suggestion: use choice constructs instead of unstratified negation
- Semantics
- Capturing unstratified negation
- Extensions
- Conclusions

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Introduction

- The stable model semantics was originally introduced to provide a declarative account of negation as failure in Prolog.
- In particular, to understand recursion through negation, i.e., unstratified negation.

Example. Consider the two programs with unstratified negation:

P_1:

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

P_2:

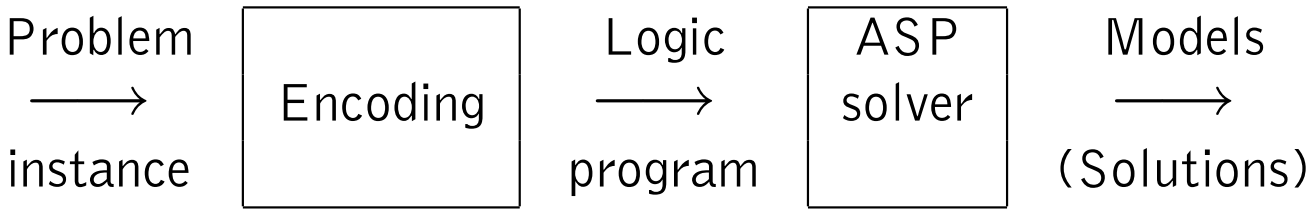
```
p :- not p.
q :- p.
```

- A Prolog system just loops forever when querying `p` for both programs.
- The stable model semantics gives an intuitive declarative semantics for the programs:
  - Program P_1 has two stable models $\{p\}$ and $\{q\}$.
  - Program P_2 no stable models.

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Introduction—cont'd

- In the mid 90s efficient implementations capable of computing stable models for large (ground) programs were emerging and this led to Answer Set Programming (ASP):



- For ASP with normal programs unstratified negation is essential: a stratified program has a unique stable model!

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Introduction—cont'd

- In ASP multiple alternative solution candidates are encoded using unstratified negation.
- However, understanding and developing such an encoding is quite challenging (often non-trivial recursive definitions).
- Example. Hamiltonian circuit

```
hc(V1,V2) :- edge(V1,V2), not otherroute(V1,V2).
otherroute(V1,V2) :- hc(V1,V3), V2 != V3.
otherroute(V1,V2) :- hc(V3,V2), V1 != V3.
r(V2) :- hc(V1,V2), r(V1), not initialnode(V1).
r(V2) :- hc(V1,V2), initialnode(V1).
:- vertex(V), not r(V).
```

- This paper: Reconsider the role of unstratified negation

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

SCI Programs

- For ASP purposes: what would be an alternative class of basic programs (instead of normal programs) having simple semantics, allowing intuitive encodings, straightforward to extend, ...?
- Alternatives: choice constructs, cardinality and other constraint atoms, disjunctions, ...
- The suggestion: choice constructs instead of unstratified negation

  ☞ **SCI** programs with **S**tratified negation, **C**hoice constructs, and **I**ntegrity constraints allowing rules of the form:

$$a \;\texttt{:-}\; b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n. \quad \text{(Normal rules)}$$

$$\{a_1, \ldots, a_l\} \;\texttt{:-}\; b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n. \quad \text{(Choice rules)}$$

$$\texttt{:-}\; b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n. \quad \text{(Integrity constraints)}$$

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

SCI Programs

- **Only stratified negation allowed:**
  predicate symbols in the program can be mapped to strata (natural numbers) such that the negation of a predicate p can be used in the body of a rule only if its stratum is lower than that of the head predicate.
- **Example.**

```
edge(1,2).  edge(1,3).  ...                    (S(edge) = 1)
r(X,Y) :- edge(X,Y).
r(X,Y) :- edge(X,Z), r(Z,Y).                    (S(r) = 1)
{asym(X)} :- r(X,Y), not r(Y,X).               (S(asym) = 2)
:- edge(1,X), asym(X).
```

SCI Programs

- SCI programs are a special case of cardinality constraint programs or programs with monotone constraints atoms.
- A program without choice and integrity rules is a normal stratified program P with a unique canonical model $\mathrm{StM}(P)$.
- $\mathrm{StM}(P)$ can be constructed iteratively bottom up layer by layer in the stratification.
- Example.

  Consider program P:

  ```
  a.
  c :- a, not b.
  d :- not c, a.
  e :- c, not d.
  ```

  The canonical model $\mathrm{StM}(P)$ contains: $\{a, c, e\}$.

- $\mathrm{StM}(P)$ coincides with the standard stable model, the well-founded model, ...

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Semantics

- For a **SCI** program a stable model S should
  - satisfy the integrity constraints and
  - be justified: contain exactly those atoms that can be justified by the rules given S.
- Given a **SCI** program P and a model candidate S, the reduct gives the rules that can be used for justifying atoms.
- Given a set S of atoms, the **SCI-reduct** P^S of P w.r.t. S is the set of rules including
  - all normal rules in P and
  - for each choice rule
    $\{a_1, \ldots, a_l\} :- b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n.$
    for every head atom $a_i \in S$ a rule
    $a_i :- b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n.$
- P^S is a normal stratified program without integrity constraints.
- The atoms justified by P^S are given by $\mathrm{StM}(P^S)$.

SCI-Stable Models

Definition. A set of atoms S is a stable model of a SCI program P iff (i) $S = \mathrm{StM}(P^S)$ and
(ii) S satisfies the integrity constraints in P.

Example.

Consider the program P:

```
c :- a.
b :- not a.
{a} :- not d.
:- not a.
```

For $S_1 = \{a, c\}$ the reduct P^{S_1}:

```
c :- a.
b :- not a.
a :- not d.
```

$\mathrm{StM}(P^{S_1}) = \{a, c\}$.
S_1 satisfies `:- not a.`
So S_1 is a stable model.

For $S_2 = \{b\}$ the reduct P^{S_2}:

```
c :- a.
b :- not a.
```

$\mathrm{StM}(P^{S_2}) = \{b\}$.
S_2 does not satisfy `:- not a.`
Hence, S_2 is not a stable model.

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Modelling

SCI programs allow a natural generate + define + test programming paradigm for uniform encodings

Example. Hamiltonian circuit.

```
{hc(V,U)} :- edge(V,U).
:- hc(V,U), hc(V,W), U != W.
:- hc(U,V), hc(W,V), U != W.
r(V) :- hc(S,V), start(S).
r(V) :- r(U), hc(U,V).
:- vtx(V), not r(V).
```

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Capturing Unstratified Negation

- Unstratified negation can be captured using **SCI** programs.
- A negative literal not p can be seen as a new atom out_p for which
  - a choice needs to be made such that
  - a model cannot contain both p and out_p but
  - one of them needs to be included.

Example. The rules:

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

translate to:

```
p :- q, out_p.
q.
{out_p}.
:- p, out_p.
:- not p, not out_p.
```

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Extensions

- Extending the language with "stratified constraints" is straightforward (predicates used in a constraint are defined on a lower stratum like for stratified negation).
- Constraints in the head can be represented using a choice rule and integrity constraints

  Example.

  ```
  odd(a1,...,al) :- b, not c.
  ```

  can be encoded with two rules

  ```
  {a1,...,al} :-b, not c.
  :- not odd(a1,...,al), b, not c.
  ```

- Monotone recursive constraints can be supported directly.

Conclusions

- For ASP purposes it is not necessary to use unstratified negation
- An alternative: a basic language based on simple choice constructs, integrity constraints, and stratified negation.
- Natural problem encodings
- Smooth extension, for instance, with constraints and aggregates.

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science