Developing an Inference Engine for ASet-Prolog

Presented By

by

Veena S. Mellarkod

January 24, 2003

Goal

The goal of this talk is to introduce a language called ASet-Prolog and present an algorithm for computing the answer-sets for programs written in this language.

Structure

The talk is structured as follows:

- language
- algorithm
- future work

ASet-Prolog

- ASet-Prolog is an extension of A-Prolog by sets and functions from sets to natural numbers (aggregates).

- There are 3 kinds of atoms in the language: $r - atoms$, $s - atoms$ and $f - atoms$.

Atoms

r-atoms: regular atoms of A-Prolog.

s-atoms: expressions of the form

$$\{\overline{X} : p(\overline{X})\} \subseteq \{\overline{X} : q(\overline{X})\}$$

f-atoms: expressions of the form

$$t = f(\{\overline{X} : p(\overline{X})\})$$

f can be card, sum, max, min etc.,

Programs

An ASet-Prolog program is a collection of rules of the form:

$$l_0 :- \ l_1, \ldots, l_m, \textit{not } l_{m+1}, \ldots, \textit{not } l_n.$$

where $l_1, \ldots, l_n$ are arbitrary atoms and l_0 is either an $r - atom$ or $s - atom$ in ASet-Prolog.

Example:

$$\Pi_0 \begin{cases} p(a). \quad p(b). \\ \{X : q(X)\} \subseteq \{X : p(X)\}. \end{cases}$$

Answer Sets:
{ p(a), p(b) }
{ p(a), p(b), q(a) }
{ p(a), p(b), q(b) }
{ p(a), p(b), q(a), q(b) }

Examples

$$\Pi_1 \begin{cases} p(a). \quad p(b). \\ \{X : q(X)\} \subseteq \{X : p(X)\}. \\ \ :- \ T = card(\{X : q(X)\}), T \leq 1. \end{cases}$$

Answer Set: { p(a), p(b), q(a), q(b) }

$$\Pi_2 \begin{cases} p(a). \quad p(b). \quad r(a). \\ s(a) \ :- \ \{X : r(X)\} \subseteq \{X : p(X)\}. \end{cases}$$

Answer Set: { p(a), p(b), r(a), s(a) }

Semantics

Let S be a set of ground r-atoms in Σ_Π.

1. An $r - atom\ l$, in Σ_Π is true in S if $l \in S$

2. An $s - atom$ is true in S if for any sequence $\overline{x}$ of ground terms of Σ_Π, either

   - $p(\overline{x}) \notin S$ or

   - $q(\overline{x}) \in S$.

3. An $f - atom$ is true in S if value of the aggregate of the set $\{\overline{x} : p(\overline{x}) \in S\}$ is equal to t.

Stable Models of ASet

Let S be a collection of ground $r - atoms$. $se(\Pi, S)$ is the program obtained by:

1. removing from Π all the rules whose bodies contain $s - atoms$ or $f - atoms$ not satisfied by S;

2. removing all remaining $s - atoms$ and $f - atoms$ from the bodies of the rules;

3. replacing rules of the form $l \leftarrow \Gamma$ where l is an $s - atom$ not satisfied by S by rules $\leftarrow \Gamma$;

4. replacing the remaining rules of the form: $\{\overline{x} : p(\overline{x})\} \subseteq \{\overline{x} : q(\overline{x}\} \leftarrow \Gamma$ by the rules $p(\overline{t}) \leftarrow \Gamma$ for each $p(\overline{t})$ from S.

S is a stable model of Π if it is a stable model of $se(\Pi, S)$.

Example

$$\Pi_3 \begin{cases} p(1).\ p(2).\ r(1). \\ q(a) :- \{X : p(X)\} \subseteq \{X : r(X)\}. \\ q(b) :- \{X : r(X)\} \subseteq \{X : p(X)\}. \\ q(c) :- T = card(\{X : r(X)\}), T > 1. \\ q(d) :- T = sum(\{X : r(X)\}), T = 1. \\ \{X : r(X)\} \subseteq \{X : p(X)\}. \end{cases}$$

Example Contd.

Let $S = \{p(1), p(2), r(1), q(b), q(d)\}$ then $se(\Pi_0, S)$ is:

p(1). p(2). r(1).
q(b) $:-$
q(d) $:-$
r(1) $:-$

S is a stable model of $se(\Pi_0, S)$.

Example Contd.

Let $S_1 = \{p(1), p(2), r(1), r(2), q(a), q(b), q(c)\}$ then $se(\Pi_0, S_1)$ is:

p(1). p(2). r(1).
q(a) :−
q(b) :−
q(c) :−
r(1) :−
r(2) :−

S_1 is a stable model of $se(\Pi_0, S_1)$.

Example Contd.

Let $S_2 = \{p(1), p(2), r(1), q(a), q(c)\}$ then $se(\Pi_0, S_2)$ is:

p(1). p(2). r(1).
q(b) $:-$
q(d) $:-$
r(1) $:-$

S_2 is not a stable model of $se(\Pi_0, S_2)$..

Coloring of Graphs

Given a graph G defined by a set of facts of the form node(X) and edge(X,Y) and a set C of colors color(red), color(green) etc., The coloring problem can be represented by a program Π:

$$\{C : colored(X, C)\} \subseteq \{C : color(C)\} :- node(X).$$

$$:- N = card(\{C : colored(X, C)\}), N \neq 1.$$

$$:- colored(X, C), colored(Y, C), edge(X, Y).$$

Course Pre-requisites

Given a record of courses passed by a student s, as facts: passed(s,c1), passed(s, c2) and passed(s, c3) and a list of pre-requisities for each class as facts: prereq(c1,c4), pre-req(c2,c4), and pre-req(c4,c5), the rule that a student S is allowed to take class C if he passed all the pre-requisities for C and didn't pass C yet, can be written as:

$can\_take(S, C) :-$
$\{X : prereq(X, C)\} \subseteq \{X : passed(S, X)\},$
not $passed(S, C)$.

Sets and Choice Rules

An Smodels Program:

$$
\begin{array}{l}
d_1(1). \\
d_2(1). \quad d_2(2). \\
\{p(X) : d_1(X)\}. \\
\{p(X) : d_2(X)\}.
\end{array}
$$

Stable Models of the Program:

$$
\begin{array}{l}
\{\ \} \\
\{p(1)\} \\
\{p(2)\} \\
\{p(1), p(2)\}
\end{array}
$$

Sets and Choice Rules

An ASET-Prolog Program:

$$d_1(1).$$
$$d_2(1). \quad d_2(2).$$
$$\{X : p(X)\} \subseteq \{X : d_1(X)\}.$$
$$\{X : p(X)\} \subseteq \{X : d_2(X)\}.$$

Answer sets of the Program:

$$\{\ \}$$
$$\{p(1)\}$$

Algorithm

```
function aset(Π, S) : boolean
    S := expand(Π, S)
    if conflict(S) then
        return false
    else if covers(S, Atoms(Π)) then
        return true
    else
        pick(l, S̄)
        if aset(Π, S ∪ {l}) then
            return true
        else
            return aset(Π, S ∪ {not l})
        end if
    end if
```

Expand

Input:

1. the program Π
2. set of literals S

Output:

$$\mathsf{S} = cl(\Pi, S)$$

Expand

function $expand(\Pi, S) : cl(\Pi, S)$

$S_0 := S;\ \Pi_0 = \Pi$

repeat

$S' := S$

$S := lc(\Pi, S)$

$S := \{not\ x \mid x \notin uc(\Pi, S)\}$

until $S = S'$ **or** $conflict(S)$

if $conflict(S)$ **then**

$S := S_0;\ \Pi := \Pi_0$

return S

$lc(\Pi, S)$

A set U of atoms is called a deductive closure of S w.r.t. Π, if it is a minimal set containing S and closed under the seven inference rules.

1. If all literals in the body of a rule $h :- l_1, \ldots, l_n$ are true in U, then $h \in U$.

Example:Let $S = \{a\}$

$$\Pi \begin{cases} a. \ b. \\ q :- a \\ p :- b, notc \\ r :- p, q \end{cases}$$

$U = \{a, b, q, p, r\}$

Inference Rule 1

Example:Let $S = \{\}$

$$\Pi \begin{cases} p(1).\ p(2).\ r(1). \\ q(1) :- \{X : r(X)\} \subseteq \{X : p(X)\}. \\ q(2) :- \{X : p(X)\} \subseteq \{X : r(X)\}. \\ t :- T = card(\{X : q(X)\}), T = 1. \end{cases}$$

$U = \{\ p(1),\ p(2),\ r(1),\ q(1),\ t\ \}$

Inference Rule 2

If an r-atom l, is neither the head of any rule in Π, nor is s-defined in Π, then *not* $l \in U$.

Example:Let $S = \{\}$

$$\Pi \begin{cases} p(1).\ p(2).\ r(1). \\ q(1) :- \{X : r(X)\} \subseteq \{X : p(X)\}. \\ q(2) :- \{X : p(X)\} \subseteq \{X : r(X)\}. \\ t :- T = card(\{X : q(X)\}), T = 1. \end{cases}$$

$U = \{\ p(1),\ p(2),\ r(1),\ q(1),\ \textit{not}\ q(2)\ \}$

Inference Rule 3

If h is an r-atom that is the head of only one rule, $r = h :- l_1, \ldots, l_n$ in Π, $h \in U$, and h is not s-defined in Π, then $l_1, \ldots, l_n \in U$.

Example:Let $S = \{q(1)\}$

$$\Pi \begin{cases} p(1).\ p(2).\ r(1). \\ q(1) :- \{X : r(X)\} \subseteq \{X : p(X)\}, t(2). \\ t(1) :- t(2). \end{cases}$$

$U = \{\ p(1),\ p(2),\ r(1),\ q(1),\ s_1,\ t(2),\ t(1)\ \}$

Inference Rule 4

If h is the head of a rule $r = h :- l_1, \ldots, l_n$ in Π, *not* $h \in U$, all literals in the body except l_i belong to U, then *not* $l_i \in U$.

Example:Let $S = \{p(1),\ r(1),\ \textit{not}\ q(1),\ s_1\}$

$$\Pi \begin{cases} p(1).\ r(1). \\ q(1) :- \{X : r(X)\} \subseteq \{X : p(X)\}, t(2). \\ t(1) :- t(2). \end{cases}$$

$U = \{\ p(1),\ r(1),\ \textit{not}\ q(1),$

$$s_1,\ \textit{not}\ t(2),\ \textit{not}\ t(1)\ \}$$

Inference Rule 5

If $s = \{X : p(X)\} \subseteq \{X : q(X)\}$ is an s-atom then

(a). If $s \notin U$, and for any $t \neq t_i$, *not* $p(t) \in U$ or $q(t) \in U$ then $p(t_i) \in U$ and $q(t_i) \notin U$.

Example:
Let $S = \{p(1),\ r(1),\ \textit{not}\ s_1\}$

$$\Pi \begin{cases} p(1).\ r(1). \\ p(2) :- \textit{not}\ r(2). \\ r(2) :- \textit{not}\ p(2). \\ q(1) :- \{X : r(X)\} \subseteq \{X : p(X)\}. \end{cases}$$

$$U = \{\ p(1),\ r(1),\ \textit{not}\ s_1,\ \textit{not}\ q(1),$$

$$r(2),\ \textit{not}\ p(2)\ \}$$

Inference Rule 5

If $s = \{X : p(X)\} \subseteq \{X : q(X)\}$ is an s-atom then

(b). If $s \in U$, then for every t,
if $p(t) \in U$ then $q(t) \in U$ or
if *not* $q(t) \in U$ then *not* $p(t) \in U$.

Example:
Let $S = \{p(1),\ r(1),\ r(2),\ s_1\ \}$

$$\Pi \begin{cases} p(1).\ r(1). \\ p(2) :- \textit{not}\ r(2). \\ r(2) :- \textit{not}\ p(2). \\ q(1) :- \{X : r(X)\} \subseteq \{X : p(X)\}. \end{cases}$$

$U = \{\ p(1),\ r(1),\ r(2),\ s_1,\ p(2),\ q(1)\ \}$

Inference Rule 6

If $s = \{X : p(X)\} \subseteq \{X : q(X)\}$ is an s-atom and for every t, *not* $p(t) \in U$ or $q(t) \in U$ then $s \in U$.

Example:
Let $S = \{p(1),\ r(1),\ p(2),\ \textit{not}\ r(2)\ \}$

$$\Pi \begin{cases} p(1).\ r(1). \\ p(2) :- \textit{not}\ r(2). \\ r(2) :- \textit{not}\ p(2). \\ q(1) :- \{X : r(X)\} \subseteq \{X : p(X)\}. \end{cases}$$

$U = \{\ p(1),\ r(1),\ p(2),\ \textit{not}\ r(2),\ s_1,\ q(1)\ \}$

Inference Rule 7

If $f_1 = f(\{t : p(t)\})$ is an f-atom, and for every t either $p(t) \in U$ or *not* $p(t) \in U$, and the value of the function f for $\{t : p(t) \in U\}$ is k, then if the expression is satisfied then $f_1 \in U$ else *not* $f_1 \in U$.

Example:
Let $S = \{p(1),\ r(1),\ p(2),\ \textit{not}\ r(2)\ \}$

$$\Pi \begin{cases} p(1).\ r(1). \\ p(2) :- \textit{not}\ r(2). \\ r(2) :- \textit{not}\ p(2). \\ q(1) :- T = sum(\{X : r(X)\}), T = 1. \\ q(2) :- T = card(\{X : p(X)\}), T \leq 1. \end{cases}$$

$U = \{\ p(1),\ r(1),\ p(2),\ \textit{not}\ r(2),$

$$f_1,\ \textit{not}\ f_2,\ q(1),\ \}$$

Computing uc(Π,S)

1. Remove all rules not satisfied by S.

2. Remove all not-atoms from body(r).

3. Compute the answer set of the resulting basic program.

Example: Let $S = \{\ p(1),\ r(1)\ \}$

$$\Pi \begin{cases} p(1).\ r(1). \\ p(2) :- \textit{not}\ r(2). \\ r(2) :- \textit{not}\ p(2). \\ q(1) :- p(1), \textit{not}\ p(2). \\ q(2) :- r(1), \textit{not}\ r(2). \end{cases}$$

$$uc(\Pi, S) = \{\ p(1),\ r(1),\ p(2),\ r(2),\ q(1),\ q(2)\ \}$$

Computing uc(Π,S)

1. Remove all rules not satisfied by S.

2. Remove all not-atoms, s-atoms and f-atoms from body(r).

3. Compute the answer set of the resulting basic program.

Example: Let $S = \{\ p(1),\ r(1)\ \}$

$$\Pi \begin{cases} p(1).\ r(1). \\ p(2) :- \textit{not}\ r(2). \\ r(2) :- \textit{not}\ p(2). \\ q(1) :- \{X : p(X)\} \subseteq \{X : r(X)\}. \\ q(2) :- \{X : r(X)\} \subseteq \{X : p(X)\}. \end{cases}$$

$$uc(\Pi, S) = \{\ p(1),\ r(1),\ p(2),\ r(2),\ q(1),\ q(2)\ \}$$

Can we do better?

Example: Let $S = \{\ p(1),\ r(1)\ \}$

$$\Pi \begin{cases} p(1).\ r(1). \\ q(1) :- r(1), b. \\ a :- \{X : p(X)\} \subseteq \{X : q(X)\}. \\ b :- c. \\ c :- b. \\ d :- a. \end{cases}$$

$uc(\Pi, S) = \{\ p(1),\ r(1),\ a,\ d\ \}$

Preferred Upper Closure:

$uc(\Pi, S) = \{\ p(1),\ r(1)\ \}$

Improved uc(Π,S)

1. Remove all rules not satisfied by S.

2. Remove all not-atoms, not-s-atoms and not-f-atoms from body(r).

3. Compute the closure for the resulting basic program.

Example: Let $S = \{\ p(1),\ r(1)\ \}$

$$\Pi \begin{cases} p(1).\ r(1). \\ q(1) :- r(1), b. \\ a :- \{X : p(X)\} \subseteq \{X : q(X)\}. \\ b :- c. \\ c :- b. \\ d :- a. \end{cases}$$

$uc(\Pi, S) = \{\ p(1),\ r(1)\ \}$

conflict(S)

S is a set of arbitrary atoms. Conflict(S) returns $true$ if: there $\exists\ a$, $a \in S$ and *not* $a \in S$.

Example: Let $S = \{\ d_1(1),\ d_2(2),\ a,\ s_1,$
$p(2),\ \textit{not}\ d_1(2), \textit{not}\ s_1\}$

$$\Pi \begin{cases} d_1(1).\ d_2(2). \\ a. \\ \{X : p(X)\} \subseteq \{X : d_1(X)\} :-\ a. \\ p(X) :-\ d_2(X). \end{cases}$$

Conflict(S) $returns$ TRUE

Difference between Sets and Choice Rules

Example:

$$\Pi \begin{cases} d_1(1).\ d_2(2). \\ a. \\ \{X : p(X)\} \subseteq \{X : d_1(X)\} :- a. \\ p(X) :- d_2(X). \end{cases}$$

ASet Returns: No Answer Set.

Smodels Returns:

$$\{\ p(2),\ d2(2),\ d1(1),\ a\ \}$$
$$\{\ p(2),\ p(1),\ d2(2),\ d1(1),\ a\ \}$$

Pick Function(Π,B) : r-atom

B is the set of r-atoms undecided in Atoms(Π).

Pick Function selects a r-atom from:

1. $B \cap X$, where X = set of r-atoms that occur as not-atoms in body(Π). OR

2. $B \cap Y$, where Y = set of r-atoms that are s-defined in Π.

Theorem

Smodels Theorem: Let Π be a program and let B be the set of atoms that appear as not-atoms in some cycles or that appear in heads of some choice rules of Π. Let A be a set of literals such that Atoms(A) = B. Then, expand(Π,A) covers Atoms(Π).

ASET Theorem: Let Π be a program and let B be the set of atoms that appear as not-atoms in Π or are s-defined in Π. Let A be a set of literals such that Atoms(A) = B. Then, expand(Π,A) covers Atoms(Π).

Can We Prove a theorem like Smodels ?

Defining Max and Min

max({ X : p(X) },S) = t
t = max($x_1, x_2, \ldots, x_n$) where $p(x_i) \in S$

(min is similarly defined.)

Example - Computing Maximum Score:
Given $student(s_1)$, $student(s_2)$, ... and
$scored(s_1, 97)$, $scored(s_2, 56)$, ...

score(M) :- student(S), scored(S,M).
maxscore(M) $:-$ M = max({ X : score(X) }).

Defining nth_max and nth_min

nth_max(n,{ X : p(X) },S) = t
where t = nth maximum in $\{x_i : p(x_i) \in S\}$

(nth_min is similarly defined)

Example: Finding the second Winner:
(Using Previous Example)

$$score(M) :- student(S), scored(S, M).$$
$$second(S) :- scored(S, M), \quad M = nth\_max(2, \{X : score(X)\}).$$

Defining function_i

function_i(i, $\{\overline{X} : p(\overline{X})\}$,S) = t
where t = function($\{ X_i : p(\overline{X}) \in S \}$).

Example: (Using Previous Example)

total(S,T) :− student(S),
T = sum_i(2, {X,C : scored(S,X,C)}).

Improving Implementation

- $range(f, 1..n)$: For each aggregate f, there is a range given in the program where the value of the aggregate lies. (For grounding by lparse)

Example:

$$\Pi \begin{cases} range(card, 0..5). \\ range(sum, 0..4). \\ p(1).\ r(1). \\ p(2) :- \textit{not}\ r(2). \\ r(2) :- \textit{not}\ p(2). \\ q(1) :- T = sum(\{X : r(X)\}), T > 3. \\ q(2) :- T = card(\{X : p(X)\}), T \leq 2. \end{cases}$$

Grounding of Aggregates

$$\Pi \begin{cases} range(card, 0..5). \\ range(sum, 0..4). \\ p(1).\ r(1). \\ p(2) :- \ not\ r(2). \\ r(2) :- \ not\ p(2). \\ q(1) :- \ sum(\{X : r(X)\}) = 4. \\ q(1) :- \ sum(\{X : r(X)\}) = 5. \\ q(2) :- \ card(\{X : p(X)\}) = 0. \\ q(2) :- \ card(\{X : p(X)\}) = 1. \\ q(2) :- \ card(\{X : p(X)\}) = 2. \end{cases}$$

Do we see a problem here ?

Too Many Ground Rules

1. For each f-atom in the program, there are n rules in the ground program, where n depends on the range of the aggregate. The number of rules increases with the increase in the range of the aggregate.

Solution:
Let us not ground the value of f-atom. Grounding of the program now is:

$$\Pi \begin{cases} range(card, 0..5). \\ range(sum, 0..4). \\ p(1).\ r(1). \\ p(2) :- \textit{not}\ r(2). \\ r(2) :- \textit{not}\ p(2). \\ q(1) :- sum(\{X : r(X)\}) > 3. \\ q(2) :- card(\{X : p(X)\}) \leq 2. \end{cases}$$

Efficient Implementation similar to weight and cardinality rules!!

Range depends on the aggregate alone

2. range(card,0..100), implies that range for the cardinality function is between 0 and 100.

$$\Pi \begin{cases} range(card, 0..100). \\ total\_students(T) :- T = card(\{X : s(X)\}). \\ total\_courses(T) :- T = card(\{X : c(X)\}). \end{cases}$$

The range should depend on the set also. This will decrease a number of unwanted ground rules.

Conclusion

A lot of interesting things to be explored!