An Introduction to Answer Sets

Gerhard Brewka

brewka@informatik.uni-leipzig.de

Universität Leipzig

Outline

1. Why are answer sets interesting?
2. How are they defined

   for definite programs?

   for normal programs?

   for extended programs?
3. How can they be used for problem solving?

Why are they interesting?

- provide meaning to logic programs with default negation not
- support problem solving paradigm where models (not theorems) represent solutions
- interesting implementations: dlv, smodels

How to define a semantics

normally:

- models = truth assignment to atoms
- represent what is possible
- can be identified with the set of true atoms

here:

- answer sets, that is sets of literals
- represent acceptable sets of beliefs
- sets of atoms not sufficient

Definite programs

Syntax of rules:

$$A \leftarrow B_1, \ldots, B_n$$

where A and the B_i are ground atoms.

A is called head, $B_1, \ldots, B_n$ body of the rule.
$\leftarrow$ can be omitted if n = 0 (fact).

Answer sets of definite programs

Let S be a set of atoms, P a definite program.

- S is closed under P iff $A \in S$ whenever $A \leftarrow B_1, \ldots, B_n \in P$ and $B_1, \ldots, B_n \in S$.
- S is grounded in P iff $A \in S$ implies there is a derivation for A from P.

Answer set: unique set of atoms closed and grounded in P, denoted $Cn(P)$.

Reminder

Derivation of A from P:
sequence $(r_1, ..., r_n)$ of rules in P such that

- A head of r_n and
- each atom appearing in body of a rule is head of a rule earlier in the sequence.

Remark

$Cn(P)$ is equivalent to

- the minimal set closed under P and
- the minimal model of P, where $\leftarrow$ is read as implication, "," as logical and.

Normal logic programs

Syntax of rules:

$$A \leftarrow B_1, \ldots, B_n, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_m$$

where A, the B_i and the C_j are ground atoms.

Note: $\mathrm{not}\, C$ reads: C is not believed!

Answer sets of normal programs

Let S be a set of atoms, P a normal program.

- S closed under P iff $A \in S$ whenever $A \leftarrow B_1, \ldots, B_n, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_m \in P$, $B_1, \ldots, B_n \in S$ and $C_1, \ldots, C_m \notin S$.
- S is grounded in P iff $A \in S$ implies there is a derivation for A from P valid in S

Answer sets: sets of atoms closed and grounded in P (also called stable models).

Valid derivations

- S defeats $A \leftarrow B_1, \ldots, B_n, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_m$ iff $C_j \in S, j \in \{1, \ldots, m\}$.
- derivation valid in S iff it is based on rules undefeated by S (disregarding not-literals)

Extended logic programs

Syntax of rules:

$$A \leftarrow B_1, \ldots, B_n, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_m$$

where A, the B_i and the C_j are ground literals.

2 types of negation:

- classical negation $\neg$
- default negation not

Answer sets of extended programs

S set of literals, P extended program.

- S closed under P iff $A \in S$ whenever $A \leftarrow B_1, \ldots, B_n, \mathrm{not}\, C_1, \ldots, \mathrm{not}\, C_m \in P$, $B_1, \ldots, B_n \in S$ and $C_1, \ldots, C_m \notin S$, or $L, \neg L \in S$ for some L.
- S grounded in P iff $A \in S$ implies there is a derivation for A from P valid in S.

Answer sets:
sets of literals closed and grounded in P

Remark

To check whether S is answer set of P

- generate the S-reduct P^S of P:
  1. delete rules with $\mathrm{not}\, C_i$ in body and $C_i \in S$,
  2. delete all not-literals from remaining rules.
- check whether $S = Cn(P^S)$.

Answer set programming

- represent problem such that solutions are (parts of) answer sets
- commonly used method: generate and test

Observation: if P does not contain Q, then

$$Q \leftarrow \mathrm{not}\, Q, body$$

eliminates answer sets satisfying body.
Abbreviation: $\leftarrow body$

Variables in programs

- definition of answer sets for propositional programs
- variables useful for problem descriptions
- $\Rightarrow$ rule with variables as shorthand for all ground instances of the rule
- ASP system: ground instantiator + solver
- instantiator produces ground version of program, solver computes its answer sets

Graph coloring

Description of graph:

$node(v_1), ..., node(v_n), edge(v_i, v_j), \ldots$

Generate:

$col(X, r) \leftarrow node(X), \text{not } col(X, b), \text{not } col(X, g)$
$col(X, b) \leftarrow node(X), \text{not } col(X, r), \text{not } col(X, g)$
$col(X, g) \leftarrow node(X), \text{not } col(X, r), \text{not } col(X, b)$

Test:

$\leftarrow edge(X, Y), col(X, Z), col(Y, Z)$

Answer sets contain solution to problem!

Meeting scheduling

Problem description:

$meeting(m_1), \ldots, meeting(m_n)$

$time(t_1), \ldots, time(t_s)$

$room(r_1), \ldots, room(r_m)$

$person(p_1), \ldots, meeting(p_k)$

$par(p_1, m_1), \ldots, par(p_2, m_3), \ldots$

Problem independent part, generate:

$at(M, T) \leftarrow meeting(M), time(T), \text{not } \neg at(M, T)$

$\neg at(M, T) \leftarrow meeting(M), time(T), \text{not } at(M, T)$

$in(M, R) \leftarrow meeting(M), room(R), \text{not } \neg in(M, R)$

$\neg in(M, R) \leftarrow meeting(M), room(R), \text{not } in(M, R)$

Meeting scheduling, test

Each meeting has assigned time and room:

$timeassigned(M) \leftarrow at(M, T)$

$roomassigned(M) \leftarrow in(M, R)$

$\leftarrow meeting(M), \text{not}\ timeassigned(M)$

$\leftarrow meeting(M), \text{not}\ roomassigned(M)$

No meeting has more than 1 time and room:

$\leftarrow meeting(M), at(M, T), at(M, T'), T \neq T'$

$\leftarrow meeting(M), in(M, R), in(M, R'), R \neq R'$

Meetings at same time need different rooms:

$\leftarrow in(M, X), in(M', X), at(M, T), at(M', T), M \neq M'$

Meetings with same person need different times:

$\leftarrow par(P, M), par(P, M'), M \neq M', at(M, T), at(M', T)$

Things to remember

- answer sets are acceptable sets of beliefs
- straightforward for definite programs: $Cn(P)$
- more difficult with default negation:
  self-referential notion of groundedness
- literals needed for extended programs
- support model based problem solving