Answer set programming

See Eiter et al 2009
Motivation: strong negation

Programme:

man(dilbert).

single(X) :- man(X), not husband(X).
husband(X) :- man(X), not single(X).

Expressing negative fact:
- single(dilbert)
Motivation: strong negation (2)

Negation as failure:

walk :- at(A,L), crossing(L), not train_approaches(L).

What if observation incomplete? (e.g. not having looked)

Strong negation:

walk :- at(A,L), crossing(L), - train_approaches(L).

Requires to know that train is not approaching!
Motivation: Integrity constraints

edge(1,2).
falsity :- not falsity, edge(X,Y), red(X), red(Y).

Which are stable models?

Alternative Notation in Programme:
:- edge(X,Y), red(X), red(Y).
Motivation: default rules

A bird flies by default:

flies(X) :- bird(X), not - flies(X).
penguin(tweety).
bird(X) :- penguin(X).
- flies(tweety).
bird(ducky).
Motivation: disjunction

woman(X) ∨ man(X) :- person(X).
Definition:
An extended logic program (ELP) is a finite set of rules
\[ a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n \]
\((n,m \geq 0)\)
where \(a\) and all \(b_i, c_i\) are atoms or strongly negated atoms in a first-order language \(L\).
Semantics of ELPs

Compilation to Normal Programms:

- For each negative literal "-p(X)"
  - translate it into "notp(X)"
  - add the clause:
    falsity :- not falsity, p(X), notp(X).

- Select the stable models of the resulting program P'. These are called the answer sets of P.

- An atom a is given a three valued view:
  - Either a is true or nota is true or a is undefined
Example ELPs

Given ELP P:
true.
trivial :- true.
a :- true.
-a :- true.

Programme P‘:
true.
trivial :- true.
a :- true.
nota :- true.
falsity :- a, nota.

Trying to find a stable model:
I={true,a,nota}

GL_1(P‘)=
true.
trivial :- true.
a :- true.
nota :- true.
falsity :- a, nota.

I is not a fixpoint!
Example ELP (2)

sci=Science Citation index
ma=Microsoft Academics

website(ma).
website(sci).
up(S) :- website(S), not -up(S).
-query(S) :- -up(S).
query(sci) :- not -query(sci), up(sci).
query(ma) :- not -query(ma), -up(sci), up(ma).
error :- -up(sci), -up(ma).

Single answer set:
M={website(sci),website(ma),
   up(sci), up(ma),
   query(sci)}
website(ma).
website(sci).
up(S) :- website(S), not –up(S).
-query(S) :- -up(S).
query(sci) :- not -query(sci), up(sci).
query(ma) :- not -query(ma), -up(sci), up(ma).
error :- -up(sci), -up(ma).
-query(S) :- not query(S), -reliable(S).
-up(sci).
-reliable(ma)

Two answer sets: M=
{website(sci),website(ma), -up(sci),up(ma), -reliable(ma), -query(sci),query(ma)}
{website(sci),website(ma), -up(sci),up(ma), -reliable(ma), -query(sci),-query(ma)}
Disjunction

- Definite clauses/definite rules
  - $A :\neg L_1, \ldots, L_n$
  - All $L_i$ are positive literals (i.e. atoms)

- Normal clauses/normal rules
  - $A :\neg L_1, \ldots, L_n$
  - All $L_i$ are positive or negative literals (i.e. atoms or negated atoms)

- Extended logic programmes as syntactic sugar for normal logic programmes
Extended disjunctive logic programmes

Definition:
An extended disjunctive logic program (EDLP) is a finite set of rules:

\[ a_1 \lor \ldots \lor a_k :- b_1, \ldots b_m, \text{not } c_1, \ldots, \text{not } c_n \]

\((k,n,m \geq 0)\)

where all \(a_i,b_i,c_i\) are atoms or strongly negated atoms.
Semantics of Extended Logic Programmes

Definition: An interpretation $I$ is a model of

- a ground clause $C$: $a_1 \lor \ldots \lor a_k \leftarrow b_1, \ldots, b_m$, not $c_1, \ldots$, not $c_n$
denoted $I \models C$, if either
  \[ \{b_1, \ldots b_m\} \not\subseteq I \text{ or } \{a_1, \ldots, a_k, c_1, \ldots, c_n\} \cap I \neq \emptyset \]

- a clause $C$, denoted $I \models C$, if $I \models C'$ for every $C' \in \{C\}^*$

- a program $P$, denoted $I \models P$, if $I \models C$ for every $C \in P^*$
Answer Set

Definition:
M is an answer set of the extended disjunctive logic programme P iff M is a minimal model of $GL_M(P)$.

Note: No, one or multiple answer sets may exist for a given programme P.

Note: for non-disjunctive P, a minimal $= \text{the least}$

Example:
man(dilbert).

\[ \text{single}(X) \lor \text{husband}(X) :- \text{man}(X). \]

$M_1 = \{ \text{man}(dilbert), \text{single}(dilbert) \}$

$M_2 = \{ \text{man}(dilbert), \text{husband}(dilbert) \}$
Caution!

\[ a \]

is not the same as

\[ :- \text{ not } a. \]

\[ a \lor b \]

is not the same as

\[ :- \text{ not } a, \text{ not } b. \]
a :- true.

is not the same as

:- not a.

Models: {a}

Models: none

a ∨ b :- true.

is not the same as

:- not a, not b.

Models: {a}, {b}

Models: none
WeST

Caution

\[ a \lor b \leftarrow \text{true}. \]
\[ a \leftarrow b. \]

Models: \{a\}

\[ a \leftarrow \neg b. \]
\[ b \leftarrow \neg a. \]
\[ a \leftarrow \neg b. \]

Models: \{a\}
Caution!

Not head-cycle free programs, e.g.:

(1) $p \lor q$ :-
(2) $p$ :- $q$.
(3) $q$ :- $p$.

{} not a model, bec. (1)
{p} not a model, bec (3)
{q} not a model, bec (2)
{p,q} is a model.

Try translation to unstratified negation:

(1') $p$ :- not $q$.
(2') $q$ :- not $p$.
(3') $p$ :- $q$.
(4') $q$ :- $p$.

{} not a model, bec. (1')
{p} not a model, bec (4')
{q} not a model, bec (3')
{p,q} is not stable!
Comparing SAT and Answer Set Programming

Theorem:
Deciding whether a given ground disjunctive program $P$ has some answer set is $\Sigma_2^P$-complete in general.

This is the class of problems decidable in polynomial time on a nondeterministic Turing machine with an oracle for solving problems in NP (i.e. $\text{NP}^{\text{NP}}$).

Complexity for nonground EDLP is $\text{NEXP}^{\text{NP}}$-complete.

Compare:
Deciding whether a SAT formula has a model is NP complete.

Much simpler!
Reviewer Selection: Cyclic Negation (T. Eiter)

(1) paper(p1). paper(p2).
(2) cand("Thomas"; p1). cand("Enrico"; p2). cand("Marco"; p2).
(3) assign(X,P) :- cand(X,P), not -assign(X,P).
(4) -assign(Y,P) :- cand(Y,P), assign(X,P), not X=Y.
(5) is_assigned(P) :- assign(X,P).
(6) paper(P) :- not is_assigned(P).

(3)+(4): Choice of one element using unstratified rules (cyclic negation)

Answer sets:

$M_1 = \{\ldots, assign("Thomas", p1), assign("Enrico", p2), -assign("Marco", p2)\}$
$M_2 = \{\ldots, assign("Thomas", p1); assign("Marco", p2), -assign("Enrico", p2)\}$
Reviewer Selection: Cyclic Negation (T. Eiter)

(1) paper(p1). paper(p2).
(2) cand("Thomas"; p1). cand("Enrico"; p2). cand("Marco"; p2).
(3) assign(X,P) :- cand(X,P), not -assign(X,P).
(4) -assign(Y,P) ∨ -assign(X,P) :- cand(Y,P), cand(X,P), not X=Y.
(5) is_assigned(P) :- assign(X,P).
(6) paper(P) :- not is_assigned(P).

(3)+(4): Choice of one element using unstratified rules (cyclic negation)

Answer sets:

\[ M_1 = \{ \ldots, \text{assign("Thomas",p1), assign("Enrico",p2), -assign("Marco", p2) } \} \]
\[ M_2 = \{ \ldots, \text{assign("Thomas",p1); assign("Marco",p2), -assign("Enrico"; p2) } \} \]
Some Properties of Answer Sets

Minimality, Non-monotonicity

- Every answer set $M$ of $P$ is a minimal model of $P$.

E.g.

- $P = \{a :\neg b\} \quad M = \{a\}$
- $P = \{a :\neg b. b.\} \quad M = \{b\}$
Supportedness

- Given an answer $M$ of $P$, for every literal $a \in M$ there is some rule $r \in P^*$ such that $M \models \text{Body}(r)$ and $M \cap \text{Head}(r) = \{a\}$. 
Some Properties of Answer Sets

Failure of Cumulativity

- From $a \in M$, for each answer set $M$ of $P$, it does not follow that $P$ and $P \cup \{a\}$ have the same answer sets (even if $P$ has answer sets).
ANSWER SET PROGRAMMING EXAMPLE AND APPLICATIONS
Answer Set Solvers

- DLV http://www.dbai.tuwien.ac.at/proj/dlv/ *
- Smodels http://www.tcs.hut.fi/Software/smodels/ **
- GnT http://www.tcs.hut.fi/Software/gnt/
- Cmodels http://www.cs.utexas.edu/users/tag/cmodels/
- ASSAT http://assat.cs.ust.hk/
- NoMore(++) http://www.cs.uni-potsdam.de/~linke/nomore/
- Platypus http://www.cs.uni-potsdam.de/platypus/
- clasp http://www.cs.uni-potsdam.de/clasp/
- XASP http://xsb.sourceforge.net, distributed with XSB v2.6
- aspps http://www.cs.engr.uky.edu/ai/aspps/
- ccalc http://www.cs.utexas.edu/users/tag/cc/

* + extensions, e.g. DLVEX, DLVHEX, DLV\textsuperscript{DB}, DLT, DLV-complex ** + Smodels\textsubscript{cc}

- Several provide a number of extensions to the language described here.
- Answer Set Solver Implementation: see Niemelä's ICLP'04 tutorial.
- ASP Solver competition: see LPNMR 2007 conference;
- ASPARAGUS Benchmark platform http://asparagus.cs.uni-potsdam.de/
Applications of ASP

- Diagnosis
- Information integration
- Constraint satisfaction
- Reasoning about actions (including planning)
- Routing and scheduling
- Security analysis
- Configuration
- Computer-aided verification
- Semantic web
- Question answering

Take it with a grain of salt: Most knowledge representations can be used for anything!
Encoding of Problems in ASP

Problem Instance \( I \) → Encoding: Program \( P \) → ASP Solver → Model(s) Solution(s)
Encoding of Problems in ASP (2)
Example: 3 colorings

Graph G=(V,E)
V={a,b,c,d}
E={(a,b),(b,c),(c,a),(a,d)},
Encode legal three colorings.
For each node n have atoms bn, rn, gn informally meaning that node n is colored blue, red, green.

Facts:
e(a,b). e(b,c). e(c,a). e(a,d).

Constraints:
:- b(X), g(X).
... 
:- e(X,Y), b(X), b(Y).
:- e(X,Y), r(X), r(Y).
:- e(X,Y), g(X), g(Y).

Rules:
b(X) ∨ r(X) ∨ g(X) :- .
Use of Double Negation

Example: Employees e and salary S

Relation: emp(E,S).

maximum: s*=max{s|empl(e,s)∈D}

% salary S is *not* maximal
-max(S) :- empl(E,S), empl(E1,S1), S<S1.
% double negation
max(S) :- empl(E,S), not –max(S).
Natural definition of greatest common divisor

% Declare when \( D \) divides a number \( N \).
\[
divisor(D,N) \leftarrow \text{int}(D), \text{int}(N), \text{int}(M), N = D \times M.
\]

% Declare common divisors
\[
cd(T,N_1,N_2) \leftarrow \text{divisor}(T,N_1), \text{divisor}(T,N_2).
\]

% Single out non-maximal common divisors \( T \)
\[
-\gcd(T,N_1,N_2) \leftarrow \text{cd}(T,N_1,N_2), \text{cd}(T_1,N_1,N_2), T < T_1.
\]

% Apply double negation: take non non-maximal divisor
\[
\gcd(T,N_1,N_2) \leftarrow \text{cd}(T,N_1,N_2), \text{not} - \gcd(T,N_1,N_2).
\]
"guess and check" methodology

"Generate and test" / Planning

Idea:

- Use nondeterminism that comes with unstratified negation and/or disjunction in rule heads to create candidate solutions to a problem (Part G)
- Check with further rules and/or constraints (Part C)
  - Checking may involve auxiliary predicates if needed

Example: 3-coloring

Generate: \( b(X) \lor r(X) \lor g(X) :- . \)

Check: \( :- e(X,Y), g(X), g(Y). \)  Etc.
**Auxiliary predicates**

Given a directed graph $G = (V,E)$, 
a path $n_0 \to n_1 \to \cdots \to n_k$ in $G$ from a start node $n_0 \in V$ is called a *Hamiltonian path*, if all nodes $n_i$ are distinct and each node in $V$ occurs in the path, i.e., $V = \{n_0, \ldots, n_k\}$.

*Example 37.* The graph $G$ is stored using the predicates $\text{node}(X)$ and $\text{edge}(X, Y)$ and the predicate $\text{start}(X)$ stores the unique node $n_0$. Consider:

\[
\text{inPath}(X, Y) \lor \text{outPath}(X, Y) \leftarrow \text{edge}(X, Y). \]  \quad \text{Guess}
\]

\[
\left\{ \begin{array}{l}
\leftarrow \text{inPath}(X, Y), \text{inPath}(X, Y_1), Y \neq Y_1. \\
\leftarrow \text{inPath}(X, Y), \text{inPath}(X_1, Y), X \neq X_1. \\
\leftarrow \text{node}(X), \text{not reached}(X).
\end{array} \right. \]  \quad \text{Check}
\]

\[
\text{reached}(X) \leftarrow \text{start}(X). \\
\text{reached}(X) \leftarrow \text{reached}(Y), \text{inPath}(Y,X). \]  \quad \text{Auxiliary Predicate}
Example: Assignments

Computer Science Department cs

member(sam,cs).  course(java,cs).  course(ai,cs).
member(bob,cs).  course(c,cs).  course(logic,cs).
member(tom,cs).

likes(sam,java).  likes(sam,c).
likes(bob,java).  likes(bob,ai).
likes(tom, ai ).  likes(tom, logic).

Our task is now to assign each member of the department some courses, such that (i) each member should have at least one course, (ii) nobody has more than two courses and (iii) only courses are assigned that the course leader likes.
Example: Assignments (2)

% assign a course that one likes
teaches(X,Y) ← member(X,cs), course(Y,cs), likes(X,Y), not ¬teaches(X,Y).

% do not assign a course that is taught by someone else
¬teaches(X,Y) ← member(X,cs), course(Y,cs), teaches(X1,Y), X1 ≠ X.

% describe if someone teaches at least one course
some_course(X) ← member(X,cs), teaches(X,Y).

% prevent that someone does not teach a course
← member(X,cs), not some course(X).

% noone teaches more than two courses
← teaches(X,Y1), teaches(X,Y2), teaches(X,Y3), Y1 ≠ Y2, Y1 ≠ Y3, Y2 ≠ Y3.
Example: Assignments (3)

We obtain the following three answer sets of $P \cup F$:

\{teaches(sam, c), teaches(bob, java), teaches(bob, ai ), \\
teaches(tom, logic), \ldots\}\}

\{teaches(sam, java), teaches(sam, c), teaches(bob, ai ), \\
teaches(tom, logic), \ldots\}\}

\{teaches(sam, c), teaches(bob, java), teaches(tom, ai ), \\
teaches(tom, logic), \ldots\}\}
Saturation – Encoding Co-NP-hard problems

Examples:
- Determining that a SAT problem is not satisfiable
- Determining that a graph is not 3-colorable

Program for non-3-colorability:
\[
\begin{align*}
b(X) &\lor r(X) \lor g(X) \leftarrow \text{node}(X). \\
\text{noncol} &\leftarrow r(X), r(Y), \text{edge}(X, Y).
\end{align*}
\]

Effects:
If graph is 3-colorable
- Each model where the graph is correctly 3-colored is a minimal model
- If the graph is incorrectly colored the model will color all nodes in all colors, returning a „maximal“ model
- Comparing the two models the minimal models will win

If graph is not 3-colorable
- It will only have one model where all nodes have all colors
Program for non-3-colorability:

\[ b(X) \lor r(X) \lor g(X) \leftarrow \text{node}(X). \]
\[ \text{noncol} \leftarrow r(X), r(Y), \text{edge}(X, Y). \]
\[ \text{noncol} \leftarrow g(X), g(Y), \text{edge}(X, Y). \]
\[ \text{noncol} \leftarrow b(X), b(Y), \text{edge}(X, Y). \]
\[ b(X) \leftarrow \text{noncol}, \text{node}(X). \]
\[ r(X) \leftarrow \text{noncol}, \text{node}(X). \]
\[ g(X) \leftarrow \text{noncol}, \text{node}(X). \]

**Effects:**

If graph is 3-colorable
- Each model where the graph is correctly 3-colored is a minimal model
- If the graph is incorrectly colored the model will color all nodes in all colors, returning a „maximal“ model
- Comparing the two models the minimal models will win

If graph is not 3-colorable
- It will only have one model where all nodes have all colors

More expressiveness than propositional logics! (even for the ground case!)

Ensure full saturation!
Saturation Recipe

Check that a property $Pr$ holds for all guesses defining a search space, using a guess and saturation check:

- A subprogram $Pguess$ defines search space
- A subprogram $Pcheck$ checks $Pr$ for a guess $Mg$.
- If $Pr$ holds for $Mg$, saturation rules $Psat$ generate the special candidate answer set $Msat$.
- If $Pr$ does not hold for $Mg$, an answer set results which is a strict subset of $Msat$ (thus preventing that $Msat$ is an answer set).

It is thus crucial that the program $Pcheck$, which formalizes $Pr$, and $Psat$ do not generate incomparable answer sets.