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) $\vee$ man(X) :- person(X).

Extended Logic Programs

Definition:

An extended logic program (ELP) is a finite set of rules

$$a \text{ :- } b_1,\ldots b_m, \text{ not } c_1,\ldots, \text{ not } c_n$$

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

where a and all b_i,c_i are atoms or strongly negated atoms in a first-order language L.

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, not falsity.

Trying to find a stable model:

I={true,a,nota}

$GL_I(P')$={

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)}

Example ELP (3)

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

With or without function symbols!

- 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:

$$a_1 \vee \ldots \vee a_k \text{ :- } b_1, \ldots b_m, \text{ not } c_1, \ldots, \text{ not } c_n$$

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

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: $a_1 \vee \ldots \vee a_k$:- $b_1,\ldots b_m$, not $c_1,\ldots,$ not c_n

  denoted I |= C, if either

  $\{b_1,\ldots b_m\} \not\subseteq$ I or $\{a_1,\ldots,a_k,c_1,\ldots,c_n\} \cap I \neq \{\}$

- a clause C, denoted I |= C, if I |= C‘ for every C‘ $\in$ {C}*

- a program P, denoted I |= P, if I |= 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 = the least*

Example:

man(dilbert).

single(X) $\vee$ husband(X) :- man(X).

M_1 = {man(dilbert),single(dilbert)}

M_2 = {man(dilbert),husband(dilbert)}

Caution!

a

is not the same as

:- not a.

a $\vee$ b

is not the same as

:- not a, not b.

Caution!

a :- true.	Models: {a}
is not the same as	
:- not a.	Models: none
a ∨ b :- true.	Models: {a}, {b}
is not the same as	
:- not a, not b.	Models: none

Caution

a ∨ b :- true.

a :- b.

Models: {a}

a :- not b.

b :- not a.

a :-b.

Models: {a}

Caution!

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

(1) $p \vee q$:-
(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 Σ_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. NP^{NP}).

Complexity for nonground EDLP is $NEXP^{NP}$-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 = {...assign("Thomas",p1),assign("Enrico",p2), -assign("Marco“, p2)}

M2 = {...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) $\vee$ -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 = {...assign("Thomas",p1),assign("Enrico",p2), -assign("Marco“, p2)}

M2 = {...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 :- not b} M = {a}
- P = {a :- not b. b.} M = {b}

Some Properties of Answer Sets

Supportedness

- Given an answer M of P, for every literal a ∈ M there is some rule r ∈ P* such that M |= Body(r) and M∩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^{DB}, DLT, DLV-complex ** + Smodels_cc

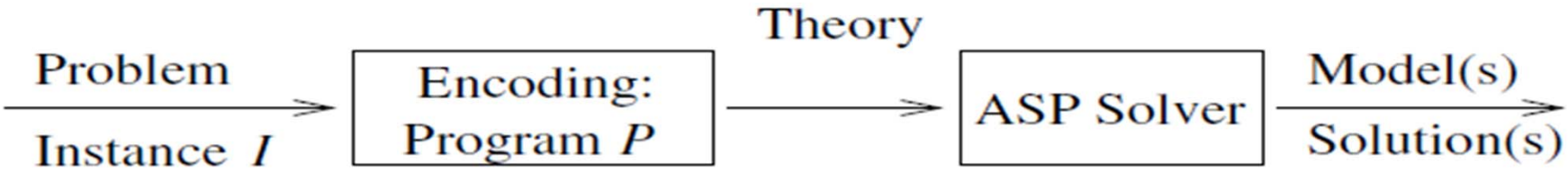
- Several provide a number of extensions to the language described here.
- Answer Set Solver Implementation: see Niemelä's ICLP'04 tutorial.
- ASP Solver competition: see LPNMR 2007 conference;
- ASPARAGUS Benchmark platform http://asparagus.cs.uni-potsdam.de/

Applications of ASP

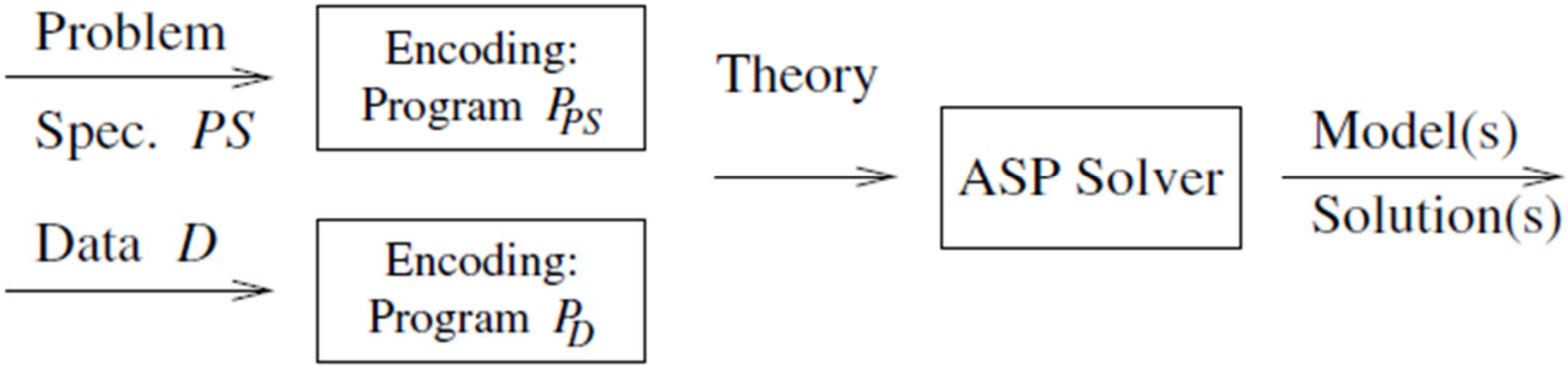
- Diagnosis
- Informaiton integration
- Constraint satisfaction
- Reasoning about actions (including planning)
- Routing and scheduling
- Security analysis
- Configuration
- Computer-aided verification
- Semantic web
- Question answering

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

Encoding of Problems in ASP



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.

SAT like problem!
But more convenient to encode.

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:

$b(X) \vee r(X) \vee g(X)$:- .

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*.

$divisor(D,N) \leftarrow int(D),\ int(N),\ int(M), N = D * M.$

% Declare common divisors

$cd(T,N1,N2) \leftarrow divisor(T,N1),\ divisor(T,N2).$

% Single out non-maximal common divisors T

$-gcd(T,N1,N2) \leftarrow cd(T,N1,N2),\ cd(T1,N1,N2),\ T < T1.$

% Apply double negation: take non non-maximal divisor

$gcd(T,N1,N2) \leftarrow cd(T,N1,N2),\ not\ -gcd(T,N1,N2).$

„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: $b(X) \vee r(X) \vee g(X)$:- .

Check: :- e(X,Y), g(X), g(Y). Etc.

Auxiliary predicates

Given a directed graph $G = (V,E)$,

a path $n0 \rightarrow n1 \rightarrow \cdots \rightarrow nk$ in G from a start node $n0 \in V$ is called a *Hamiltonian path*, if all nodes ni are distinct and each node in V occurs in the path, i.e., $V = \{n0, \ldots, nk\}$.

Example 37. The graph G is stored using the predicates $node(X)$ and $edge(X, Y)$ and the predicate $start(X)$ stores the unique node $n0$. Consider:

$inPath(X, Y) \vee outPath(X, Y) \leftarrow edge(X, Y).$ } **Guess**

$\leftarrow inPath(X, Y), inPath(X, Y1), Y \neq Y1.$
$\leftarrow inPath(X, Y), inPath(X1, Y), X \neq X1.$
$\leftarrow node(X), \text{not } reached(X).$
} **Check**

$reached(X) \leftarrow start(X).$
$reached(X) \leftarrow reached(Y), inPath(Y,X).$
} **Auxiliary Predicate**

Example: Assignments

Computer Science Department cs

member(*sam,cs*).
member(*bob,cs*).
member(*tom,cs*).

course(*java,cs*).
course(*c,cs*).

course(*ai,cs*).
course(*logic,cs*).

likes(*sam,java*).
likes(*bob,java*).
likes(*tom, ai*).

likes(*sam,c*).
likes(*bob,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*(*X*1, *Y*),*X*1 ≠ *X*.

% describe if someone teaches at least one course

some_course(*X*) ← *member*(*X*, *cs*), *teaches*(*X*, *Y*).

% prevent that someones does not teach a course

← *member*(*X*, *cs*), not *some course*(*X*).

% noone teaches more than two courses

← *teaches*(*X*, *Y*1), *teaches*(*X*, *Y*2), *teaches*(*X*, *Y*3),*Y*1 ≠ *Y*2, *Y*1 ≠ *Y*3, *Y*2 ≠ *Y*3.

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), . . .}

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

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

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:

$b(X) \vee r(X) \vee g(X) \leftarrow node(X).$

$noncol \leftarrow r(X), r(Y), edge(X, Y).$

$noncol \leftarrow g(X), g(Y), edge(X, Y).$

$noncol \leftarrow b(X), b(Y), edge(X, Y).$

$b(X) \leftarrow noncol, node(X).$

$r(X) \leftarrow noncol, node(X).$

$g(X) \leftarrow noncol, node(X).$

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

Saturation – Encoding Co-NP-hard problems

Ensure full saturation!

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

Program for non-3-colorability:

$b(X) \vee r(X) \vee g(X) \leftarrow node(X).$

$noncol \leftarrow r(X), r(Y), edge(X, Y).$

$noncol \leftarrow g(X), g(Y), edge(X, Y).$

$noncol \leftarrow b(X), b(Y), edge(X, Y).$

$b(X) \leftarrow noncol, node(X).$

$r(X) \leftarrow noncol, node(X).$

$g(X) \leftarrow noncol, node(X).$

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

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.