Answer Set Programming: an Introduction

Stefania Costantini

Dipartimento di Informatica
Universita' degli Studi di L'Aquila
via Vetoio Loc. Coppito, I-67100 L'Aquila (Italy)
stefcost@di.univaq.it

Example: 3-coloring in Logic Programming

Problem: assigning colors red/blue/green to vertices of a graph, so as no adjacent vertices have the same color.

Logic programming representation of the graph:

$node(0..3).$

$col(red).$
$col(blue).$
$col(green).$

$edge(0,1).$
$edge(1,2).$
$edge(2,0).$
$edge(1,3).$
$edge(2,3).$

... Inference Engine ...

Expected solutions:

$\{color(0,red), color(1,blue), color(2,green), color(3,red)\}$
$\{color(0,red), color(1,green), color(2,blue), color(3,red)\}$
$\{color(0,blue), color(1,red), color(2,green), color(3,blue)\}$
$\{color(0,blue), color(1,green), color(2,red), color(3,blue)\}$
$\{color(0,green), color(1,blue), color(2,red), color(3,green)\}$
$\{color(0,green), color(1,red), color(2,blue), color(3,green)\}$

3-coloring in Prolog

$graph\_coloring(G, L) : -gr\_col(G, [], L).$

$gr\_col([], L, L) : -!.$
$gr\_col([N|R], P, L) : -vertex\_color(N, C, P), gr\_col(R, [color(N, C)|P], L).$

$vertex\_color(N, C, P) : -col(C), ok\_color(N, C, P).$
$neighbourhood(N, B, L) : -(edge(N, N1); edge(N1, N)),$
$member(N1, B), !,$
$neighbourhood(N, [N1|B], L).$
$neighbourhood(\_, L, L).$

$ok\_color(N, C, P) : -neighbourhood(N, [], L), check\_color(L, C, P).$

$check\_color([], \_, \_) : -!.$
$check\_color([N1|B], C, P) : - \setminus + member(color(N1, C), P), !,$
$check\_color(B, C, P).$

$member(E, [E|X]) : -!.$
$member(E, [\_|X]) : -member(E, X).$

$? - consult[3col.pro]$
$? - graph\_coloring([0, 1, 2, 3], L).$
$L = [color(0, red), color(1, blue), color(green), color(3, red)];$
$L = [color(0, red), color(1, green), color(blue), color(3, red)]$
$\ldots$

3-coloring in Answer Set Programming

$color(X, red)|color(X, blue)|color(X, green) : -node(X).$

$: -edge(X, Y), col(C), color(X, C), color(Y, C).$

$hide\, node(X).$
$hide\, edge(X, Y).$
$hide\, col(C).$

Using the SMODELS inference engine we obtain:
$lparse\ <\ 3col.txt \mid smodels\, 0$

$Answer1$
$\{color(0, red), color(1, blue), color(green), color(3, red)\}$
$Answer2$
$\{color(0, red), color(1, green), color(blue), color(3, red)\}$
$\ldots$

Horn Logic Programming

Unification + SLD-Resolution

Least (Herbrand) model semantics

No negation but, implicitly, equality (as identity) and disequality.

$not\_member(E, [])$.
$not\_member(E, [T|X]) \leftarrow E \neq T, not\_member(E, X)$.

Least Herbrand Model: Example

$p \leftarrow g, h.$
$g \leftarrow r, s.$
$r \leftarrow f.$
$s.$
$f.$
$h.$

Step 1: facts

$M_0 = \{s, f, h\}$

Step 2: $M_1 = M_0$ plus what I can derive from facts

$M_1 = \{s, f, h, r\}$

Step 3: $M_2 = M_1$ plus what I can derive from M_1

$M_2 = \{s, f, h, r, g\}$

Step 4: $M_3 = M_2$ plus what I can derive from M_2

$M_2 = \{s, f, h, r, g, p\}$

If I try to go on, no more added conclusions: fixpoint
(Least Herbrand Model, in this case finite)

Least Herbrand Model: Example

$nat(0).$
$nat(s(X)) \leftarrow nat(X).$

Step 1: facts

$M_0 = \{nat(0)\}$

Step 2: $M_1 = M_0$ plus what I can derive from facts

$M_1 = \{nat(0), nat(s(0))\}$

Step 3: $M_2 = M_1$ plus what I can derive from M_1

$M_2 = \{nat(0), nat(s(0)), nat(s(s(0)))\}$

Step 4,5,6... add $nat(s \ldots s(0) \ldots)$

Least Herbrand Model, in this case infinite, fixpoint after an infinite (denumerable) number of steps.

Horn Logic Programming

Function symbols and recursive definitions
(infinite Herbrand Universe)
Expressivity: Church-Turing computability

DATALOG: no function symbols
(finite Herbrand Universe, finite program grounding)
Expressivity: proper subset of P

Datalog

Datalog Program

$p(X) \leftarrow r(X), q(X).$
$r(Z) \leftarrow s(Z).$
$s(a).$
$s(b).$
$q(b).$

Its grounding

$p(a) \leftarrow r(a), q(a).$
$p(b) \leftarrow r(b), q(b).$
$r(a) \leftarrow s(a).$
$r(b) \leftarrow s(b).$
$s(a).$
$s(b).$
$q(b).$

Its Least Herbrand Model

$M = \{s(a), s(b), q(b), r(a), r(b), p(b)\}$

Datalog

A shortcut

$p1 \leftarrow r1, q1.$
$p2 \leftarrow r2, q2.$
$r1 \leftarrow s1.$
$r2 \leftarrow s2.$
$s1.$
$s2.$
$q2.$

$M = \{s1, s2, q2, r1, r2, p2\}$

Terminology: $p \leftarrow q, not\, r$ is a *rule*, where p is the *head*, or *conclusion*, and $q, not\, r$ is the *body*, or the *conditions*. p, q and r are *atoms*. Atoms which occur positively in the rule are *positive literals*, and the negations are *negative literals*. The same atom can occur in a rule both as a positive literal, and inside a negative literal (e.g. rule $p \leftarrow not\, p$).

Negation in Logic Programming

Negation operator not (\+)

Procedural meaning in Prolog:

Negation As (finite) Failure

Declarative semantics:

based on Closed World Assumption

No problem, procedurally, for so-called stratified programs

$q \leftarrow not\, r.$

$r \leftarrow not\, g.$

$g \leftarrow not\, h.$

But declaratively? Two minimal models

$\{r, h\}$ (not intended) and

$\{q, g\}$ (intended)

Unique least model no longer guaranteed!

Negation in Logic Programming

Procedural problems also possible:

$p \leftarrow q, not\, s.$
$q.$
$r \leftarrow a.$
$r \leftarrow b.$
$a \leftarrow not\, b.$
$b \leftarrow not\, a.$
Classical models $\{p, q, r, a\}$ and $\{p, q, r, b\}$, $\{s, q, r, a\}$ and $\{s, q, r, b\}$, procedurally, loop $a - b$

Empty least model possible

$p \leftarrow not\, p.$
$q \leftarrow p.$
Classical least model $\{\}$.

Negation in Logic Programming: Semantics

First group of semantic proposals:
keep a single intended model

By narrowing class of programs:
perfect model (stratification) (Apt, Blair & Walker), only stratified programs allowed

By weakening semantics (3-valued models):
well-founded model $WFS = \langle T; F \rangle$
(Van Gelder, Ross & Schlipf)
In the example:
$\langle T = \{p, q\}; F = \{f\} \rangle$
all the other atoms are "undefined", i.e., $U = \{a, b, r\}$

Atoms which are classical logical consequences can be undefined! (r in the example)

Second group of semantic proposal:
collection of *intended* models

- supported models (Clark)
- stable models (Gelfond & Lifschitz)

Answer Set/Stable model semantics

$p :- not\, p.$
Classical minimal model $\{p\}$ NOT STABLE: no stable models!
$a :- not\, b.$
$b :- not\, a.$
Classical minimal models $\{a\}$, $\{b\}$ STABLE

$p :- not\, p, not\, a.$
$a :- not\, b.$ Classical minimal models
$b :- not\, a.$
$\{b, p\}$ NOT STABLE
$\{a\}$ STABLE

$p :- not\, p.$
$p :- not\, a.$
$a :- not\, b.$
$b :- not\, a.$
Classical minimal models
$\{b, p\}$ STABLE
$\{a, p\}$ NOT STABLE

Stable Models = Answer Sets
Stable Model Semantics = Answer Set Semantics.

Answer Set semantics

$p \leftarrow q, not\, s.$
$q.$
$g \leftarrow s, q.$
$r \leftarrow a, q.$
$r \leftarrow b, not\, s.$
$a \leftarrow not\, b.$
$b \leftarrow not\, a.$
Classical minimal models $\{p, q, r, a\}$, $\{p, q, r, b\}$, $\{s, q, g, r, a\}$ and $\{s, q, g, r, b\}$.

Answer Sets $\{p, q, r, a\}$ and $\{p, q, r, b\}$.

Relation to classical logic:
every answer set is a minimal model
not all minimal models are answer sets.

Relation to WFS: for every answer set S, $T \subseteq S$,
i.e., the Answer Set Semantics
assigns a truth value to undefined atoms.
Here, WFS $= \langle T = \{p, q\}; F = \{s, g\}\rangle$.

How to find answer sets

Underlying concept: a minimal model is an answer set only if no atom which is true in the model depends (directly or indirectly) upon the negation of another atom which is true in the model.

How to find answer sets: there are various ways.

If you have a classical minimal model M of program P where P has no positive cycles: check if it is supported $\forall A \in M \; \exists$ rule $\rho \in P$ with head A such that all conditions of ρ are true in P.

$p \leftarrow q, not\, s.$
$q.$
$g \leftarrow s, q.$
$r \leftarrow a, q.$
$r \leftarrow b, not\, s.$
$a \leftarrow not\, b.$
$b \leftarrow not\, a.$

Check minimal model $\{p, q, r, a\}$:
supported, then it is an answer set.
Check minimal model $\{s, q, g, r, a\}$:
not supported, then it is not an answer set

How to find answer sets

If you have just an interpretation I, apply the Gelfond-Lifschitz Γ operator by steps (i-iii) below:
(i) cancel from P all rules whose conditions are false in I;
(ii) cancel the negative literals in all the other rules;
(iii) find the Least Herbrand model of the residual positive program.
Then, I is an answer set iff $I = \Gamma(I)$.

$p \leftarrow q, not\, s.$
$q.$
$g \leftarrow s, q.$
$r \leftarrow a, q.$
$r \leftarrow b, not\, s.$
$a \leftarrow not\, b.$
$b \leftarrow not\, a.$

Check interpretation $\{s, q, g, r, a\}$, residual program is

$q.$
$g \leftarrow s, q.$
$r \leftarrow a, q.$
$a.$

Its Least Herbrand Model is $\{q, r, a\}$, then this is not an answer set.

How to find answer sets

Simplify the program w.r.t. the WFS = $\langle T; F \rangle$, by steps (a-c) below:

(a) cancel all rules with a condition false w.r.t. the WFS;
(b) cancel from all the other rules all literals which are true w.r.t. the WFS;
(c) remove facts.

Then, all the other atoms in the residual program are undefined w.r.t. the WFS.
Try to find answer sets of this residual program.
Add the set T, for finding the answer sets of the original program.

Many ASP solvers use this method, and adopt various strategies for finding the answer sets of the residual program.

How to find answer sets

$p \leftarrow q, not\ s.$
$q.$
$g \leftarrow s, q.$
$r \leftarrow a, q.$
$r \leftarrow b, not\ s.$
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

WFS $= \langle T = \{p, q\}; F = \{s, g\}\rangle$. Residual program is:

$r \leftarrow a.$
$r \leftarrow b.$
$a \leftarrow not\ b.$
$b \leftarrow not\ a.$

Drawbacks of Answer Set semantics

Fix-point definition:
interpretation I is an answer set iff $I = \Gamma(I)$

No Relevance :
for atom A, $REL\_RUL(A)$ may have answer sets where A is true/false, while the overall program does not.

Example :
$p :- not\, p, not\, a.$

$q :- not\, q, not\, b.$

$a :- b.$
$b :- a.$

$REL\_RUL(a) = \{a :- b.\ b :- a.\}$
with answer sets $\{a\}$ and $\{b\}$.

Overall program: no stable models.

P_1 with answer sets, P_2 with answer sets
$P_1 \cup P_2$ may not have answer sets.

Stable Model Semantics (Answer Sets Semantics)

Nice formal features:
related to well-founded semantics, and default logic

Difficult to reconcile with query-based logic programming (skeptical semantics? unique stable model? Too complex!)

Solution: new logic programming paradigm

- SLP: Stable Logic Programming
  (Marek & Truszczynski)
- ASP: Answer Set Programming (Gelfond & Lifschitz)

for for DATALOG plus negation (no function symbols, or limited use)

Solutions are represented by stable models (answer sets), and not by answer substitutions in response to a query.

Inference engine: answer set solvers
SMODELS, Dlv, DeRes, CCalc

Complexity: existence of a stable model NP-complete

Expressivity:

- all decision problems in NP and
- all search problems whose associated decision problems are in NP (claim!)

Answer Set Programming

Constraints

$:\!-v, w, z.$
rephrased as
$p :\!-not\, p, v, w, z.$

Disjunction

$v|w|z.$
rephrased as
$v :\!-not\, w, not\, z.$
$w :\!-not\, v, not\, z.$
$z :\!-not\, v, not\, w.$

XOR

$v + w + z.$
rephrased as
$v|w|z.$
$:\!-w, z.$
$:\!-v, z.$
$:\!-v, w.$

Answer Set Programming

Classical Negation $\neg p$

$-p :- q, r.$
rephrased as
$p' :- q, r.$
$:- p, p'$

Example

$mushroom(boletus\_edulis).$
$mushroom(amanita\_phalloides).$

$eat(M) :- mushroom(M), -poisonous(M).$

$not\ poisonous$ would be hazardous!

$discard(M) :- mushroom(M), poisonous(M).$

$poisonous(M) :- not\ -poisonous(M).$
$-poisonous(boletus\_edilis).$

Answer Set Programming (SMODELS)

Choice rules

$n\{p(X,Y) : q(Y)\}m :- r(X).$

Meaning: forall X such that $r(X)$ holds, each answer set must contain from n to m literals of the form $p(X,Y)$, given that $q(Y)$ holds. r and q define the *domains* of variables X and Y respectively.

Interesting special case, when only one $p(X,Y)$ is allowed: Unique solution allowed

$1\{p(X,Y) : q(Y)\}1 :- r(X).$

Examples

$1\{lives(X,Y) : place(Y)\}1 :- person(X).$

$1\{enrolled(X,Y) : course(Y)\}n :- student(X).$

Answer Set Programming: deals with Uncertainty

$new(n1).$
$new(n2).$
$source(n1, a1).$
$association(a1).$
$source(n2, government).$

$economical(N) : -new(N), source(N, A),$
$\qquad\qquad\qquad\qquad economical\_association(A).$
$cultural(N) : -new(N), source(N, A), cultural\_association(A).$
$political(N) : -new(N), source(N, government).$

$cultural\_association(A) : -association(A),$
$\qquad\qquad\qquad\qquad not\, economical\_association(A).$
$economical\_association(A) : -association(A),$
$\qquad\qquad\qquad\qquad not\, cultural\_association(A).$

$economical(N) : -new(N), not\, cultural(N).$
$cultural(N) : -new(N), not\, political(N), not\, economical(N).$
$political(N) : -new(N), not\, cultural(N).$

Answer Set Programming: deals with Uncertainty

Prolog:

$? - economical(n1).$
$? - political(n1).$
$? - cultural(n1)$
infinite loop!

$? - economical(n2).$
$? - cultural(n1)$
infinite loop!

$? - political(n2).$ yes

Well-Founded Semantics:
truth value undefined instead of infinite loop

Answer Set Solver (SMODELS):
Answer1
$\{political(n2), economical(n2), cultural(n1), cultural\_association(a1)\}$
Answer2
$\{political(n2), economical(n2), political(n1), political\_association(a1)\}$

Each Answer Set makes a different (consistent) hypothesis
on association a1 (political/cultural).

Answer Set Programming and Intelligent Agents

Planning oriented Agents: cycle *observe-think-act*

- observe: collect new facts;
- think: answer set programming for making a plan;
- act: execute the plan.

Reactive/proactive agents: cycle *event-condition-action*

- Reactive: events come asynchronously form the outside, or from the inside;
- Proactive: conditions are internal inference processes, possibly involving answer set programming for checking constraints, resolving uncertainty, making a plan;
- Able to act, and affect the environment;
- Able to remember, past events, conditions and actions can be used in further inference.