AUTOMATED REASONING

Agostino Dovier

Università di Udine
CLPLAB

Udine, November 2016

Let us consider programs consisting of rules of the kind:

$$H \leftarrow B_1, \ldots, B_m, \texttt{not}\ C_1, \ldots, \texttt{not}\ C_n \tag{1}$$

where H, B_i, C_j are atoms, $n \geq 0$, $m \geq 0$ is said an (ASP) rule.

Sets of these rules are called *general programs*.

An extended T_P can be defined:

$$T_P(I) = \left\{ a : \begin{array}{l} a \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n \in \text{ground}(P), \\ \{b_1, \ldots, b_m\} \subseteq I, \\ \{c_1, \ldots, c_n\} \cap I = \emptyset \end{array} \right\}$$

EXAMPLE

Let $P = p(a) \leftarrow \texttt{not}\ p(b)$ (it's the theory: $T = p(a) \vee p(b)$).
There are 4 Herbrand interpretations:

$$\begin{tikzcd}
 & \{p(a), p(b)\} & \\
\{p(a)\} \arrow[ur] & & \{p(b)\} \arrow[ul] \\
 & \emptyset \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

3 of them are models. There is no A such that $T \models A$.

Moreover, $T_P(\emptyset) = \{p(a)\}$, $T_P(\{p(b)\}) = \emptyset$: this is not monotone.

We need other techniques for reasoning on the semantics of programs with Negation (stable model semantics).

Example

Let $P = p(a) \leftarrow \texttt{not}\ p(b)$ (it's the theory: $T = p(a) \vee p(b)$).
There are 4 Herbrand interpretations:

$$\begin{array}{ccccc}
 & & \{p(a), p(b)\} & & \\
 & \nearrow & & \nwarrow & \\
\{p(a)\} & & & & \{p(b)\} \\
 & \nwarrow & & \nearrow & \\
 & & \emptyset & &
\end{array}$$

3 of them are models. There is no A such that $T \models A$.

Moreover, $T_P(\emptyset) = \{p(a)\}$, $T_P(\{p(b)\}) = \emptyset$: this is not monotone.

We need other techniques for reasoning on the semantics of programs with Negation (stable model semantics).

SEMANTICS OF GENERAL PROGRAMS

WHAT IS THE EFFECT OF THE LOSS OF MONOTONICITY?

```
student(alberto).    student(bruno).     student(carlo).
course(fondamenti).      course(asd).

studied(alberto, fondamenti). studied(bruno, fondamenti).
studied(carlo,asd).

can_participate_exam(S,E) :-
        student(S), course(E),
        studied(S,E),
        not fail_selftest(S,E).

fail_selftest(S,E) :- test(S,E,VOTO), VOTO < 15.
```

What can we deduce? And if now knew that `test(carlo,asd,10).` And if now knew that `test(alberto,fondamenti,5).` ???

SEMANTICS OF GENERAL PROGRAMS

WHAT IS THE EFFECT OF THE LOSS OF MONOTONICITY?

```
student(alberto).   student(bruno).     student(carlo).
course(fondamenti).     course(asd).

studied(alberto, fondamenti). studied(bruno, fondamenti).
studied(carlo,asd).

can_participate_exam(S,E) :-
        student(S), course(E),
        studied(S,E),
        not fail_selftest(S,E).

fail_selftest(S,E) :- test(S,E,VOTO), VOTO < 15.
```

What can we deduce? And if now knew that `test(carlo,asd,10).` And if now knew that `test(alberto,fondamenti,5).` ???

SEMANTICS OF GENERAL PROGRAMS

WHAT IS THE EFFECT OF THE LOSS OF MONOTONICITY?

```
student(alberto).    student(bruno).      student(carlo).
course(fondamenti).        course(asd).

studied(alberto, fondamenti). studied(bruno, fondamenti).
studied(carlo,asd).

can_participate_exam(S,E) :-
        student(S), course(E),
        studied(S,E),
        not fail_selftest(S,E).

fail_selftest(S,E) :- test(S,E,VOTO), VOTO < 15.
```

What can we deduce? And if now knew that `test(carlo,asd,10).` And if now knew that `test(alberto,fondamenti,5).` ???

```
student(alberto).   student(bruno).     student(carlo).
course(fondamenti).     course(asd).

studied(alberto, fondamenti). studied(bruno, fondamenti).
studied(carlo,asd).

can_participate_exam(S,E) :-
        student(S), course(E),
        studied(S,E),
        not fail_selftest(S,E).

fail_selftest(S,E) :- test(S,E,VOTO), VOTO < 15.
```

What can we deduce? And if now knew that `test(carlo,asd,10).` And if now knew that `test(alberto,fondamenti,5).` ???

```
student(alberto).   student(bruno).     student(carlo).
course(fondamenti).     course(asd).

studied(alberto, fondamenti). studied(bruno, fondamenti).
studied(carlo,asd).

can_participate_exam(S,E) :-
        student(S), course(E),
        studied(S,E),
        not fail_selftest(S,E).

fail_selftest(S,E) :- test(S,E,VOTO), VOTO < 15.
```

What can we deduce? And if now knew that `test(carlo,asd,10).` And if now knew that `test(alberto,fondamenti,5).` ???

Given a program *P*:

```
r(a, c).
r(a, d).
q(X) :- r(X,Y), not s(Y).
p(a) :- not p(b).
p(b) :- not p(a).
```

it is normalized, obtaining *norm*(P):

```
r(X1,X2) :- X1=a, X2=c.
r(X1,X2) :- X1=a, X2=d.
q(X1) :- r(X1,Y), not s(Y).
p(X1) :- X1=a, not p(b).
p(X1) :- X1=b, not p(a).
```

Let us collect equal heads and add iff and explicit quantifiers, obtaining $iff(P)$:

$$\begin{array}{l}
\mathtt{r}(\mathtt{X}_1,\mathtt{X}_2) \leftrightarrow (\mathtt{X}_1{=}\mathtt{a} \wedge \mathtt{X}_2{=}\mathtt{c}) \vee (\mathtt{X}_1{=}\mathtt{a} \wedge \mathtt{X}_2{=}\mathtt{d})\\
\mathtt{q}(\mathtt{X}_1) \leftrightarrow \exists\, \mathtt{Y}\ (\mathtt{r}(\mathtt{X}_1,\mathtt{Y}) \wedge \neg\ \mathtt{s}(\mathtt{Y}))\\
\mathtt{p}(\mathtt{X}_1) \leftrightarrow (\mathtt{X}_1{=}\mathtt{a} \wedge \neg\ \mathtt{p}(\mathtt{b})) \vee (\mathtt{X}_1{=}\mathtt{b} \wedge \neg\ \mathtt{p}(\mathtt{a}))\\
\mathtt{s}(\mathtt{X}_1) \leftrightarrow \mathtt{false}
\end{array}$$

The completion of P is:

```
r(X1,X2) ↔ (X1=a ∧ X2=c) ∨ (X1=a ∧ X2=d)
q(X1) ↔ ∃ Y (r(X1,Y) ∧ ¬ s(Y))
p(X1) ↔ (X1=a ∧ ¬ p(b)) ∨ (X1=b ∧ ¬ p(a))
s(X1) ↔ false
```

(plus the so-called *freeness axioms*)

$$
\begin{array}{l}
\mathtt{r}(X_1,X_2) \leftrightarrow (X_1=\mathtt{a} \wedge X_2=\mathtt{c}) \vee (X_1=\mathtt{a} \wedge X_2=\mathtt{d}) \\
\mathtt{q}(X_1) \leftrightarrow \exists Y\ (\mathtt{r}(X_1,Y) \wedge \neg\ \mathtt{s}(Y)) \\
\mathtt{p}(X_1) \leftrightarrow (X_1=\mathtt{a} \wedge \neg\ \mathtt{p}(\mathtt{b})) \vee (X_1=\mathtt{b} \wedge \neg\ \mathtt{p}(\mathtt{a})) \\
\mathtt{s}(X_1) \leftrightarrow \mathtt{false}
\end{array}
$$

We need to consider 28 atoms:

s(a)	s(b)	s(c)	s(d)
p(a)	p(b)	p(c)	p(d)
q(a)	q(b)	q(c)	q(d)
r(a,a)	r(a,b)	r(a,c)	r(a,d)
r(b,a)	r(b,b)	r(b,c)	r(b,d)
r(c,a)	r(c,b)	r(c,c)	r(c,d)
r(d,a)	r(d,b)	r(d,c)	r(d,d)

Which of them are in all models of the completions?

Which one in no-one?

$$
\begin{array}{l}
\mathtt{r}(X_1,X_2) \leftrightarrow (X_1{=}a \wedge X_2{=}c) \vee (X_1{=}a \wedge X_2{=}d)\\
\mathtt{q}(X_1) \leftrightarrow \exists\, Y\ (\mathtt{r}(X_1,Y) \wedge \neg\ \mathtt{s}(Y))\\
\mathtt{p}(X_1) \leftrightarrow (X_1{=}a \wedge \neg\ \mathtt{p}(b)) \vee (X_1{=}b \wedge \neg\ \mathtt{p}(a))\\
\mathtt{s}(X_1) \leftrightarrow \mathtt{false}
\end{array}
$$

s(a) F	s(b) F	s(c) F	s(d) F
p(a)	p(b)	p(c)	p(d)
q(a)	q(b)	q(c)	q(d)
r(a,a)	r(a,b)	r(a,c)	r(a,d)
r(b,a)	r(b,b)	r(b,c)	r(b,d)
r(c,a)	r(c,b)	r(c,c)	r(c,d)
r(d,a)	r(d,b)	r(d,c)	r(d,d)

Let’s analyze s

$$
\begin{array}{l}
\texttt{r}(X_1,X_2) \leftrightarrow (X_1{=}\texttt{a} \wedge X_2{=}\texttt{c}) \vee (X_1{=}\texttt{a} \wedge X_2{=}\texttt{d})\\
\texttt{q}(X_1) \leftrightarrow \exists\, Y\ (\texttt{r}(X_1,Y) \wedge \neg\ \texttt{s}(Y))\\
\texttt{p}(X_1) \leftrightarrow (X_1{=}\texttt{a} \wedge \neg\ \texttt{p}(\texttt{b})) \vee (X_1{=}\texttt{b} \wedge \neg\ \texttt{p}(\texttt{a}))\\
\texttt{s}(X_1) \leftrightarrow \texttt{false}
\end{array}
$$

s(a) F	s(b) F	s(c) F	s(d) F
p(a)	p(b)	p(c)	p(d)
q(a)	q(b)	q(c)	q(d)
r(a,a) F	r(a,b) F	r(a,c) T	r(a,d) T
r(b,a) F	r(b,b) F	r(b,c) F	r(b,d) F
r(c,a) F	r(c,b) F	r(c,c) F	r(c,d) F
r(d,a) F	r(d,b) F	r(d,c) F	r(d,d) F

Let’s analyze r

$$\mathtt{r(X_1,X_2)} \leftrightarrow \mathtt{(X_1{=}a} \wedge \mathtt{X_2{=}c)} \vee \mathtt{(X_1{=}a} \wedge \mathtt{X_2{=}d)}$$
$$\mathtt{q(X_1)} \leftrightarrow \exists\, \mathtt{Y}\ \mathtt{(r(X_1,Y)} \wedge \neg\, \mathtt{s(Y))}$$
$$\mathtt{p(X_1)} \leftrightarrow \mathtt{(X_1{=}a} \wedge \neg\, \mathtt{p(b))} \vee \mathtt{(X_1{=}b} \wedge \neg\, \mathtt{p(a))}$$
$$\mathtt{s(X_1)} \leftrightarrow \mathtt{false}$$

s(a) F	s(b) F	s(c) F	s(d) F
p(a)	p(b)	p(c)	p(d)
q(a) T	q(b) F	q(c) F	q(d) F
r(a,a) F	r(a,b) F	r(a,c) T	r(a,d) T
r(b,a) F	r(b,b) F	r(b,c) F	r(b,d) F
r(c,a) F	r(c,b) F	r(c,c) F	r(c,d) F
r(d,a) F	r(d,b) F	r(d,c) F	r(d,d) F

Let’s analyze q

$$
\begin{array}{l}
\mathtt{r}(X_1,X_2) \leftrightarrow (X_1{=}\mathtt{a} \land X_2{=}\mathtt{c}) \lor (X_1{=}\mathtt{a} \land X_2{=}\mathtt{d}) \\
\mathtt{q}(X_1) \leftrightarrow \exists\, Y\ (\mathtt{r}(X_1,Y) \land \neg\ \mathtt{s}(Y)) \\
\mathtt{p}(X_1) \leftrightarrow (X_1{=}\mathtt{a} \land \neg\ \mathtt{p}(\mathtt{b})) \lor (X_1{=}\mathtt{b} \land \neg\ \mathtt{p}(\mathtt{a})) \\
\mathtt{s}(X_1) \leftrightarrow \mathtt{false}
\end{array}
$$

s(a) F	s(b) F	s(c) F	s(d) F
p(a) ?	p(b) ?	p(c) F	p(d) F
q(a) T	q(b) F	q(c) F	q(d) F
r(a,a) F	r(a,b) F	r(a,c) T	r(a,d) T
r(b,a) F	r(b,b) F	r(b,c) F	r(b,d) F
r(c,a) F	r(c,b) F	r(c,c) F	r(c,d) F
r(d,a) F	r(d,b) F	r(d,c) F	r(d,d) F

Let's analyze p

- Three sets of atoms emerge:
  - Those true in all Herbrand models of the completion
  - Those false in all Herbrand models of the completion
  - The others (true in some models, false in others models)
- This suggest a data structure storing the set I^+ (always true) and I^- (always false). This is sometimes called Fitting 3-valued semantics
- These sets can be computed using the notion of *well-founded* model.
- The well-founded model is a pair $\langle I^+, I^- \rangle$ which is unique and computable in polynomial time on the ground program.
- If $I^+ \cup I^- \neq B_P$ the well-founded model is not a real model!
- If it is a model it will be the unique stable model

- Three sets of atoms emerge:
  - Those true in all Herbrand models of the completion
  - Those false in all Herbrand models of the completion
  - The others (true in some models, false in others models)
- This suggest a data structure storing the set I^+ (always true) and I^- (always false). This is sometimes called Fitting 3-valued semantics
- These sets can be computed using the notion of *well-founded* model.
- The well-founded model is a pair $\langle I^+, I^- \rangle$ which is unique and computable in polynomial time on the ground program.
- If $I^+ \cup I^- \neq B_P$ the well-founded model is not a real model!
- If it is a model it will be the unique stable model

- We have already used the notion of ground program
- We will start our reasoning on the ground version of the program
- Given a general program P, $ground(P)$ is the set of all ground instances of P obtained replacing the variables in the clauses with all elements of $\mathcal{H}_P$
- Later we'll see the complexity of computing $ground(P)$.

$P =$

```
p(a).   p(b).   q(b).   q(c).
r(X,Y)  :- p(X),q(Y).
```

$ground(P) =$

```
p(a).   p(b).   q(b).   q(c).
r(a,a)  :- p(a),q(a).
r(a,b)  :- p(a),q(b).
r(a,c)  :- p(a),q(c).
r(b,a)  :- p(c),q(a).
r(b,b)  :- p(c),q(b).
r(b,c)  :- p(c),q(c).
r(c,a)  :- p(c),q(a).
r(c,b)  :- p(c),q(b).
r(c,c)  :- p(c),q(c).
```

- Let P be

```
p :- not q.
```

  We know that P has two minimal models: $\{p\}$ e $\{q\}$.

- $\{q\}$ does not capture the meaning “if you have no reasons for believing in q, then believe in p”). There are no info in P to justify that q is true.
- We would like to have as unique model $\{p\}$ (it is also the well-founded model).
- If P is:

```
p :- not q.   q :- not p.
```

  then well-founded simply states $I^+ = \emptyset$, $I^- = \emptyset$ (i.e., nothing)

- Given a general program P and a candidate model S, let us define P^S (the reduct of P w.r.t. S) as follows:
  1. remove every rule that contains a naf-literal `not` L in the body such that $L \in S$;
  2. remove every naf-literal from the bodies of the remaining rules.
- Let us observe that P^S is a definite program. We can compute its minimum model M_{P^S}. If $M_{P^S} = S$ then S is a stable model (a.k.a. answer set) for P.

Let us consider the program P

```
p :- a.
a :- not b.
b :- not a.
```

the candidate stable models for P are all the subsets of $\mathcal{B}_P = \{a, b, p\}$.

Let us consider the program P

```
p :- a.
a :- not b.
b :- not a.
```

the candidate stable models for P are all the subsets of $\mathcal{B}_P = \{a, b, p\}$.

$\emptyset$ We have that $P^{\emptyset} = \{p \leftarrow a.\ \ a.\ \ b.\}$. $\emptyset$ is not the minimum model of $P^{\emptyset}$. Thus $\emptyset$ is not an answer set of P.

Let us consider the program P

```
p :- a.
a :- not b.
b :- not a.
```

the candidate stable models for P are all the subsets of $\mathcal{B}_P = \{a, b, p\}$.

$\{a\}$ We have that $P^{\{a\}} = \{p \leftarrow a.\ \ a.\}$. $\{a\}$ is not the minimum model of $P^{\{a\}}$. Thus $\{a\}$ is not an answer set of P.

Let us consider the program P

```
p :- a.
a :- not b.
b :- not a.
```

the candidate stable models for P are all the subsets of $\mathcal{B}_P = \{a, b, p\}$.

$\{b\}$ We have that $P^{\{b\}} = \{p \leftarrow a.\ \ b.\}$. $\{b\}$ is the minimum model of $P^{\{b\}}$. Thus, $\{b\}$ is an answer set of P.

Let us consider the program P

```
p :- a.
a :- not b.
b :- not a.
```

the candidate stable models for P are all the subsets of $\mathcal{B}_P = \{a, b, p\}$.

$\{p\}$ We have that $P^{\{p\}} = \{p \leftarrow a.\ \ a.\ \ b.\}$. $\{p\}$ is not the minimum model of $P^{\{p\}}$. Thus, $\{p\}$ is not an answer set of P.

Let us consider the program P

```
p :- a.
a :- not b.
b :- not a.
```

the candidate stable models for P are all the subsets of $\mathcal{B}_P = \{a, b, p\}$.

$\{p, a\}$ We have that $P^{\{p,a\}} = \{p \leftarrow a. \ \ a.\}$. $\{p, a\}$ is the minimum model of $P^{\{p,a\}}$. Thus, $\{p, a\}$ is an answer set of$P$.

Let us consider the program P

```
p :- a.
a :- not b.
b :- not a.
```

the candidate stable models for P are all the subsets of $\mathcal{B}_P = \{a, b, p\}$.

$\{a, b\}$, $\{b, p\}$ e $\{a, b, p\}$ are not answer sets of P since they include properly answer sets (e.g., $\{b\}$) [This is a theorem. Answer Sets are always minimal]

Let us consider the program P

```
p :- a.
a :- not b.
b :- not a.
```

the candidate stable models for P are all the subsets of $\mathcal{B}_P = \{a, b, p\}$.

Thus P has two answer sets: $\{b\}$ and $\{p, a\}$.

Let us consider the program P

```
a :- not b.
b :- not c.
d.
```

The set $S_1 = \{b, d\}$ is an answer set of P. As a matter of fact, $P^{S_1} = \{b.\ \ d.\}$ that has S_1 as minimum model.
Instead, $S_2 = \{a, d\}$ is not an answer set of P: $P^{S_2} = \{a.\ \ b.\ \ d.\}$ has not S_2 as minimum model.

Let us consider the program P

```
p :- not p,  d.
d.
```

It admits the (logical) model $\{p, d\}$. Observe that any model of P must contain d. Thus we have two possible candidates for being answer sets:

- $S_1 = \{d\}$: then $P^{S_1} = \{p \leftarrow d.\ \ d.\}$. Its minimum model is not S_1.
- $S_2 = \{d, p\}$: then $P^{S_2} = \{\ d.\}$. Its minimum model is not S_2.

This program has logical models but it has not stable models!

Let us consider the program P

```
p :- not p,   d.
d.
```

It admits the (logical) model $\{p, d\}$. Observe that any model of P must contain d. Thus we have two possible candidates for being answer sets:

- $S_1 = \{d\}$: then $P^{S_1} = \{p \leftarrow d.\ \ d.\}$. Its minimum model is not S_1.
- $S_2 = \{d, p\}$: then $P^{S_2} = \{\ d.\}$. Its minimum model is not S_2.

This program has logical models but it has not stable models!

Semantics of General Programs

"Constraints"

- We should think to the body of an ASP rule as a justification for supporting the truth of its head.
- Intuitively, *"p is in the answer set only if it is supported by the fact that it is the head of a body which is true in the answer set. The only exception is that you cannot support p by the presence of* `not` *p in its body"*
- E.g.

```
p :- not p, d.
```

  does not support th etruth of *p*.
- (but *p* could be supported by another rule, in case)

Semantics of General Programs

"Constraints"

- From the answer set point of view, if `p` does not occur elsewhere in (head of rules of) the program

```
p :- not p, d.
```

  is equivalent to state that *d* must be false
- This can be simply stated by

```
:- d
```

  (called constraint)
- constraints are therefore syntactic sugar

Let us consider the following program:

```
a :- not n_a.   n_a :- not a.
b :- not n_b.   n_b :- not b.
c :- not n_c.   n_c :- not c.
d :- not n_d.   n_d :- not d.
```

Its answer sets are all (and only) the sets containing exactly one option between

- `a` and `n_a`,
- `b` and `n_b`,
- `c` and `n_c`,
- `d` and `n_d`.

Also for this case we have a syntactic sugar.

THEOREM

Given a ground program P, the problem of establishing whether it admits answer sets (stable models) is NP-complete.

NP Let P ground program, a candidate stable model S will contain only atoms occurring in P, thus $|S| \leq |P|$. Computing P^S, the fixpoint computation of M_{P^S}, and checking if $S = M_{P^S}$ therefore polynomial w.r.t. $|P|$. Thus the problem is in NP.

Theorem

Given a ground program P, the problem of establishing whether it admits answer sets (stable models) is NP-complete.

Hardness Let us consider an instance φ of 3SAT:

$$\underbrace{(A \vee \neg B \vee C)}_{\texttt{c1}} \wedge \underbrace{(\neg A \vee B \vee \neg C)}_{\texttt{c2}}$$

and define accordingly the program P_φ:

```
a  :- not na.   na :- not a.
b  :- not nb.   nb :- not b.
c  :- not nc.   nc :- not c.
c1 :- a.   c1 :- nb.   c1 :- c.
c2 :- na.   c2 :- b.   c2 :- nc.
:- not c1.   :- not c2.
```

P_φ can be computed in LOGSPACE and it is immediate to check that it admits a stable model iff φ is satisfiable.

- P definite program: unique minimal model M_P. If P is ground and finite you can compute it in PTIME.
- P general program. B_P is always a logical model, but it is not interesting.
- P general program.It admits a unique well-founded model. If P is ground and finite you can compute it in PTIME. If it is total it is also the unique stable model.
- If instead it is partial, and P is ground and finite establishing the existence of a stable model is NP complete.
- We have a programming paradigm exactly for the class NP.
- It is also useful for non monotonic reasoning.