Datalog and Answer Set Programming (part 4)

Riccardo Rosati

Knowledge Representation and Semantic Technologies
Master of Science in Engineering in Computer Science
Sapienza Università di Roma
a.a. 2020/2021

http://www.diag.uniroma1.it/rosati/krst/

SAPIENZA
UNIVERSITÀ DI ROMA

Datalog with negation

Rules with negation

Datalog rule with negation = expression of the form

$$\alpha :\!- \beta_1, \ldots, \beta_n, not\ \gamma_1, \ldots, not\ \gamma_m$$

where:

- n,m are non-negative integers
- α is an atom
- every β_i is an atom
- every γ_i is an atom (and is called negated atom)
- (**safeness** condition) every variable symbol occurring in the rule must appear in at least one of the positive atoms of the rule body $\beta_1, \ldots, \beta_n$

A **Datalog program with negation** is a set of Datalog rules with negation.

Examples of rules with negation

EDB = $\{p/1\}$, IDB = $\{q/0, r/2, s/2, t/1, u/3\}$

Const = $\{a, b, c\}$

Var = $\{X, Y, Z, W\}$

Correct rules with negation:

- $r(X,Y)$:– $r(Y,Z), s(Y,X), not\ r(X,Z)$.
- $r(a,b)$:– $not\ p(b,c)$.
- $t(X)$:– $r(X,Y), s(Y,Z), not\ r(X,Z), not\ s(Z,Y)$.
- $t(a)$:– $not\ r(b,c)$.
- $s(b,a)$:– $r(a,b), not\ q$.
- q :– $not\ p(a), not\ s(b,c)$.

Examples of rules with negation

Pred = $\{p/1, q/0, r/2, s/2, t/1, u/3\}$

Const = $\{a, b, c\}$

Var = $\{X, Y, Z, W\}$

Incorrect rules with negation:

- $t(X)$:– $not\ p(X)$.
  - variable X is not safe
- $t(Y)$:–$p(Y), not\ p(X)$.
  - variable X is not safe
- $t(W)$:– $r(X, Y), s(Y, Z), not\ s(Y, W)$.
  - variable W is not safe
- $not\ t(X)$:– $p(X)$.
  - cannot use not in the head atom

Semantics of rules with negation

A ground rule r α :– $\beta_1, \dots, \beta_n$, $not\ \gamma_1, \dots, not\ \gamma_m$ is **satisfied** in an interpretation I if at least one of the following conditions holds:

- at least one of the atoms $\beta_1, \dots, \beta_n$ does not belong to I
- at least one of the atoms $\gamma_1, \dots, \gamma_m$ belongs to I
- the atom α belongs to I

Examples:

1) the rule $r(a,b)$:– $not\ p(b,c)$.

- is satisfied in the interpretations $\{p(b,c)\}$, $\{r(a,b)\}$, $\{p(b,c), r(a,b)\}$
- is not satisfied in the interpretation $\{\ \}$

2) the rule $r(a,b)$:– $q(a), not\ p(b,c)$.

- is satisfied in the interpretations $\{\ \}$, $\{p(b,c)\}$, $\{r(a,b)\}$, $\{p(b,c), r(a,b)\}$, $\{p(b,c), q(a)\}$, $\{q(a), r(a,b)\}$, $\{p(b,c), q(a), r(a,b)\}$
- is not satisfied in the interpretation $\{q(a)\}$.

Semantics of programs with negation

An interpretation I is a **model** for a ground Datalog program with negation P if all the rules of P are satisfied in I.

To define models for non-ground programs with negation, we extend the notion of grounding to rules with negation in the obvious way:

The **grounding** of a Datalog **rule** with negation r with respect to a set of constants C, denoted as $ground(r, C)$, is the set of all ground rules with negation that can be obtained from r by replacing, for every variable x occurring in r, every occurrence of x with a constant from C.

The **grounding** of a non-ground Datalog **program** with negation P, $ground(P)$, is the ground Datalog program with negation obtained by the union of all the sets $ground\big(r, HU(P)\big)$ such that $r \in P$.

An interpretation I is a **model** for a non-ground Datalog program with negation P if I is a model for $ground(P)$.

Semantics of programs with negation

Following what is done for positive programs, the next step would be to consider only minimal models of a program with negation.

However, this step is problematic in the presence of negation:

1) First, differently from the positive case, **multiple** minimal models for a program with negation may exist:

e.g. (see previous examples), the program consisting of the single rule $r(a,b) :\!- \; not\; p(b,c).$ has two minimal models:

$I_1 = \{p(b,c)\}$

$I_2 = \{r(a,b)\}$

2) Moreover, some minimal models are **not "intended"** ones:

e.g. the above minimal model $I_1 = \{p(b,c)\}$ appears quite strange, since in the program there is no rule that allows for deriving $\{p(b,c)\}$ (i.e., that has $p(b,c)$ in the head of the rule).

Semantics of programs with negation

So, the declarative semantics based on a "classical" notion of model (and interpretation of negation) does not seem satisfactory.

Also, extending the operational semantics of positive programs to the presence of negation in rules seems problematic:

Operational semantics and negation

Example: let P be the program

$q(a) :- \ not \ q(a)$.

Let I be our starting empty interpretation.

The immediate consequence operator T_P applied to I would produce the interpretation I'={q(a)}, because q(a) is false in I, therefore not q(a) is true in I, and so q(a) is derived.

But if now we apply T_P to I', we obtain the empty interpretation: ideed, since not q(q) is true in I', not q(a) is now false in I', therefore there are no immediate consequences. So, we have obtained again the initial interpretation I.

It is immediate to conclude that **the iterated application of the T_P operator will never converge to a least fixpoint**.

Answer Set Semantics

We thus introduce a new declarative semantics for Datalog programs with negation, called **Answer Set Semantics**.

Let P be a program with negation and let I be an interpretation. The **reduct** of P with respect to I, denoted as P/I, is the positive ground program obtained as follows:

Delete from ground(P) every rule R such that an atom not β occurs in the body of R and β belongs to I;

Delete from every rule R in ground(P) every atom not β occurring in the body of R such that β does not belong to I.

Let P be a ground Datalog program with negation. An interpretation I is an **answer set** of P if I is the minimal model of the positive program P/I.

Reduct of a program: Example

Example: let P be the program

$r(a)$:– $p(a), not\ q(a)$.

$s(a)$:– $not\ t(a)$.

$t(a)$:– $r(a), not\ p(a)$.

$p(a)$.

1) Let I be the interpretation $\{p(a), s(a)\}$. Then, P/I is the positive program

$r(a)$:– $p(a)$.

$s(a)$.

$p(a)$.

2) Let I' be the interpretation $\{p(a), t(a)\}$. Then, P/I' is the positive program

$r(a)$:– $p(a)$.

$p(a)$.

Reduct of a program: Example

Example (contd.): let P be the program

$r(a)$:– $p(a), not\ q(a)$.

$s(a)$:– $not\ t(a)$.

$t(a)$:– $r(a), not\ p(a)$.

$p(a)$.

3) Finally, let I'' be the interpretation $\{p(a), r(a), s(a)\}$. Then, P/I'' is the positive program

$r(a)$:– $p(a)$.

$s(a)$.

$p(a)$.

Answer Sets: Example

Let P be a ground Datalog program with negation. An interpretation I is an **answer set** of P if I is the minimal model of the positive program P/I.

Example (contd.):

- the minimal model of program P/I is $\{p(a), r(a), s(a)\}$, therefore I is **not** an answer set of P
- the minimal model of program P/I' is $\{p(a), r(a)\}$, therefore I' is **not** an answer set of P
- the minimal model of program P/I'' is $\{p(a), r(a), s(a)\}$, therefore I'' is an answer set of P

Properties of Answer Sets

Property: Every answer set is a minimal model of P, but not vice versa.

Example:

Let P be the program consisting of the single rule $r(a,b)$:– $not\ p(b,c)$.

P has two minimal models:

$I_1 = \{p(b,c)\}$

$I_2 = \{r(a,b)\}$

However, only I_2 is an answer set of P, since:

- P/I_1 is the empty program, whose minimal model is the empty set
- P/I_2 is the program $r(a,b)$. whose minimal model is I_2

Properties of Answer Sets

Property: A program with negation may have 0, 1 or multiple answer sets.

Examples:

1) Let P be the program consisting of the single rule $p(a)$:– $not\ p(a)$.

Then, P has no answer sets.

2) Let P be the program

$p(a)$:– $not\ q(a)$.

$q(a)$:– $not\ p(a)$.

Then, P has two answer sets:

$I_1 = \{p(a)\}$

$I_2 = \{q(a)\}$

Reasoning over programs with negation

Basic reasoning tasks:

- decide whether P has at least one answer set
- compute all the answer sets of P

Derived reasoning tasks:

- **Skeptical entailment**: given a program P and a ground atom α, establish whether α belongs to all the answer sets of P.
- **Credulous entailment**: given a program P and a ground atom α, establish whether α belongs to at least one answer set of P.

Reasoning technique

Basic reasoning technique that computes all the answer sets of P:

```
AS= {};
for every interpretation I over HB(P) do
  if I == MM(P/I)
  then add I to AS;
return AS;
```

(Checking whether I==MM(P/I) can be done through the naive or semi-naive evaluation of positive programs)

The computational cost of the above algorithm is **exponential**, even if we start from a ground program P.

(The reason is that there is an exponential number of subsets of HB(P) with respect to the size of HB(P), which is proportional to the size of ground(P))