Datalog and Answer Set Programming

Riccardo Rosati

Knowledge Representation and Semantic Technologies
Master of Science in Engineering in Computer Science
Sapienza Università di Roma
a.a. 2021/2022

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

Outline

- Positive Datalog
- Positive Datalog with constraints
- Datalog with negation
- Datalog with stratified negation
- Answer Set Programming (Datalog with negation and disjunction)

Positive Datalog: syntax

Datalog: alphabets

We start from the following alphabets (sets of symbols):

- Alphabet of predicate symbols **Pred**
  - every predicate symbol is associated with an **arity** (non-negative integer) (e.g. $p/2, r/1, s/0, \ldots$)
- Alphabet of constant symbols **Const**
  - Syntactic convention: we will use symbols starting with lowercase letters (e.g. $a, b, c, \ldots$)
- Alphabet of variable symbols **Var**
  - Syntactic convention: we will use symbols starting with uppercase letters (e.g. $X, Y, Z, \ldots$)

The three alphabets are pairwise disjoint (every symbol belongs at most to one alphabet)

We also define the (derived) set of terms **Term = Const** ∪ **Var**

Datalog: atoms

atom = expression of the form

$$p(t_1, \ldots, t_n)$$

where:

- n is a non-negative integer
- p is a predicate symbol of arity n
- every t_i is a term

Datalog: positive rules

Positive rule = expression of the form

$\alpha :- \beta_1, \ldots, \beta_n$.

where:

- n is a non-negative integer
- α is an atom
- every β_i is an atom
- (**safeness** condition) every variable symbol occurring in α must appear in $\beta_1, \ldots, \beta_n$

α is called the rule **head**

$\beta_1, \ldots, \beta_n$ is called the rule **body**

Examples of positive rules

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

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

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

Correct positive rules:

- $p(X)$:– $r(X, Y), s(Y, Z)$.
- q :– $r(X, Y), s(Y, Z)$.
- $r(X, Z)$:– $r(X, Y), s(Y, Z)$.
- $u(X, Y, Z)$:– $r(X, Y), s(Y, Z)$.
- $u(X, a, b)$:– $r(X, Y)$.
- $u(c, a, b)$:– .

Examples of positive rules

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

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

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

Incorrect positive rules:

- $p(\underline{W})$:– $r(X, Y), s(Y, Z)$.
  - variable W is not safe
- q :– $r(X, Y), s(Y, Z)$
  - missing "." at end of rule !!!
- $t(X, Y, \underline{r})$:– $r(X, Y), s(Y, Z)$.
  - uses a predicate in argument position
- $\underline{Y}(X, a, b)$:– $r(X, Y)$.
  - uses a variable in predicate position

Facts

A **fact** is a positve rule with an empty body

E.g.:

- $u(c, a, b)$:– .
- $p(a)$:– .
- q :– .

When representing facts we omit the "if" symbol, i.e.:

- $u(c, a, b)$.
- $p(a)$.
- q.

Notice that variable symbols cannot appear in a fact (due to the safeness condition)

Positive Datalog

Positive Datalog program = set of Positive datalog rules

Example:

$$
\begin{aligned}
&r(X,Z) \coloneq p(X,Y), p(Y,Z).\\
&s(X,Y) \coloneq r(Y,X).\\
&t(X,Y) \coloneq q(X), r(X,Y), s(Y,Z).\\
&p(a,b).\\
&p(b,c).\\
&q(b).
\end{aligned}
$$

EDB and IDB predicates

Predicates in a Datalog program are actually partitioned into **EDB** (Extensional Database) predicates and **IDB** (Intensional Database) predicates:

- EDB predicates can only be used in facts and in the body of rules
- IDB predicates cannot be used in facts

E.g.: EDB= $\{p/2, q/1\}$, IDB= $\{r/2, s/2\}$

$p(a,b).$
$p(b,c).$
$q(b).$
$r(X,Y) :- p(X,Y).$
$r(X,Z) :- p(X,Y), r(Y,Z).$
$s(X,Z) :- r(X,Y), q(Y).$

IDB

Recursive rules

A Datalog rule is **recursive** if the predicate occurring in its head also occurs in its body.

- EDB predicates can only be used in facts and in the body of rules
- IDB predicates cannot be used in facts

E.g., the following are recursive rules:

$r(X,Z) :- p(X,Y), r(Y,Z).$

$s(X,Y) :- s(Y,X).$

Ground rules and programs

A Datalog rule is **ground** if no variable occurs in it

Examples of ground rules:

$r(a,b) :- p(b,c).$

$s(c,d) :- t, r(a,a).$

$t :- r(a,c), s(b,c), q(a).$

$p(a,b).$

$q(b).$

(of course, every fact is a ground rule)

A Datalog program is **ground** if all its rules are ground

Positive Datalog: semantics

Herbrand Base

Given a Datalog program P, the **Herbrand Universe** of P (denoted by HU(P)) is the set of constant symbols occurring in P

Given a ground Datalog program P, the **Herbrand Base** of P (denoted by HB(P)) is the set of ground atoms occurring in P

E.g., let P_1 be the following ground program:

$$r(a,b) \mathrel{:-} p(b,a).$$
$$s(c,d) \mathrel{:-} t, r(a,a).$$
$$t \mathrel{:-} r(a,c), s(b,c), q(a).$$
$$p(b,a).$$
$$q(b).$$

Then, $HB(P_1) = \{r(a,b), p(b,a), s(c,d), t, r(a,a), r(a,c), s(b,c), q(a), q(b)\}$

Interpretations of ground programs

Given a ground Datalog program P, an **interpretation** for P is a subset of HB(P).

E.g., the following are possible interpretations for the ground program P_1 of the previous example:

$I_1 = \{r(a,a), p(b,a)\}$

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

$I_3 = \{r(a,b), p(b,a), s(c,d), t, r(a,a), r(a,c), s(b,c), q(a), q(b)\}$

$I_4 = \{\ \}$

$I_5 = \{p(b,a), q(b), r(a,b)\}$.

Models for ground programs

A ground positive rule r is **satisfied** in an interpretation I if:

either some atom in the body of r does not belong to I

or the atom in the head of r belongs to I

Examples:

- the rule $r(a, b) :- p(b, a).$ is satisfied in the interpretations I_2 , I_3 , I_4 , I_5 , but not in I_1.
- the rule (fact) $q(b).$ is satisfied in the interpretations I_2 , I_3 , I_5 , but not in I_1, I_4

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

Models for ground programs

Example: let P be the program

$r(a)$:– $p(a)$.

$r(b)$:– $q(b)$.

$p(a)$.

then, the following interpretations are models for P:

$\{p(a), r(a)\}$

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

while the following are NOT models for P:

$\{\ \}$

$\{p(a)\}$

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

Herbrand Base of non-ground programs

Herbrand Base of a **non-ground** program P = HB(P) = set of all the ground atoms that can be built with the predicates and the constants occurring in P

E.g., if P is the program

$$r(X,Y) \mathrel{:-} p(X,Y).$$
$$s(X,Y) \mathrel{:-} r(X,Z), s(Z,Y).$$
$$p(b,a).$$
$$q(b).$$

Then HB(P) =

$$\left\{ \begin{matrix} p(a,a), p(a,b), p(b,a), p(b,b), q(a), q(b), r(a,a), r(a,b), r(b,a), \\ r(b,b), s(a,a), s(a,b), s(b,a), s(b,b) \end{matrix} \right\}$$

Interpretations

Given a non-ground Datalog program P, an **interpretation** for P is a subset of HB(P).

E.g. for the previous program P, some interpretations for P are:

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

Grounding

A **variable instantiation** for a rule r in a program P is a function µ: $Vars(r) \rightarrow HU(P)$, i.e. a function mapping every variable symbol occurring in r to a constant symbol in $HU(P)$.

Given a variable instantiation µ for r in P, we denote as $\mu(r)$ the ground rule obtained from r by replacing every variable X occurring in r with $\mu(X)$. We call $\mu(r)$ an **instantiation** of rule r in P.

The **grounding** of a **rule** r in P, denoted as $ground(r, P)$, is the set of all the possible instantiations $\mu(r)$ of r in P.

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

Models

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

Minimal models

An interpretation I is a **minimal model** for a (non-ground) Datalog program P, if I is a model for P and there exists no model I' for P such that I' is a strict subset of I.

Property: every positive Datalog program P has exactly one minimal model (we denote such a model by MM(P)).

Minimal model: example 1

Example: let P be the program

$r(a)$:– $p(a)$.

$r(b)$:– $q(b)$.

$p(a)$.

the following interpretations are the models for P:

$$\left.\begin{array}{l}\{p(a), r(a)\}\\ \{p(a), r(a), q(b), r(b)\}\end{array}\right\} \ ?$$

So, the minimal model for P is:

$\{p(a), r(a)\}$

Minimal model: example 2

Example: let P be the program

$r(X,Y)$:– $p(X,Y)$.

$s(X,Y)$:– $r(X,Z), s(Z,Y)$.

$p(b,a)$.

$q(b)$.

HB(P) =

$$\left\{\begin{matrix} p(a,a), p(a,b), p(b,a), p(b,b), q(a), q(b), r(a,a), r(a,b), r(b,a), \\ r(b,b), s(a,a), s(a,b), s(b,a), s(b,b) \end{matrix}\right\}$$

Minimal model: example 2 (contd.)

The following interpretations are models for P:

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

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

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

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

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

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

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

etc…

The minimal model for P is:

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

Positive Datalog: reasoning

Reasoning in positive Datalog

Basic reasoning task: construction of MM(P)

Derived reasoning task: ground atom entailment

- Given a Datalog program P and a ground atom α, we say that P **entails** α if $\alpha \in$ MM(P) ✓

Immediate consequence operator

Given a ground positive Datalog program P, the **immediate consequence operator for P**, denoted as T_P, is the function over the domain of interpretations for P defined as follows:

$$T_P(I) = \{ \alpha \mid \text{there exists a rule } \alpha \text{ :– } \underline{\beta_1, \ldots, \beta_n}. \text{ in } P \text{ such that } \underline{\{\beta_1, \ldots, \beta_n\}} \subseteq I \}$$

The **least fixed point** of the function T_P is the minimal interpretation I such that $T_P(I) = I$.

Immediate consequence operator

Example: let P be the following ground program:

$r(a) \coloneq p(a).$
$r(b) \coloneq q(b).$
$q(b) \coloneq r(b).$
$p(a).$

Then:

$T_P(\{\,\}) = \{\, p(a) \,\}$
$T_P(\{\, p(a) \,\}) = \{\, p(a), r(a) \,\}$
$T_P(\{\, p(a), r(a) \,\}) = \{\, p(a), r(a) \,\}$ (least fixed point)
$T_P(\{\, q(b)\}) = \{\, p(a), r(b) \,\}$
$T_P(\{\, p(a), r(b)\}) = \{\, p(a), r(a), q(b) \,\}$
$T_P(\{\, p(a), r(a), q(b) \,\}) = \{\, p(a), r(a), r(b) \,\}$
$T_P(\{\, p(a), r(a), r(b) \,\}) = \{\, p(a), r(a), q(b) \,\}$
$T_P(\{\, p(a), q(b), r(a), r(b) \,\}) = \{\, p(a), q(b), r(a), r(b) \,\}$ (fixed point)

Operational semantics

Property: For every ground positive Datalog program P, the immediate consequence operator T_P has a unique least fixed point that coincides with MM(P). ✓

The above property provides an **operational semantics** for ground positive Datalog program.

In fact, from the previous property we can immediately derive the following algorithm for the computation of the miminal model of a ground positive program P.

Naive evaluation

```
Algorithm naive-evaluation
Input: ground positive Datalog program P
Output: MM(P)
begin
  let I'={};
  repeat
    let I=I';
    compute I'=$T_P$(I)
  until I'==I;
  return I
end
```

This algorithm executes at most k+1 iterations of the repeat-until loop, where k is the number of rules of P

Reasoning over non-ground programs

The naive evaluation algorithm could be used also for non-ground programs:

1. First, compute the ground program ground(P);
2. Then, execute the naive evaluation algorithm on ground(P).

The interpretation returned by the algorithm is the minimal model of ground(P) and therefore the minimal model of P.

We now present the **semi-naive evaluation** algorithm, which optimizes the naive evaluation.

Delta predicates

Idea of the optimization: reformulate the program P in a way such that the immediate consequence operator T_P can derive a ground atom only if its derivation depends on at least one ground atom derived in the previous application of T_P in the algorithm.

This minimizes redundant derivations, i.e., repeated derivations of the same ground atoms in different applications of T_P.

This idea is realized by introducing **delta** versions of the IDB predicates of the program, and rewriting the program through the usage of such delta predicates.

Delta predicates

In the algorithm, the extension of a Δ-predicate Δp represents the ground atoms relative to predicate p derived by the **previous** application of the immediate consequence operator.

A second set of Δ'-predicates is used in rule heads: in this way, each Δ'p represents the ground atoms relative to predicate p derived by the **current** application of the immediate consequence operator.

Δ-transformation of a rule

In the following, we denote a rule r of a positive program P as follows:

$$\alpha :- \beta_1, \dots, \beta_k, \gamma_1, \dots, \gamma_h.$$

where every β_i is an IDB-atom (i.e. an atom in which an IDB predicate of P occurs), and every γ_i is an EDB-atom

Given a rule r of the above form, Δr is the following set of k rules:

$$\Delta'\alpha :- \Delta\beta_1, \beta_2, \dots, \beta_k, \gamma_1, \dots, \gamma_h.$$
$$\Delta'\alpha :- \beta_1, \Delta\beta_2, \dots, \beta_k, \gamma_1, \dots, \gamma_h.$$
$$\dots$$
$$\Delta'\alpha :- \beta_1, \beta_2, \dots, \Delta\beta_k, \gamma_1, \dots, \gamma_h.$$

where:

- if $\beta_i = r(t_1, \dots, tn)$, then $\Delta\beta_i$ denotes the atom $\Delta r(t_1, \dots, tn)$
- if $\alpha = r(t_1, \dots, tn)$, then $\Delta'\alpha$ denotes the atom $\Delta' r(t_1, \dots, tn)$

Δ-transformation of a rule

Therefore, if the body of r contains k atoms with IDB predicates, Δr contains k rules (for every such atom there is a rule where the delta predicate is used in such an atom instead of the standard predicate)

Δ-transformation of a rule

Example: let IDB = $\{r/1, s/2\}$, EDB = $\{p/2\}$, and let P be as follows:

$r(X)$:– $s(X,Y), p(X,Y)$. [r1]

$s(X,Y)$:– $p(X,Y), p(Y,Z)$. [r2]

$s(X,Y)$:– $s(X,Z), r(Y), p(Y,X)$. [r3]

$p(a,b)$. [r4]

Then:

Δr1 = { $\Delta' r(X)$:– $\Delta s(X,Y), p(X,Y)$. }

Δr2 = { }

Δr3 = { $\Delta' s(X,Y)$:– $\Delta s(X,Z), r(Y), p(Y,X)$.

$\Delta' s(X,Y)$:– $s(X,Z), \Delta r(Y), p(Y,X)$. }

?

Δr4 = { }

Δ-transformation of a program

Given a positive Datalog program P, the Δ-transformation of P, denoted as ΔP, is the program obtained as the union of all the sets Δr of every rule r in P.

Example (contd.):

$$\Delta P = \Delta r1 \cup \Delta r2 \cup \Delta r3 \cup \Delta r4 =$$

$$= \{ \Delta' r(X) :\!- \Delta s(X,Y), p(X,Y).$$

$$\Delta' s(X,Y) :\!- \Delta s(X,Z), r(Y), p(Y,X).$$

$$\Delta' s(X,Y) :\!- s(X,Z), \Delta r(Y), p(Y,X). \}$$

The algorithm then computes the immediate consequence operator $T_{\Delta P}$ of ΔP, treating both Δ-predicates and Δ′-predicates like all other standard predicates.

Semi-naive evaluation

```
Algorithm semi-naive-evaluation
Input: positive Datalog program P
Output: MM(P)
begin
  let I=EDB(P); //(EDB(P) is the set of facts in P)
  compute I'=$T_P$(I);
  if I==I' then return I;
  $\Delta'$I = { $\Delta'\alpha \mid \alpha \in$ I' − I };
  repeat
     let I = I ∪ { $\alpha \mid \Delta'\alpha \in \Delta'$I };
     let $\Delta$I = { $\Delta\alpha \mid \Delta'\alpha \in \Delta'$I };
     let $\Delta'$I = $T_{\Delta P}$(I ∪ $\Delta$I)
  until $\Delta'$I=={};
  return I
end
```

Example of execution of the algorithm

Let P be the following positive Datalog program:

$$r(X,Y) :- p(X,Y).$$
$$r(X,Z) :- p(X,Y), r(Y,Z).$$
$$s(X,Y) :- r(Y,X).$$
$$p(a,b).$$
$$p(b,c).$$
$$p(c,d).$$

- The initial value of I is the set of facts of P, i.e. $\{ p(a,b), p(b,c), p(c,d) \}$
- Then, I'=T$_P$(I)= I ∪ $\{ r(a,b), r(b,c), r(c,d) \}$ (applying the first rule)
- So, Δ'I = $\{\Delta' r(a,b),\ \Delta' r(b,c),\ \Delta' r(c,d) \}$
- Now notice that ΔP is the following program:

$$\Delta' r(X,Z) :- p(X,Y), \Delta r(Y,Z). \quad [r1]$$
$$\Delta' s(X,Y) :- \Delta r(Y,X). \quad [r2]$$

Example (contd.)

- Then, the algorithm executes the repeat-until loop for the first time, obtaining:

$$I = I \cup \{\, r(a,b), r(b,c), r(c,d) \,\}$$

$$\Delta I = \{\, \Delta r(a,b), \Delta r(b,c), \Delta r(c,d) \,\}$$

$$\Delta' I = T_{\Delta P}(I \cup \Delta I) =$$

$$= \{\, \Delta' r(a,c), \Delta' r(b,d), \Delta' s(b,a), \Delta' s(c,b), \Delta' s(d,c) \,\}$$

(applying rules r1 and r2 of ΔP)

- Then, the algorithm executes the repeat-until loop for the second time, obtaining:

$$I = I \cup \{\, r(a,c), r(b,d), s(b,a), s(c,b), s(d,c) \,\}$$

$$\Delta I = \{\, \Delta r(a,c), \Delta r(b,d), \Delta s(b,a), \Delta s(c,b), \Delta s(d,c) \,\}$$

$$\Delta' I = T_{\Delta P}(I \cup \Delta I) =$$

$$= \{\, \Delta' r(a,d), \Delta' s(c,a), \Delta' s(d,b) \,\}$$ (applying r1 and r2)

Example (contd.)

- Then, the algorithm executes the repeat-until loop for the third time, obtaining:

$$I = I \cup \{ r(a,d), s(c,a), s(d,b) \}$$
$$\Delta I = \{ \Delta r(a,d), \Delta s(c,a), \Delta s(d,b) \}$$
$$\Delta' I = T_{\Delta P}(I \cup \Delta I) = \{ \Delta' s(d,a) \} \text{ (applying r2)}$$

- Then, the algorithm executes the repeat-until loop for the fourth time, obtaining:

$$I = I \cup \{ s(d,a) \}$$
$$\Delta I = \{ \Delta s(d,a) \}$$
$$\Delta' I = T_{\Delta P}(I \cup \Delta I) = \{ \}$$

Example (contd.)

- Since Δ'`I` is empty, the algorithm exits the repeat-until loop and terminates returning the interpretation I, that is, the set

$$\{\, p(a,b), p(b,c), p(c,d), r(a,b), r(b,c), r(c,d), r(a,c), r(b,d), \\ s(b,a), s(c,b), s(d,c), r(a,d), s(c,a), s(d,b), s(d,a) \,\}$$

Such an interpretation I is the minimal model of P.

Positive Datalog with constraints

Constraints

A **constraint** is a new kind of rule of the form:

$$:\!-\ \beta_1, \ldots, \beta_n.$$

where:

- n is a positive integer
- every β_i is an atom

That is, a constraint is a rule with an empty head.

Examples of constraints:

$:\!-\ p(X, Y), r(Y, Z).$

$:\!-\ p(a, b), q(b).$

$:\!-\ s(Y, X).$

A constraint is **ground** if it does not contain occurrences of variables.

Datalog programs with constraints

A **Datalog program with constraints** is a set of positive rules and constraints.

Semantics of constraints

A ground constraint :– $\beta_1, \ldots, \beta_n$. is **satisfied** in an interpretation I if at least one of its atoms β_i does **not** belong to I.

(Remark: The notion of grounding of a program P, $ground(P)$, is applicable also in the case when P contains constraints)

Semantics of programs with constraints

An interpretation I is a **model** for a Datalog program with constraints P if every ground rule and every ground constraint in $ground(P)$ is satisfied in I.

(analogous definition of minimal model as in the case of positive Datalog)

Property: every positive Datalog program with constraints P has either no models (and hence no minimal models) or exactly one minimal model (and in the latter case we denote such a model by MM(P)).

Reasoning over programs with constraints

The techniques for reasoning with positive Datalog programs (naive or semi-naive evaluation) can be easily extended to the presence of contraints in the program.

Let P be a Datalog program with constraints. Then:

- Let P' be the program obtained from P eliminating all the constraints;
- Compute (using e.g. the naive or semi-naive evaluation) MM(P'), the minimal model of P';
- Check whether every constraint in P is satisfied in MM(P'):

  if this is the case, then MM(P') is the minimal model of P;

  otherwise P has no models (and hence no minimal models).

(Of course, every constraint in P is satisfied in MM(P') iff every ground constraint in $ground(P)$ is satisfied in MM(P'))

Example

Let P be the following program with constraints:

$r(X,Y) :- p(X,Y).$
$r(X,Z) :- p(X,Y), r(Y,Z).$
$s(X,Y) :- r(Y,X).$
$:- p(X,X).$ [c1]
$:- r(X,Y), r(Y,X).$ [c2]
$p(a,b).$
$p(b,c).$
$p(c,d).$

First, we notice that the program P’ obtained from P eliminating the two constraints c1 and c2 corresponds to the previous program for which we have computed the minimal model through the semi-naive computation.

Example

So, the program P’ has the following minimal model MM(P’):

$$\{\, p(a,b), p(b,c), p(c,d), r(a,b), r(b,c), r(c,d), r(a,c), r(b,d),\\ s(b,a), s(c,b), s(d,c), r(a,d), s(c,a), s(d,b), s(d,a) \,\}$$

Now we have to check whether the constraints c1 and c2 are satisfied in MM(P’). We have:

$ground(c1, HU(P))$ =

$:-\ p(a,a).$

$:-\ p(b,b).$

$:-\ p(c,c).$

$:-\ p(d,d).$

Example

$ground(c2, HU(P)) =$

$$
\begin{aligned}
&:\!- \ r(a,a), r(a,a).\\
&:\!- \ r(a,b), r(b,a).\\
&:\!- \ r(a,c), r(c,a).\\
&:\!- \ r(a,d), r(d,a).\\
&:\!- \ r(b,a), r(a,b).\\
&:\!- \ r(b,b), r(b,b).\\
&:\!- \ r(b,c), r(c,b).\\
&:\!- \ r(b,d), r(d,b).\\
&:\!- \ r(c,a), r(a,c).\\
&:\!- \ r(c,b), r(b,c).\\
&:\!- \ r(c,c), r(c,c).\\
&:\!- \ r(c,d), r(d,c).\\
&:\!- \ r(d,a), r(a,d).\\
&:\!- \ r(d,b), r(b,d).\\
&:\!- \ r(d,c), r(c,d).\\
&:\!- \ r(d,d), r(d,d).
\end{aligned}
$$

Example

It is easy to verify that every ground constraint in $ground(c1, HU(P))$ is satisfied in MM(P’), and that every ground constraint in $ground(c2, HU(P))$ is satisfied in MM(P’).

Consequently, MM(P’) is the minimal model of P.

Example

Now let P’’ be the program obtained from P adding the following constraint:

$:- r(X,Y), r(Y,W), r(W,Z), s(Z,X).$ [c3]

Among others, the following ground constraint belongs to $ground\big(c3, HU(P)\big)$:

$:- r(a,b), r(b,c), r(c,d), s(d,a).$

This ground constraint in NOT satisfied in MM(P’) (because all the atoms in the body of the constraints belong to MM(P’)), therefore c3 is non satisfied in MM(P’). Consequently, P’’ has no models (and hence no minimal models).

Datalog with negation

Rules with negation

Datalog rule with negation = expression of the form

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

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 $\beta_1, \dots, \beta_n$

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.

(Remark: The notion of grounding of a program P, $ground(P)$, is applicable also in the case when P is a program with negation)

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 $r(a,b) \text{ :– } not\ p(b,c).$ 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., there is no rule having $p(b,c)$ in its head).

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

$q(a) :- \mathit{not}\ q(a)$.

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 is the positive ground program obtained as follows:

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

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

Let P be a 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

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 **not** 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 $p(a) \text{ :– } not\ p(a)$.

Then, P has no answer sets.

2) Let P be the program

$p(a) \text{ :– } not\ q(a)$.

$q(a) \text{ :– } not\ p(a)$.

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;
```

→ the . . . Case

Alg

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

The computational cost 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))

Datalog with stratified negation

Non-recursive use of negation

As previously illustrated, the addition of negation to Datalog creates both semantic and computational issues.

The problematic examples presented above suggest that the use of **negation** creates problems when it is used in combination with **recursion**.

To overcome such problems, we define a subclass of programs with a non-recursive form of negation, called programs with **stratified negation**.

Labeled dependency graph

Let P be a Datalog program with negation. The **labeled dependency graph** of P is a graph G=(V,E,L) where V is the set of vertices, E is the set of edges, and L is a labeling of the edges in E, defined as follows:

- There is one vertex v in V for every IDB predicate in P;
- There is an edge (s,t) in E (without label) if s,t are IDB predicates and P contains a rule R such that t appears in the head of R and s appears in a positive atom in the body of R;
- There is a **negated** edge, i.e. an edge with label "-", (s,t) in E if s,t are predicates and P contains a rule R such that t appears in the head of R and s appears in a negative atom in the body of R.

Labeled dependency graph: Example

Example: let P be the following Datalog program with negation:

$r(X,Y) \mathrel{:-} p(X,Y),\ not\ p(Y,X).$

$r(X,Y) \mathrel{:-} p(X,Y),\ r(Y,Z).$

$s(X,Y) \mathrel{:-} r(Z,X),\ r(Z,Y),\ not\ r(X,Y).$

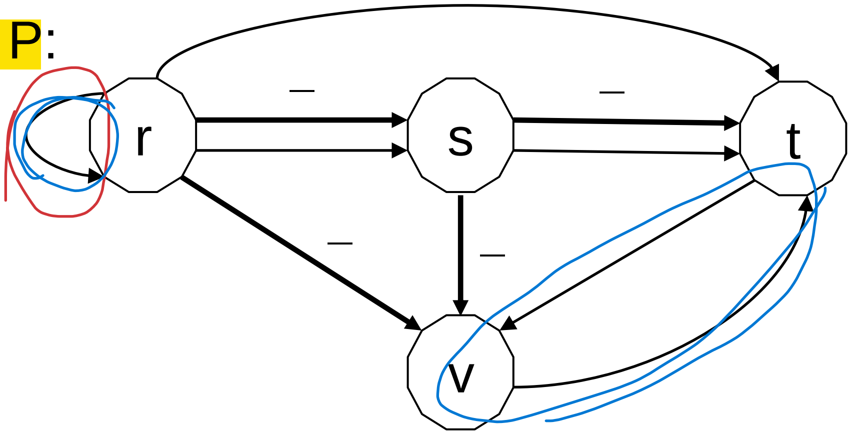
$t(X) \mathrel{:-} r(Y,X),\ s(X,X),\ not\ s(Y,Y).$

$t(X) \mathrel{:-} v(X,Y).$

$v(X,Y) \mathrel{:-} t(X),\ t(Y),\ not\ r(X,Y),\ not\ s(X,Y).$

$p(a,b).\ p(b,c).$

Labeled dependency graph G of P:



Programs with stratified negation

We say that a Datalog program with negation P is **stratified** if no cycle of the labeled dependency graph of P contains a negated edge.

Example (contd.):

The labeled dependency graph G has no cycle that contains a negated edge: the only cycles are: 1) the single edge (r,r) which is a positive edge; 2) the path (t,v), (v,t) which is only made of positive edges.

Consequently, P is a stratified program (or, equivalently, a program with stratified negation).

Operational semantics

For programs with stratified negation, the problems with the operational semantics (i.e. the iterated application of the immediate consequence operator) previously described do not arise.

Property: every stratified program P has a **unique** answer set, which coincides with the **unique** minimal model of P, and is denoted by MM(P).

It is possible to identify an algorithm that **deterministically** converges to the unique minimal model of the program with stratified negation.

Stratification of a program

A vertex v of G has a **negative dependency** if there exists a vertex v’ in G such that there is a path from v’ to v passing through a negated edge

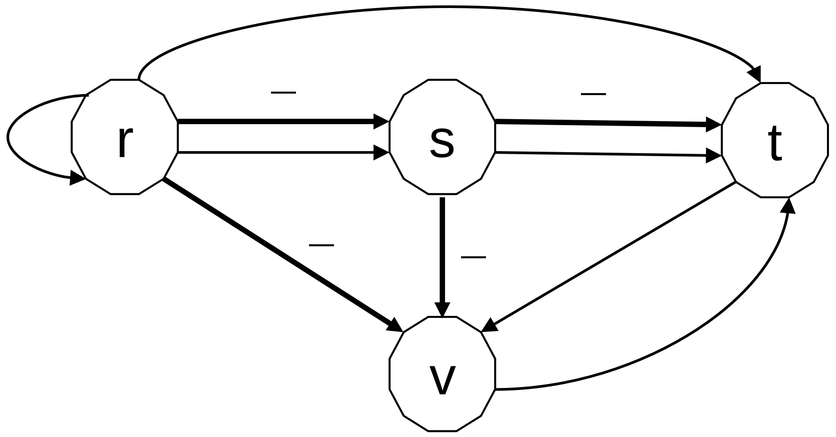
The **stratification** of a stratified program P is a sequence $S_1, \ldots, S_k$ of sets of IDB predicates of P (called **strata**) obtained as follows:

```
G = labeled dependency graph of P;
i=0;
while G is not empty do begin
   i=i+1;
   $S_i$ = the set of vertices of G that do not have negative dependencies;
   G = the labeled dependency graph of P obtained considering (in addition
      to the initial EDB predicates) the predicates in $S_1, \ldots, S_i$ as EDB
end;
return $S_1, \ldots, S_i$;
```

Stratification: Example

Example (contd.):

The algorithm first considers the initial labeled dependency graph G of P:

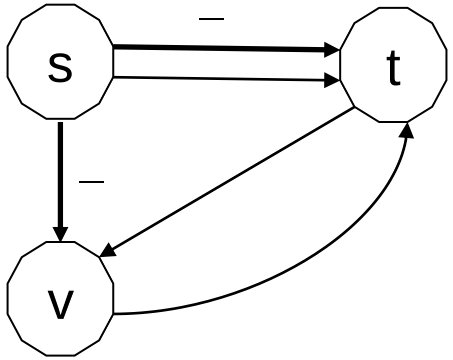


The only vertex that does not have negative dependencies is r, therefore

$S_1 = \{ r \}$

Stratification: Example

We now analyze the labeled dependency graph obtained considering r an EDB predicate, i.e. we eliminate the vertex r and all its outcoming edges:

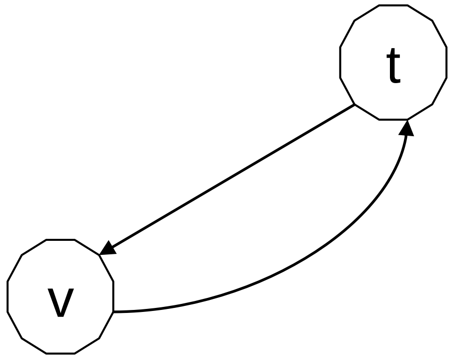


The only vertex that does not have negative dependencies is s, therefore

$S_2 = \{ s \}$

Stratification: Example

We now analyze the labeled dependency graph obtained considering both r and s as EDB predicates, i.e. we eliminate from the previous graph the vertex s and all its outcoming edges:



Since there are no more negated edges, both t and v have no negative dependencies, therefore

S_3 = { t, v }

And since, after deleting t and v, the graph is empty, the algorithm ends returning the stratification S_1, S_2, S_3.

Minimal model of a stratified program

Algorithm for computing the minimal model of a stratified program P:

```
Let $S_1,\dots, S_k$ be the stratification of P;
$MM_0$ = EDB(P);
For i=1 to k do begin
    $P(S_i)$ = the program obtained from P by considering only the rules
    having a predicate from $S_i$ in their head;
    $MM_i$ = minimal model of $P(S_i)$ cup $MM_{i-1}$
end;
return $MM_i$;
```

Example

Example (contd.):

If we execute the previous algorithm on program P, we obtain:

MM_0 = EDB(P) = $\{ p(a,b), p(b,c) \}$

$P(S_1)$ =

$r(X,Y) :- p(X,Y),\ not\ p(Y,X).$

$r(X,Y) :- p(X,Y),\ r(Y,Z).$

$MM_1 = MM_0 \cup \{ r(a,b), r(b,c), r(a,c) \}$

$P(S_2)$ =

$s(X,Y) :- r(Z,X),\ r(Z,Y),\ not\ r(X,Y).$

$MM_2 = MM_1 \cup \{ s(b,b), s(c,c) \}$

Example

P(S_3) =

$t(X) :- r(Y,X),\ s(X,X),\ not\ s(Y,Y).$

$t(X) :- v(X,Y).$

$v(X,Y) :- t(X),\ t(Y),\ not\ r(X,Y),\ not\ s(X,Y).$

$MM_3 = MM_2 \cup \{\ t(b), t(c), v(c,b)\ \}$

Therefore, the algorithm returns the model MM_3, i.e.:

$\{\ p(a,b), p(b,c), r(a,b), r(b,c), r(a,c), s(b,b), s(c,c), t(b), t(c), v(c,b)\ \}$

Adding disjunction to Datalog

Disjunctive rules

A **disjunctive** rule is an expression of the form

$$\alpha_1 \vee \ldots \vee \alpha_k \text{ :– } \beta_1, \ldots, \beta_n, not\ \gamma_1, \ldots, not\ \gamma_m.$$

where:

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

The rule is called **positive** if m=0, is called **non-disjunctive** if k=1, and is called **constraint** if k=0.

A **disjunctive program** (also called **Answer Set Program**) is a set of disjunctive rules.

Semantics of disjunctive rules

A ground disjunctive rule $\alpha_1 \vee \ldots \vee \alpha_k$:– $\beta_1, \ldots, \beta_n, not\ \gamma_1, \ldots, not\ \gamma_m$. is **satisfied** in an interpretation I if at least one of the following conditions holds:

- at least one of the atoms $\beta_1, \ldots, \beta_n$ does not belong to I
- at least one of the atoms $\gamma_1, \ldots, \gamma_m$ belongs to I
- at least one of the atoms $\alpha_1, \ldots, \alpha_k$ belongs to I

Example:

the rule $q(a) \vee q(b)$:– $p(b, a)$.

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

Answer Set Semantics

(same notion of reduct and answer set as the ones given for non-disjunctive programs)

Let P be a disjunctive program and let I be an interpretation. The **reduct** of P with respect to I is the positive disjunctive 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 disjunctive program. An interpretation I is an **answer set** of P if I is a minimal model of the positive disjunctive program P/I.

Reduct of a disjunctive program: Example

Example: let P be the program

$r(a) \lor s(a) \coloneq p(a), not\ q(a).$

$s(a) \lor t(a) \coloneq p(a), not\ r(a).$

$p(a).$

1) Let I_1 be the interpretation $\{p(a)\}$. Then, P/I_1 is the positive disjunctive program

$r(a) \lor s(a) \coloneq p(a).$

$s(a) \lor t(a) \coloneq p(a).$

$p(a).$

It is easy to verify that the first two rules of P/I_1 are not satisfied in I_1, therefore I_1 is not a model (and consequently not a minimal model) for P/I_1, so I_1 is not an answer set of P.

Example (contd.)

Example (contd.): let P be the program

$r(a) \lor s(a) :\!- \; p(a), not\ q(a)$.

$s(a) \lor t(a) :\!- \; p(a), not\ r(a)$.

$p(a)$.

2) Let I_2 be the interpretation $\{p(a), r(a)\}$. Then, P/I_2 is the positive program

$r(a) \lor s(a) :\!- \; p(a)$. ?

$p(a)$.

It is easy to verify that I_2 is a minimal model of P/I_2, therefore I_2 is an answer set of P.

Example (contd.)

Example (contd.): let P be the program

$r(a) \lor s(a) :\!- p(a), not\ q(a).$

$s(a) \lor t(a) :\!- p(a), not\ r(a).$

$p(a).$

3) Let I_3 be the interpretation $\{p(a), s(a)\}$. Then, P/I_3 is the positive disjunctive program

$r(a) \lor s(a) :\!- p(a).$

$s(a) \lor t(a) :\!- p(a).$

$p(a).$

It is easy to verify that I_3 is a minimal model of P/I_3, therefore I_3 is an answer set of P.

Example (contd.)

Example (contd.): let P be the program

$r(a) \vee s(a) \coloneq p(a), not\ q(a)$.

$s(a) \vee t(a) \coloneq p(a), not\ r(a)$.

$p(a)$.

4) Let I_4 be the interpretation $\{p(a), r(a), s(a)\}$. Then, P/I_4 is the positive disjunctive program

$r(a) \vee s(a) \coloneq p(a)$.

$p(a)$.

It is easy to verify that I_4 is a model of P/I_4, but is **not** a **minimal** model of P/I_4: the minimal models of P/I_4 are $\{p(a), r(a)\}$ and $\{p(a), s(a)\}$. Therefore, I_4 is not an answer set of P.

References

- Serge Abiteboul, Richard Hull, Victor Vianu: Foundations of Databases. Addison-Wesley 1995, ISBN 0-201-53771-0. http://webdam.inria.fr/Alice/
- Vladimir Lifschitz: Answer Set Programming. Springer 2019, ISBN 978-3-030-24657-0.
- Eiter, T., Gottlob, G. and Mannila, H. (2001): Disjunctive Datalog, ACM Transactions on Database Systems 22(3), July 2001
- Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, Francesco Scarcello: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic 7(3): 499-562 (2006)