Semantics

Model-theoretic, Fixpoint, and Proof-theoretic Semantics

In general, there are two possible views on Datalog programs

- Datalog rules are First-order Logic formulas (where the free variables are all-quantified), which define the desired answer set
  - $\Rightarrow$ leads to a model-theoretic or proof-theoretic semantics
- Datalog rules are operational rules that can be used to calculate the answer set.
  - $\Rightarrow$ leads to a fixpoint semantics

We now formalize the above-mentioned semantics for Datalog$^+$, study their properties, and finally show that they coincide.

Model-theoretic Semantics

Preliminaries

- Let $\mathcal{R}$ be a database schema and $\mathcal{I}$ be an instance over $\mathcal{R}$, i.e. $\mathcal{I}$ assigns a finite relation to each relational symbol of $\mathcal{R}$.
- $\mathcal{I}$ can be understood as a set of *(database) facts* of the form

$$\{R(a_1, \ldots, a_n) \mid R \in \mathcal{R}, (a_1, \ldots, a_n) \in \mathcal{I}(R)\}$$

Preliminaries (ctd)

- Let ρ be a rule of a Datalog$^+$ program Π over database schema $\mathcal{R}$ of the form

$$\rho := H \leftarrow G_1, \ldots, G_k$$

  and let $\mathcal{I}$ be an instance over $\mathcal{R}$.
- $\mathcal{I}$ satisfies ρ, if for each assignment ν of the variables in ρ:

$$\nu(G_1), \ldots \nu(G_k) \in \mathcal{I} \Longrightarrow \nu(H) \in \mathcal{I}$$

- $\mathcal{I}$ satisfies Π iff it satisfies every rule ρ from Π

Example

Consider the schema $\mathcal{R} := \{\mathrm{S}, \mathrm{T}\}$ and the instance $\mathcal{I}$ defined as

$$\mathcal{I} := \{\mathrm{S}(a), \mathrm{S}(b), \mathrm{T}(a)\}$$

Further consider the Datalog$^+$ program Π defined by the two rules

$$\rho_1 : \ \mathsf{S}(\mathsf{X}) \leftarrow \mathsf{T}(\mathsf{X}),$$
$$\rho_2 : \ \mathsf{T}(\mathsf{X}) \leftarrow \mathsf{S}(\mathsf{X}).$$

Then $\mathcal{I}$ satisfies ρ_1, but it does not satisfy ρ_2. Therefore, it does not satisfy program Π.

Preliminaries

Let Π be a Datalog program, let $\mathcal{R}$ be a database schema that contains exactly the relational symbols appearing in Π,[2] and let $edb(\Pi)$ ($idb(\Pi)$) denote the set of extensional (intensional) relational symbols of Π.

- The input for Π is an instance over $edb(\Pi)$.
- A *model* of Π is an instance over $\mathcal{R}$, i.e. $edb(\Pi) \cup idb(\Pi)$, that satisfies Π.

Definition: Model-theoretic Semantics

The model-theoretic semantics of Π w.r.t. to input $\mathcal{I}$, denoted as $\Pi(\mathcal{I})$, is the **minimal** model of Π containing the input $\mathcal{I}$.

[2] From now on, we call such a schema *matching* database schema

Fixpoint Semantics

Preliminaries

- Let ρ be a rule and ν be a variable assignment over all variables appearing in ρ. $\nu(\rho)$ is called *instantiation* or *ground instance* of ρ.
- Let Π be a Datalog$^+$ program and $\mathcal{I}$ be an instance over $\mathcal{R}$.
- The *immediate consequences* of Π and $\mathcal{I}$ are the facts A for which there holds:
  - (i) $A \in \mathcal{I}(R)$ **for some EDB relation** $R \in$ **edb**(Π), or
  - (ii) There is an instantiation $A \leftarrow A_1, \ldots, A_n$ of a rule from Π such that $A_i \in \mathcal{I}$ for $1 \leq i \leq n$.

Preliminaries (ctd)

- The *immediate consequence operator* T of a Datalog$^+$ program Π, T_Π, is a mapping over the instances of $\mathcal{R}$ defined as follows:

$$T_\Pi(\mathcal{I}) := \{A \mid A \text{ is an immediate consequence of } \Pi \text{ w.r.t. } \mathcal{I}\}$$

- T is called *monotonic*, if for all instances $\mathcal{I}, \mathcal{J}$ it holds that

$$\mathcal{I} \subseteq \mathcal{J} \implies T(\mathcal{I}) \subseteq T(\mathcal{J}).$$

- An instance $\mathcal{I}$ is called *fixpoint* of T, if $T(\mathcal{I}) = \mathcal{I}$.
- T_Π is monotonic[3].
- An instance $\mathcal{I}$ is a model of Π if and only if $T_\Pi(\mathcal{I}) \subseteq \mathcal{I}$. $(*)$
- Every fixpoint of T_Π is a model of Π, but the inverse does not hold.

[3] **Follows as the application of T_Π reproduces the EDB relations.**

Theorem

Let Π be a Datalog$^+$ program with input $\mathcal{I}$. T_Π has a minimal fixpoint that contains $\mathcal{I}$. This minimal fixpoint is exactly the minimal model of $\Pi(\mathcal{I})$.

Computation of the Minimal Fixpoint

- Let Π be a Datalog$^+$ program with input $\mathcal{I}$. Then it holds that

$$\mathcal{I} \subseteq T_\Pi(\mathcal{I}) \subseteq T_\Pi^2(\mathcal{I}) \subseteq T_\Pi^3(\mathcal{I}) \subseteq \ldots \subseteq \mathcal{M}(\Pi, \mathcal{I}).$$

- Let N be the number of facts in $\mathcal{M}(\Pi, \mathcal{I})$. Then $T_\Pi^i(\mathcal{I}) = T_\Pi^N(\mathcal{I}), i \geq N$, and, in particular, $T_\Pi(T_\Pi^N(\mathcal{I})) = T_\Pi^N(\mathcal{I})$.
- We denote the fixpoint $T_\Pi^N(\mathcal{I})$ as $T_\Pi^\infty(\mathcal{I})$.

Remark

The naive Datalog$^+$ evaluation algorithm discussed earlier implements the iterations of the T-operator for a Datalog$^+$ program Π with input $\mathcal{I}$.

Datalog¬

Inflationary Semantics

- We now consider Datalog programs with negation in their bodies
- To evaluate such programs, we take the underlying active domain as a basis.
- To derive new facts, we fire all rules of a program simultaneously and, in each step, derive facts for each rule w.r.t. all possible variable assignments.
- A negative fact of an IDB relation of the form $\neg P$ in a rule is considered to be satisfied whenever the fact P has not been derived in previous iterations.
- The rules are evaluated iteratively, until a fixpoint is reached. A fact is considered to be derived, if it has been derived in some iteration.
- The output of the program is defined as the set of all facts that are derived within the iteration process

Definition: Immediate Consequence Operator for Datalog$^\neg$

Let Π be a Datalog$^\neg$ program and let $\mathcal{K}$ be an instance over the relational schema defined by Π.

A fact A is called *immediate consequence* of Π and $\mathcal{K}$, if

- $A \in \mathcal{K}(R)$ for some EDB relation R, or
- $A \leftarrow L_1, \ldots, L_m$ is an instantiation (w.r.t. the active domain) of a rule from Π such that for every positive L_i it holds that $L_i \in \mathcal{K}$ and for every negative $L_i := \neg A_i$ it holds that $A_i \notin \mathcal{K}$.

The *immediate consequence operator* Γ of Π, Γ_Π, is then defined as follows.

$$\Gamma_\Pi(\mathcal{K}) := \mathcal{K} \cup \{A \mid A \text{ is an immediate consequence of } \Pi \text{ w.r.t. } \mathcal{K}\}$$

Example: Datalog$^\neg$

Complement of the transitive closure of G: is the following program correct?

$$T(X, Y) \leftarrow G(X, Y)$$
$$T(X, Y) \leftarrow T(X, Z), G(Z, Y)$$
$$Compl(X, Y) \leftarrow \neg T(X, Y)$$

Inflationary Fixpoint

Let $\mathcal{I}$ be an input for program Π. By definition, it holds that

$$\Gamma_\Pi(\mathcal{I}) \subseteq \Gamma_\Pi^2(\mathcal{I}) \subseteq \Gamma_\Pi^3(\mathcal{I}) \ldots$$

This sequence has a fixpoint $\Gamma_\Pi^\infty(\mathcal{I})$ with output $\Pi(\mathcal{I})$, which is reached after a finite number of iterations.

Remark: In contrast to operator $T_\Pi^\infty(\mathcal{I})$ for Datalog$^+$ we can observe that $\Gamma_\Pi^\infty(\mathcal{I})$ is not necessarily minimal and, in particular, not a minimal model (see the example on the next slide).

Example: Fixpoint and Minimal Models of Datalog$^\neg$

Consider the program Π:

$$R(0) \leftarrow Q(0), \neg R(1)$$
$$R(1) \leftarrow Q(0), \neg R(0)$$

Let $\mathcal{I} := \{Q(0)\}$. Then $\Pi(\mathcal{I}) = \{Q(0), R(0), R(1)\}$. $\Pi(\mathcal{I})$ is a model, but not a minimal one. The two models shown below are minimal (each):

$$\{Q(0), R(0)\} \qquad \{Q(0), R(1)\}$$

Example: Datalog$^\neg$

Let the relation G represent the edge relation of a directed graph. The following Datalog$^\neg$ program defines a relation *Closer* such that *Closer*(X,Y,X',Y') holds iff the shortest path from X to Y in G is shorter than the shortest path from X' to Y'. Whenever two nodes are not connected via edges of G, we assume the distance to be ∞.

$$
\begin{aligned}
T(X, Y) &\leftarrow G(X, Y) \\
T(X, Y) &\leftarrow T(X, Z), G(Z, Y) \\
\mathit{Closer}(X, Y, X', Y') &\leftarrow T(X, Y), \neg T(X', Y')
\end{aligned}
$$

Example

What does the following program compute when given a non-empty graph edge relation G as input?

$$
\begin{aligned}
&T(X, Y) \leftarrow G(X, Y) \\
&T(X, Y) \leftarrow T(X, Z), G(Z, Y) \\
&oldT(X, Y) \leftarrow T(X, Y) \\
&oldTexceptFinal(X, Y) \leftarrow T(X, Y), T(X', Z'), T(Z', Y'), \neg T(X', Y') \\
&CT(X, Y) \leftarrow \neg T(X, Y), oldT(X', Y'), \neg oldTexceptFinal(X', Y')
\end{aligned}
$$

Stratified Datalog$^\neg$

Towards model-theoretic semantics of Datalog$^\neg$.

Operator $T^\#$: adapting the T-Operator to negation.

Let Π be a - safe - Datalog$^\neg$ program and $\mathcal{I}$ be an instance of $\mathcal{R}$. The *immediate consequences* of Π and $\mathcal{I}$ are facts A defined as follows:

- $A \in \mathcal{I}(R)$ for some EDB relation $R \in \mathcal{R}$, or
- there is an instantiation $A \leftarrow L_1, \ldots, L_n$ of a rule from Π such that for every positive L_i it holds that $L_i \in \mathcal{I}$ and for every negative $L_i = \neg A_i$ it holds that $A_i \notin \mathcal{I}$, $1 \leq i \leq n$.

The *immediate consequence operator* $T^\#$ of Π, $T^\#_\Pi$, is a mapping over the instances of $\mathcal{R}$ defined as follows.

$$T^\#_\Pi(\mathcal{I}) = \{A \mid A \text{ is an immediate consequence of } \Pi \text{ w.r.t. } \mathcal{I}\}$$

Discussion: **Operator $T^{\#}$, when applied on all kind of programs, has some unexpected properties.**

- Consider a program Π of the form

$$p \leftarrow \neg p$$

  We can observe that the $T^{\#}_{\Pi}$ operator has no fixpoint, because the iterative fixpoint computation $\{T^{\#,i}_{\Pi}(\emptyset)\}_{i>0}$ does not terminate.
- The $T^{\#}$ operator does not necessarily have a unique fixpoint, e.g. for Π of the form

$$p \leftarrow \neg q$$
$$q \leftarrow \neg p$$

  the two minimal fixpoints are $\{p\}$ and $\{q\}$, respectively. Both of them do not result from the corresponding terminating iterative computation.

$\Rightarrow$ iteration of the $T^{\#}$ operator is not a satisfactory solution

Datalog$^\neg$: Stratified Semantics

Remark

If we extend each program w.r.t. to an n-ary IDB relation R by a rule of the form

$$R(X_1, \ldots, X_n) \leftarrow R(X_1, \ldots, X_n),$$

then the $T^\#$ operator evaluates the program according to the inflationary semantics.

Rules of the form $p \leftarrow p$ are equivalent to $p \vee \neg p$ and thus tautologies.

Stratified Semantics

Definition: Stratification

A Datalog program is called *stratified* if its dependency graph does not contain a cycle going through a negative (i.e., $\neg$-labeled) edge.

Definition: Stratification of a Program

- Let R be a relational symbol contained in a rule of some stratified Datalog$^\neg$ program Π
- Let $S(R)$ be the maximum over the number of $\neg$-labeled edges over all paths leading to R. $S(R)$ is called *stratum* of R.
- Let n be the maximum within the set $\{S(R) \mid R \text{ is a relational symbol in } \Pi\}$

  The partitioning $\{\Pi^1, \Pi^2, \ldots, \Pi^n\}$ of the rules in Π such that each Π^i contains exactly those rules of Π whose head relational symbol has stratum $i-1$ is called *stratification* of Π.

Evaluation of Stratified Programs

- The (stratified) semantics of a stratified Datalog$^\neg$ program is defined by a stratified composition of minimal models, where each minimal model can be compute by iteration of the $T^\#$ operator.
- When used in the context of stratified semantics, we therefore shall refer to the $T^\#$ operator as T operator.

Stratified Semantics

Let $\sigma := \Pi^1, \Pi^2, \ldots, \Pi^n$ be a stratification of some Datalog$^\neg$ program Π and let the input $\mathcal{E}$ be an instance over the EDB relations of Π. Further let

$$\begin{aligned} \mathcal{I}_0 &:= \mathcal{E}, \\ \mathcal{I}_i &:= \mathcal{I}_{i-1} \cup \Pi^i(\mathcal{I}_{i-1}), 0 < i \leq n. \end{aligned}$$

The *stratified semantics* $\Pi^{strat}(\mathcal{E})$ of Π w.r.t. input $\mathcal{E}$ is defined as $\mathcal{I}_n$.

$\Pi^i(\mathcal{I}_{i-1})$ is the minimal model of program Π^i w.r.t. input $\mathcal{I}_{i-1}$.

Theorem

$\Pi^{strat}(\mathcal{E})$ is a minimal model of Π w.r.t. input $\mathcal{E}$. This model is also a minimal fixpoint w.r.t. T_Π.

Hence, the stratified semantics chooses a minimal model out of the set of all models. This model is uniquely determined by the *syntax* of the program, i.e. its stratification. Informally speaking, the programmer's intuition defines the minimal model.

$\Rightarrow$ due to this reason, the stratified semantics is generally accepted as a semantics for stratified Datalog$^\neg$ programs

Example

Consider the program

$$\begin{aligned} \mathit{greenPath}(X, Y) &\leftarrow \mathit{green}(X, Y) \\ \mathit{greenPath}(X, Y) &\leftarrow \mathit{greenPath}(X, Z), \mathit{greenPath}(Z, Y) \\ \mathit{bingo}(X, Y) &\leftarrow \mathit{red}(X, Y), \neg \mathit{greenPath}(X, Y) \end{aligned}$$

and input

green		
	1	2

red		
	1	2
	2	3

The following two models are minimal.

- greenPath $= \{(1, 2)\}$, bingo $= \{(2, 3)\}$,
- greenPath $= \{(1, 2), (2, 3), (1, 3)\}$, bingo $= \emptyset$.

The first one is obtained by application of the stratified semantics.

The inflationary fixpoint is defined by greenPath $= \{(1, 2)\}$, bingo $= \{(1, 2), (2, 3)\}$; it is a model, but not a minimal one.

Example

Consider the following two propositional programs.

$$\Pi_1 : \quad p \leftarrow \neg q$$
$$\Pi_2 : \quad q \leftarrow \neg p$$

It holds that $\Pi_1 \equiv \Pi_2 \equiv (p \vee q)$.

First observe that $\mathcal{M}_1 = \{p\}$ and $\mathcal{M}_2 = \{q\}$ both are minimal models of $p \vee q$, and thus minimal models of both Π_1 and Π_2.

Π_1 and Π_2 have different stratified semantics, though:

- $\mathcal{M}_1 := \{p\}$ is the stratified semantics of Π_1
- $\mathcal{M}_2 := \{q\}$ is the stratified semantics of Π_2

Limitations of the Stratified Semantics

- Drawback: some Datalog$^\neg$ don't have a stratified semantics.
- Well-founded semantics is defined for every Datalog$^\neg$ program .
- Whenever a Datalog$^\neg$ program is stratified, its stratified and its well-founded semantics coincide.
- There exists an algorithm to compute the well-founded semantics called *Alternating Fixpoint Algorithm*.

Running example: An abstraction of a two-person game

A game is represented by its states $a, b, c, \ldots$ which could reached during playing. The possible decisions, called moves, are held in a binary relation *moves*, where $(a, b) \in moves$ indicates, that when in state a, one can choose to move to state b. A player loses if he or she is in a state from which there are no moves. The goal is to compute a set of winning states, i.e. the set of states such that there exists a winning strategy for a player in this state. The winning states are obtained in a unary relation *win*. The following rule describes this game:

$$win(X) \leftarrow move(X, Y), \neg win(Y)$$

a → b → c → d

Example 1

Consider the rule

$$win(X) \leftarrow move(X, Y), \neg win(Y)$$

and the input $\mathcal{E}(move) = \{(a, b), (b, c), (c, d)\}$.

Question: which instances of *win* are a model of the rule when given this input?

- The models $win := \{a, c\}$ and $win := \{b, d\}$ are minimal.
- Only the model $win := \{a, c\}$ is minimal and *supported*; it is the well-founded semantics.

a, c winning strategy
b, d, not.

Example 2

Consider the same rule

$$win(X) \leftarrow move(X, Y), \neg win(Y)$$

and the input $\mathcal{E}(move) := \{(a, b), (b, c), (a, c)\}$.

The model win $= \{a, b\}$ is minimal; it is the well-founded semantics.

Example 3

Consider the same rule

$$win(X) \leftarrow move(X, Y), \neg win(Y)$$

Now let $\mathcal{E}(move) := \{(a, b), (b, c), (c, a), (a, d), (d, e), (d, f), (f, g)\}$.

Are there "intuitive" models? If so, which ones?

The following *3-ary model* is intuitively convincing:

true:	$win(d), win(f)$
false:	$win(e), win(g)$
undefined:	$win(a), win(b), win(c)$

It is the well-founded semantics.