Semantics for Datalog with Subsumption and Completion

Luka Janjić
King's College London
luka.janjic@kcl.ac.uk

Michael D. Adams
National University of Singapore
https://michaeldadams.org/

Many programming problems can be naturally viewed as computing the least fixed point of a monotonic transition function over sets of known facts (database states). Datalog is the prototypical programming language based on this view of computation. However, core Datalog is not expressive enough for many problem domains, leading to a host of extensions. Two features that significantly broaden the applicability are the subsumption of facts and completion of database states. Subsumption allows for the facts to be partially ordered, indicating that some facts are "stronger" than others and make them obsolete, while completion selects a subset of database states as "complete" and sends any other states into complete ones. Traditionally, the semantics of Datalog programs are given model-theoretically in terms of Herbrand structures, and operationally in terms of so called fixed-point semantics. We give a generalized version of both flavors of semantics, incorporating the notions of subsumption and completion, and entailing the capabilities of many Datalog extensions.

1 Introduction

Many programming problems can be naturally viewed as computing the least fixed point of a monotonic transition function over sets of known facts. We dub this fixed-point oriented programming (FPOP) [3]. Datalog [4] is the prototypical programming language based on this view of computation. A Datalog program consists of rules of the form $A_1, \ldots, A_n \vdash C$, where A_i and C are atoms. We see a rule as a universally quantified implication $\forall x_1 \ldots x_m.\ A_1 \wedge \ldots \wedge A_n \implies C$, quantifying over variables that occur in the rule. To execute a Datalog program, we treat the rules as functions, taking known facts that match the premises and producing new facts corresponding to the conclusion. As such, the rules are iterated until a fixed point is reached.

Recently, we introduced a new FPOP language called Fixen [5], extending Datalog with subsumption and completion. To introduce the concepts, consider the problem of computing the shortest path in a graph. We could attempt to express it as the following Datalog[1] program:

```
rule init    : ⊢ distTo "start" 0
rule addDist : distTo v₁ d₁, edge v₁ v₂ d₂ ⊢ distTo v₂ (d₁ + d₂)
```

The issue with this program is that, if the graph contains cycles, this program will never terminate. The solution lies in observing that, in the iterative implementations of shortest path algorithms, the programmer further expresses that shorter paths are "better"; that shorter paths to the same node make longer ones obsolete. This kind of relationship between facts we call *subsumption*. Thus, in Fixen, we would also express that any `distTo x d1` is subsumed by any `distTo x d2` if d2 ≤ d1.

Next, suppose that we are given a maximal distance d_{max} such that we are not interested in any nodes that lie beyond that distance of the start. Then we may want to limit the valid states to only those where

[1] We use Fixen notation, which inverts the usual Datalog notation and follows the standard order of writing implications: the premises are on the left hand side, while the conclusion is on the right.

Submitted to:
FICS 2026

the second argument of any `distTo` fact is bounded by d_{max}. In Fixen we can achieve this by a completion function $\mathcal{C}(X) = \{x \in X \mid x = \texttt{distTo v d} \implies \texttt{d} \leq d_{max}\}$, filtering the out-of-bound facts. Note that this is a very simple example chosen to illustrate the idea of completions, not to demonstrate their expressiveness. More involved examples in the area of static analysis can be found in the paper [5].

2 Overview

The intuitive meaning of an FPOP program is that, given an initial database state, it represents the state that satisfies the rules, where we mean "satisfies" in the logical sense: if the premises are in the database, then the conclusion is, too. On the other hand, operationally, we think of a program as a procedure for finding a satisfying database state. Starting from the smallest under-approximation, each iteration computes an increment and merges it with the current state until the satisfying one is reached.

We model the database states by a *join-semilattice with a least element*. A *join-semilattice* $(L, \sqsubseteq, \sqcup)$ consists of a set L partially ordered by a relation $\sqsubseteq$, where every pair of elements has the *least upper bound* given by a binary operator $\sqcup$. The *least element* $l \in L$ is such that for every other $l' \in L$ we have that $l \sqsubseteq l'$, and is necessarily unique. For brevity, in the rest of the paper we will refer to these simply as semilattices. Relating this to the intuition, the set L represents the possible database states. The relation $\sqsubseteq$ formalizes the notion of a database state being more general than another: $D \sqsubseteq D'$ means that all the facts contained in the state D are contained by the state D'. The operator $\sqcup$ formalizes the notion of merging two states: $D \sqcup D'$ is the smallest state that contains all the facts of D and of D'. Finally, the least element represents the initial database state. Thus, the model-theoretic semantics define the semilattice of states and the notion of a state satisfying the rules of the program, while the fixed-point semantics specify the iterative, refinement-based approach of finding such a state.

3 Model-theoretic semantics

The standard way of giving model-theoretic semantics for Datalog [1] is in terms of Herbrand structures. A *Herbrand universe* $\mathcal{U}$ is the set of all ground terms that occur in the program. A *Herbrand base* $\mathcal{B}$ is the set of all ground atoms: relations applied to ground terms. We also refer to ground atoms as *facts*. An *interpretation* I is any subset of the base $\mathcal{B}$. The set of interpretations forms the powerset (semi)lattice in the usual sense.

The satisfaction relation $\models$ relates Datalog programs and interpretations. Generally, for some logical element X and some interpretation I, we will write $I \models X$ and say that X is *satisfied* by I. A fact F is satisfied by an interpretation I when $F \in I$. A rule R is satisfied by I if, for every instantiation of the rule, whenever all its premises are satisfied, its conclusion is satisfied. We write $\text{conc}_R(I)$ for the set of immediate conclusions of R that follow from I. To make this more precise, let an *instantiation* of a well-formed rule $[v_1/x_1, \ldots, v_m/x_m](A_1, \ldots, A_n \vdash C)$ be obtained by instantiating all universally quantified variables $x_1 \ldots x_m$ by values $v_1 \ldots v_m \in \mathcal{U}$. We denote the set of all substitutions that instantiate every variable in R with $\text{Subst}(R)$. Then, we can define $\text{conc}_R(I) = \left\{\sigma(B) \mid \sigma \in \text{Subst}(R), \forall j.\, I \models \sigma(A_j)\right\}$. Thus, $I \models R$ if $\text{conc}_R(I) \subseteq I$. For a program P, we say it is satisfied by an interpretation I when all the rules of P are satisfied by I. An interpretation that satisfies a program is called its *model*. It can be easily demonstrated that every program has a unique *least model*, which we take as its meaning. Moreover, due to the restrictions placed on valid Datalog programs, the least model is always finite and can be computed in a straightforward way, shown by its fixed-point semantics.

3.1 Subsumption and completion

We turn to the model-theoretic semantics of Fixen, showing how we extend the Datalog case to account for subsumption and completion. Herbrand universe $\mathcal{U}$ and base $\mathcal{B}$ are no longer plain sets, but partially ordered sets (posets). The relation $\leftarrow$ partially ordering $\mathcal{B}$ represents fact subsumption. All relations are required to be monotonic: for any relation $\mathtt{R}$ if $\mathtt{x}_1 \sqsubseteq \mathtt{y}_1 \ldots \mathtt{x}_n \sqsubseteq \mathtt{y}_n$ then $\mathtt{R}\ \mathtt{x}_1 \ldots \mathtt{x}_n \leftarrow \mathtt{R}\ \mathtt{y}_1 \ldots \mathtt{y}_n$. The set of interpretations is the set of all downward closed sets (downsets) $\mathcal{I} = \{\downarrow X \mid X \subseteq \mathcal{B}\}$. A set $X \subseteq \mathcal{B}$ is downward closed when $x \in X$ and $y \leftarrow x$ implies $y \in X$. $\downarrow X$ denotes the downward closure of a set X, that is, the least downset containing X.

Let a completion $C : \mathcal{I} \to \mathcal{I}$ be a monotonic idempotent function over the interpretations. Reflecting the intuition behind completion, the database states no longer denote interpretations, but those interpretations closed under C. Thus, we define the set of *complete interpretations* $\mathcal{K} = \{C(I) \mid I \in \mathcal{I}\}$.

Before we can relate programs and complete interpretations, we must clarify the behavior of functions. In Datalog, any occurrence of function symbols in rules is prohibited. However, in Fixen, functions can occur in the conclusions of rules, such as the $+$ function in the conclusion of the `addDist` rule in the example from Section 1. The functions are assumed to be given externally, and their interpretation is fixed, so we keep them opaque to the semantics. We use the operator $[\![-]\!]$ to evaluate a ground atom by evaluating any function symbol occurring in its arguments, and generalize the immediate conclusions to $\mathrm{conc}_R(I) = \left\{ [\![\sigma(B)]\!] \mid \sigma \in \mathrm{Subst}(R), \forall j.\, I \vDash \sigma(A_j) \right\}$. Accordingly, we distinguish between ground atoms, in which function calls may occur, and facts, which are fully evaluated ground atoms; the Herbrand base contains only facts, not all ground atoms. Lastly, to ensure downward-closedness and completeness, we define the complete consequences of a rule $\mathrm{infer}_R(K) = C\left(\downarrow \mathrm{conc}_R(K)\right)$, and adapt the rule satisfaction condition to $\mathrm{infer}_R(K) \subseteq K$.

Complete interpretations form a semilattice $(\mathcal{K}, \subseteq, \sqcup)$ with $\subseteq$ as the order relation, $I \sqcup J = C(I \cup J)$ as the join operator, and $\bot = C(\emptyset)$ as the least element. The binary join operator generalizes to the least upper bound of a set $\bigsqcup \mathbf{X} = C\left(\bigcup \mathbf{X}\right)$ for some $\mathbf{X} \subseteq \mathcal{K}$. To see that it is an upper bound, notice that for any $X_i \in \mathbf{X}$ we have $X_i \subseteq \bigcup \mathbf{X}$ and from monotonicity and idempotence of C we conclude $X_i = C(X_i) \subseteq C\left(\bigcup \mathbf{X}\right)$. To see that it is the least such, suppose $K \in \mathcal{K}$ is an upper bound of $\mathbf{X}$. Thus, $X_i \subseteq K$ for each X_i, and hence $\bigcup \mathbf{X} \subseteq K$. Then, by monotonicity and idempotence of C we conclude that $C\left(\bigcup \mathbf{X}\right) \subseteq C(K) = K$.

We summarize the model-theoretic semantics:

Interpretations	$\mathcal{I} = \{\downarrow X \mid X \in \mathcal{P}(\mathcal{B})\}$
Complete interpretations	$\mathcal{K} = \{C(I) \mid I \in \mathcal{I}\}$
Least complete interpretation	$\bot = C(\emptyset)$
Least upper bound	$\bigsqcup X = C\left(\bigcup X\right)$
Immediate conclusions	$\mathrm{conc}_R(I) = \left\{ [\![\sigma(B)]\!] \mid \sigma \in \mathrm{Subst}(R), \forall j.\, I \vDash \sigma(A_j) \right\}$
Complete consequences	$\mathrm{infer}_R(K) = C\left(\downarrow \mathrm{conc}_R(K)\right)$
Fact satisfaction	$K \vDash F \iff F \in K$
Rule satisfaction	$K \vDash R \iff \mathrm{infer}_R(K) \subseteq K$
Program satisfaction	$K \vDash P \iff \forall R \in P.\, K \vDash R$

4 Fixed-point semantics

The goal is to show that the least fixed point of the rules, viewed as a transition function, may be computed by iterative application and corresponds to the least model of the program. We show that every fixed point of the program's transition function is a model of that program. From there it follows that the least fixed point is the least model and that the least model always exists.

4.1 Meaning of a program

The transition function of the program is defined by the transition functions for each individual rule. For a rule $R = A_1, \ldots, A_n \vdash B$, its transition function $T_R : \mathcal{K} \to \mathcal{K}$ is simply its complete consequence function $T_R(K) = \text{infer}_R(K)$. The transition function $T_P : \mathcal{K} \to \mathcal{K}$ for the entire program applies all rules to the current interpretation and joins their conclusions to the input. Thus, the full transition function is:

$$T_P(K) = C\left(K \cup \bigcup_R T_R(K) \right) = \bigsqcup \big(\{K\} \cup \{\, T_R(K) \mid R \in P\} \big)$$

The meaning of a program P under fixed-point semantics is the **least fixed point** of its transition function lfp(T_P). We want to ensure that the least fixed point of the transition function can be obtained by iterated application to the bottom element. In order to guarantee this, we require two conditions: the transition function must be monotonic and the lattice of complete interpretations $\mathcal{K}$ must have no infinite ascending chains ($K_1 \subset K_2 \subset K_3 \subset \ldots$ for $K_i \in \mathcal{K}$). The latter we call the *ascending chain condition* (ACC).

A common case in which $\mathcal{K}$ meets the ACC is when $\mathcal{B}$ meets the ACC and has a finite width; the width of a set being the size of the largest subset of incomparable elements. To see this, suppose, by contrapositive, that there is an infinite ascending chain $K_1 \subset K_2 \subset \ldots$ in $\mathcal{K}$. Then, let $\Delta_i = K_{i+1} \setminus K_i$ be the difference between consecutive sets in the chain. Notice that, since K_i are downsets, for any $x \in \Delta_i$ and $y \in \Delta_j$ where $i < j$, either $x \leftarrow y$ or x and y are incomparable. Since the chain is infinite, we must have at least one of the following: infinitely many incomparable elements or an infinite ascending chain of elements.

Alternatively, we can cope with infinite ascending chains of interpretations by using a suitable completion function. For example, in applications to abstract interpretation, a common way to deal with potential non-termination is to use a technique called narrowing, which, in a Fixen implementation, corresponds to using a completion function that sends all states above some threshold to a maximal state.

To see that the transition function is always monotonic, let $I, J \in \mathcal{K}$ be complete interpretations where $I \subseteq J$. Suppose $x \in {\downarrow}\text{conc}_R(I)$, which means $x \sqsubseteq [\![\sigma(B)]\!]$ for some $\sigma \in \text{Subst}(R)$ such that $I \vDash \sigma(A_j)$ for all A_j. Since $I \subseteq J$, we also have that $J \vDash \sigma(A_j)$ for all A_j, and hence $x \sqsubseteq [\![\sigma(B)]\!] \in \big\{ [\![\sigma(B)]\!] \mid \sigma \in \text{Subst}(R), \forall j.\, J \vDash \sigma(A_j) \big\}$ and $x \in {\downarrow}\text{conc}_R(J)$. Thus ${\downarrow}\text{conc}_R(I) \subseteq {\downarrow}\text{conc}_R(J)$, and since C is monotonic $T_R(I) = C({\downarrow}\text{conc}_R(I)) \subseteq C({\downarrow}\text{conc}_R(J)) = T_R(J)$, showing that T_R is monotonic. From there, it is easy to see that T_P is also monotonic.

Thus, only the programs whose semilattice over $\mathcal{K}$ meets the ascending chain condition are semantically well defined.

4.2 Correspondence with models

First, we show that the fixed points of a transition function T_P of a valid program P correspond to its models. Let $K \in \mathcal{K}$ be a complete interpretation and suppose that K is a fixed point. Hence $T_P(K) =$

$\bigsqcup(\{K\} \cup \{T_R(K) \mid R \in P\}) = K$, which holds if and only if $T_R(K)$ is upper bound by K for all rules R. By definition, this is equivalent to $K \vDash R$ holding for each R, which is the definition of $K \vDash P$, so fixed points and models coincide.

Next, we show that the least fixed point of T_P is computed by iterated application to the bottom element $\bot$. We first show that iterated application reaches a fixed point. By definition, we have that $C(\bot) \subseteq T_P(\bot)$. By monotonicity of T_P, we have that ${T_P}^n(\bot) \subseteq {T_P}^{n+1}(\bot)$ for any n. Therefore, we have an ascending chain $\bot \subseteq T_P(\bot) \subseteq \ldots \subseteq {T_P}^n(\bot) \subseteq \ldots$. Due to the ascending chain condition, this chain stabilizes: there is some n for which ${T_P}^n(\bot) = {T_P}^{n+1}(\bot)$. To see that this is the least fixed point, suppose that there is an $X \in \mathcal{K}$ such that $T_P(X) = X$. By definition, we have $\bot \subseteq X$, and hence, by monotonicity, ${T_P}^n(\bot) \subseteq {T_P}^n(X) = X$. Therefore, ${T_P}^n(K)$ is the least fixed point.

Since the models and fixed points correspond, it follows that the least model is the least fixed point, and since the least fixed point always exists, so does the least model. Together, this establishes the correctness of the iterative approach to evaluation of Fixen programs with respect to the model-theoretic semantics.

5 Practical concerns

A crucial aspect of Fixen is the generalization of Datalog to support potentially infinite interpretations. However, in an implementation, we must represent an interpretation by a finite state. We do this by keeping only the set of maximal facts in the database, justified by the fact that a downset is uniquely defined by the set of its maximal elements. Thus, to ensure finiteness of state, each complete interpretation should have a finite set of maximal elements.

Since we represent interpretations by their maximal elements, we must cope with another problem. Consider a simple program with only the rule `p x ⊢ q (f x)`, where `f 2` $=$ `0` and otherwise `f x` $=$ `x`, and where `p x` $\leftarrow$ `p y` when `x` $\geq$ `y`. Suppose we start with the initial fact `p 1`. Then the state that we explicitly represent is {`p 1`}, while the interpretation it denotes is {`p x` | `x` $\geq$ `1`}. Executing the program with only this representation, we match the first premise with the fact `p 1`, conclude `q (f 1)` $=$ `q 1` and seemingly reach the fixed point with the state {`p 1`, `q 1`}. Nevertheless, `p 2` is in the interpretation, but since we only consider our state, we never conclude `q (f 2)` $=$ `q 0` as prescribed by the semantics.

Generally speaking, such issues might occur whenever there is a breach of monotonicity of a rule. A rule $R = A_1, \ldots, A_n \vdash B$ is monotonic when for any substitutions $\sigma, \sigma' \in \mathrm{Subst}(R)$, if $\sigma(A_i) \leftarrow \sigma'(A_i)$ for all i then $[\![\sigma(B)]\!] \leftarrow [\![\sigma'(B)]\!]$. This is distinct from the monotonicity of infer_R which holds regardless of monotonicity of R. If a rule is monotonic, it is sufficient to consider only the maximal facts that fit the premises, since any subsumed premises will always yield subsumed conclusions. It is easy to see that a rule is monotonic as long as any functions used in the conclusions are monotonic. Therefore, to guarantee the correctness of our implementation, we require the functions used to be monotonic; a restriction that was not overly restrictive for the examples we considered.

References

[1] Stefano Ceri, Georg Gottlob, Letizia Tanca et al. (1989): *What you always wanted to know about Datalog(and never dared to ask)*. *IEEE transactions on knowledge and data engineering* 1(1), pp. 146–166.

[2] Edsger Wybe Dijkstra (1959): *A note on two problems in connexion with graphs*. *Numerische Mathematik* 1, pp. 269–271, doi:10.1007/BF01386390.

[3] Yong Qi Foo, Brian Sze-Kai Cheong & Michael D. Adams (2025): *Fixed-Point-Oriented Programming: A Concise and Elegant Paradigm*. arXiv:2507.21439.

[4] Hervé Gallaire & Jack Minker (1977): *Logic and Data Bases*. *Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*.

[5] Luka Janjić, Brian Sze-Kai Cheong, Ken Jin Ooi, Yong Qi Foo & Michael D. Adams (2026): *Fixen: A Fixed-Point-Oriented Language With Priorities and Completions*. Under review.