

Datalog and Friends

Prof. Dr. G. Lausen, M. Meier, M. Schmidt

28. Juni 2010

Deductive Databases: Datalog

Motivation

- Datalog: **D**atabases in **l**ogic.
- Logic-based query language for the relational model with clean and compact semantics
- Resembles the *Prolog* programming language
- Queries are expressed as rules (vs. operational queries in relational algebra/declarative queries in SQL)
- Widely studied in database research, central topics:
 - Semantics of Datalog
 - Relationship Datalog vs. relational algebra
 - Extensions of Datalog
 - Expressiveness of Datalog and its fragments

Deductive Databases: Datalog

Datalog Queries: Rules

- Queries are expressed as rules
- A rule is an implication of the form

$$H(\bar{x}) \leftarrow L_1(\bar{x}, \bar{y}_1), \dots, L_n(\bar{x}, \bar{y}_n), \text{ where}$$

- $n \geq 0$,
- H is a relational symbol,
- L_1, \dots, L_n are literals (i.e., possibly negated relational atoms, possibly with constants),
- $\bar{x}, \bar{y}_1, \dots, \bar{y}_n$ are vectors of variables,
- $R(\bar{z})$ denotes an atomic formula over variables from \bar{z} (i.e., a formula using a subset of the variables in \bar{z})
- H is called *head* and $L_1(\bar{x}, \bar{y}_1), \dots, L_n(\bar{x}, \bar{y}_n)$ is called *body*.
- A set of rules is called *program*

Datalog: Example Queries

Example

Consider the relational schema: $Flight[Company, From, To, Start, End]$ and the database instance (specifying daily flight connections):

<i>Company</i>	<i>From</i>	<i>To</i>	<i>Start</i>	<i>End</i>
LH	FFT	BER	9:00	10:00
AA	ST	NY	9:30	16:00
LH	MUE	ROMA	10:00	12:00
BA	DAL	LON	17:00	24:00
LH	FFT	DAL	8:00	16:00
BA	LON	NY	10:00	15:00

Datalog: Example Queries

Which destinations are reachable from Frankfurt (FFT) with direct flight connection?

$$\text{FDest}(X) \leftarrow \text{Flight}(_, \text{'FFT'}, X, _, _)$$

Note

The placeholder “_” is a shortcut for a (distinguished) variable that appears only in the body of the rule.

Which destinations can be reached from Frankfurt (FFT) when starting at 9:00 and changing the plane at most once?

$$\text{FDest9am}(X) \leftarrow \text{Flight}(_, \text{'FFT'}, X, \text{'9:00'}, _)$$
$$\text{FDest9am}(Y) \leftarrow \text{Flight}(_, \text{'FFT'}, X, \text{'9:00'}, _), \text{Flight}(_, X, Y, _, _)$$

Datalog: Example Queries

Which destinations are reachable from Frankfurt (FFT) with direct flight connection?

$$\text{FDest}(X) \leftarrow \text{Flight}(_, \text{'FFT'}, X, _, _)$$

Note

The placeholder “_” is a shortcut for a (distinguished) variable that appears only in the body of the rule.

Which destinations can be reached from Frankfurt (FFT) when starting at 9:00 and changing the plane at most once?

$$\text{FDest9am}(X) \leftarrow \text{Flight}(_, \text{'FFT'}, X, \text{'9:00'}, _)$$
$$\text{FDest9am}(Y) \leftarrow \text{Flight}(_, \text{'FFT'}, X, \text{'9:00'}, _), \text{Flight}(_, X, Y, _, _)$$

Datalog: Example Queries

Which destinations can be reached from Frankfurt (FFT)?

$$\text{DestRec}(X) \leftarrow \text{Flight}(-, \text{'FFT'}, X, -, -)$$
$$\text{DestRec}(Y) \leftarrow \text{DestRec}(X), \text{Flight}(-, X, Y, -, -)$$

Which destinations can be reached from Frankfurt (FFT) taking only Lufthansa (LH) flights?

$$\text{LHDestRec}(X) \leftarrow \text{Flight}(\text{'LH'}, \text{'FFT'}, X, -, -)$$
$$\text{LHDestRec}(Y) \leftarrow \text{LHDestRec}(X), \text{Flight}(\text{'LH'}, X, Y, -, -)$$

All destinations that can be reached from Frankfurt except those for which a Lufthansa-only connection exists.

$$\text{Destination}(X) \leftarrow \text{DestRec}(X), \neg \text{LHDestRec}(X)$$

Datalog: Example Queries

Which destinations can be reached from Frankfurt (FFT)?

$$\text{DestRec}(X) \leftarrow \text{Flight}(-, \text{'FFT'}, X, -, -)$$
$$\text{DestRec}(Y) \leftarrow \text{DestRec}(X), \text{Flight}(-, X, Y, -, -)$$

Which destinations can be reached from Frankfurt (FFT) taking only Lufthansa (LH) flights?

$$\text{LHDestRec}(X) \leftarrow \text{Flight}(\text{'LH'}, \text{'FFT'}, X, -, -)$$
$$\text{LHDestRec}(Y) \leftarrow \text{LHDestRec}(X), \text{Flight}(\text{'LH'}, X, Y, -, -)$$

All destinations that can be reached from Frankfurt except those for which a Lufthansa-only connection exists.

$$\text{Destination}(X) \leftarrow \text{DestRec}(X), \neg \text{LHDestRec}(X)$$

Datalog: Example Queries

Which destinations can be reached from Frankfurt (FFT)?

$$\text{DestRec}(X) \leftarrow \text{Flight}(-, \text{'FFT'}, X, -, -)$$
$$\text{DestRec}(Y) \leftarrow \text{DestRec}(X), \text{Flight}(-, X, Y, -, -)$$

Which destinations can be reached from Frankfurt (FFT) taking only Lufthansa (LH) flights?

$$\text{LHDestRec}(X) \leftarrow \text{Flight}(\text{'LH'}, \text{'FFT'}, X, -, -)$$
$$\text{LHDestRec}(Y) \leftarrow \text{LHDestRec}(X), \text{Flight}(\text{'LH'}, X, Y, -, -)$$

All destinations that can be reached from Frankfurt except those for which a Lufthansa-only connection exists.

$$\text{Destination}(X) \leftarrow \text{DestRec}(X), \neg \text{LHDestRec}(X)$$

Datalog: Basics

Additional Definitions

Consider a Datalog rule of the form

$$H(\bar{x}) \leftarrow L_1(\bar{x}, \bar{y}_1), \dots, L_n(\bar{x}, \bar{y}_n)$$

- If $n = 0$, then its body is empty and the rule is called *fact*
- Relation symbols that appear solely on the body of rules are called *extensional*; the remaining relational symbols are called *intensional*
- Accordingly, we distinguish between the extensional database (EDB) and the intensional database (IDB)

Datalog: Basics

From Datalog Rules to Datalog Programs

- A set of rules ρ is called *program* Π .
- Let Π be a program. The *dependency graph* of Π is a directed, labeled digraph (V, E) containing two types of edges (positive and negative edges) defined as follows:
 - V is the set of relational symbols appearing in the rules of ρ
 - Let P be a relational symbol of a positive literal appearing in the body of some rule ρ in Π and let Q be the relational symbol of ρ 's head. Then the (positive) edge $P \rightarrow Q$ is contained in E .
 - Let P be the relational symbol of a negative literal appearing in the body of some rule ρ in Π and let Q be the relational symbol of ρ 's head. Then the (negative) edge $P \xrightarrow{-} Q$ is contained in E .
- We call a program *recursive*, if its dependency graph has a cycle.
- Note: the definition of recursiveness ignores edge labels; we will come back to these labels at a later point

Datalog: Basics

Example: Dependency Graph

Consider the Datalog program Π defined by the rules

$$\begin{aligned} \text{DestRec}(X) &\leftarrow \text{Flight}(-, \text{'FFT'}, X, -, -) \\ \text{DestRec}(Y) &\leftarrow \text{DestRec}(X), \text{Flight}(-, X, Y, -, -) \\ \text{NotDestRec}(X) &\leftarrow \text{City}(X), \neg \text{DestRec}(X) \end{aligned}$$

Then the dependency graph of Π is defined as $G := (V, E)$, where

$$\begin{aligned} V &:= \{ \text{DestRec}, \text{Flight}, \text{City}, \text{NotDestRec} \}, \\ E &:= \{ \text{Flight} \longrightarrow \text{DestRec}, \text{DestRec} \longrightarrow \text{DestRec}, \\ &\quad \text{City} \longrightarrow \text{NotDestRec}, \text{DestRec} \overset{\neg}{\longrightarrow} \text{NotDestRec} \}. \end{aligned}$$

This program is recursive (cycle: $\text{DestRec} \longrightarrow \text{DestRec}$).

Datalog: Basics

Definition: Active Domain

- Recall: the active domain of an instance \mathcal{I} , $adm(\mathcal{I})$, is defined as the set of all constants appearing in \mathcal{I} .
- The active domain of a Datalog program Π w.r.t. input \mathcal{I} , $adm(\Pi, \mathcal{I})$, is the set of all constants appearing in Π and \mathcal{I} .

Datalog: Basics

Safe Datalog

- Let Π be a Datalog program and let \mathcal{I}_E be an instance of extensional relational symbols (input) and \mathcal{I}_A be an instance of the intensional relational symbol (output, i.e. set of answers).
- Typically: given input \mathcal{I}_E we are interested in the output \mathcal{I}_A
- A rule is called *safe* if every variable appears in a positive literal in its body

Lemma

Let Π be a Datalog program. If every rule of Π is safe and \mathcal{I}_E is finite, then the output \mathcal{I}_A of Π is finite.

Datalog: Basics

Example: Safe Datalog Program

```
DestRec(X) ← Flight(–, 'FFT', X, – , –)  
DestRec(Y) ← DestRec(X), Flight(–, X, Y, – , –)
```

Example: Non-safe Datalog Program

```
Goal(X) ← Flight(–, 'FFT', Y, – , –)
```

Example: Non-safe Datalog Program

```
Goal(X) ← ¬ Flight(–, 'FFT', X, – , –)
```

Note

- The result of non-safe Datalog programs may depend on the underlying domain (we may evaluate them w.r.t. the active domain)

Datalog: Basics

Example: Safe Datalog Program

```
DestRec(X) ← Flight(–, 'FFT', X, – , –)  
DestRec(Y) ← DestRec(X), Flight(–, X, Y, – , –)
```

Example: Non-safe Datalog Program

```
Goal(X) ← Flight(–, 'FFT', Y, – , –)
```

Example: Non-safe Datalog Program

```
Goal(X) ← ¬ Flight(–, 'FFT', X, – , –)
```

Note

- The result of non-safe Datalog programs may depend on the underlying domain (we may evaluate them w.r.t. the active domain)

Roadmap

Outlook: Datalog

In the following, we study different fragments of Datalog:

Name	Informal Description
Datalog ⁺	Positive Datalog, i.e. Datalog without negated literals
Datalog [¬]	Datalog as defined before, i.e. literals in the body may contain negated subgoals
Datalog ^{¬¬}	An extension of Datalog [¬] , where we also allow head predicates to be negated
Stratified Datalog	An important subclass of Datalog [¬] , obtained from a syntactic restriction (will be defined later)
NR-Datalog [¬]	Non-recursive Datalog with negation

We study fundamental properties of these fragments, discuss different semantics, evaluation techniques, and relations between these fragments and relational algebra.

Background Reading

Literature

S. Abiteboul, R. Hull, V. Vianu: *Foundations of Databases*.

- *Part D – Chapter 12: Datalog* covers most of our study of the Datalog⁺ fragment
- *Part D – Chapter 15: Negation in Datalog* covers semantics for Datalog[−] fragments

Note: the book is freely available for download at

<http://www.inf.unibz.it/~nutt/FDBs0809/>,

(username and password for downloading the files are provided at the bottom of the page).

Datalog⁺

Definition: Datalog⁺

We start with positive Datalog, called Datalog⁺, which is the subset of full Datalog defined as follows

- A positive rule is an implication of the form

$$H(\bar{x}) \leftarrow G_1(\bar{x}, \bar{y}_1), \dots, G_n(\bar{x}, \bar{y}_n), \text{ where}$$

- $n \geq 0$,
 - H, G_1, \dots, G_n are relational symbols,
 - $\bar{x}, \bar{y}_1, \dots, \bar{y}_n$ are vectors of variables, and
 - $R(\bar{z})$ denotes an atomic formula over variables from \bar{z} (i.e., a formula using a subset of the variables in \bar{z}).
- A set of safe positive rules is called Datalog⁺ program

Evaluation of Datalog⁺ Programs

Naive Evaluation Algorithm for Datalog⁺ Programs

Goal: compute the output \mathcal{I}_A of some Datalog⁺ program Π w.r.t. the input \mathcal{I}_E

- (1) Begin: initialize the relations of the intensional relational symbols R with \emptyset , i.e. $\mathcal{I}_A^0(R) = \emptyset$. Put $j := 0$.
- (2) (2a) Put $j := j + 1$.
 Let ρ be a rule from Π of the form $H \leftarrow G$, where $H = R(a_1, \dots, a_k)$ with variables or constants a_i ($1 \leq i \leq k$). Put

$$\mathcal{I}_\rho(R) := \{(\nu(a_1), \dots, \nu(a_k)) \mid (\mathcal{I}_E \cup \mathcal{I}_A^{j-1}) \models_\nu G, \nu \text{ is a variable assignment for } G\}$$
- (2b) Let R be an intensional relational symbol and let $\rho_1^R, \dots, \rho_l^R$ be the rules having predicate R in their head. Put

$$\mathcal{I}_A^j(R) := \bigcup_{i=1}^l \mathcal{I}_{\rho_i}(R)$$
- (2) Repeat step (2) until $\mathcal{I}_A^j(R) = \mathcal{I}_A^{j-1}(R)$ for all intensional relational symbols R .

Evaluation of Datalog⁺ Programs

Examples: Naive Evaluation Algorithm

- $Dest('FFT', X) \leftarrow Flight(-, 'FFT', X, -, -)$

j	$\mathcal{I}_A (Dest)$
0	\emptyset
1	$\{(FFT, BER), (FFT, DAL)\}$
2	$\{(FFT, BER), (FFT, DAL)\}$

- $DestRec(X) \leftarrow Flight(-, 'FFT', X, -, -)$
 $DestRec(Y) \leftarrow DestRec(X), Flight(-, X, Y, -, -)$

j	$\mathcal{I}_A (DestRec)$
0	\emptyset
1	$\{BER, DAL\}$
2	$\{BER, DAL, LON\}$
3	$\{BER, DAL, LON, NY\}$
4	$\{BER, DAL, LON, NY\}$

Evaluation of Datalog⁺ Programs

Proposition

Given a Datalog⁺ program Π , the naive evaluation algorithm always terminates.

Proof Sketch

Follows from the observation that the computation in steps (2a) and (2b) is monotonic and the finiteness of the output (Datalog⁺ is safe).

Datalog⁺ 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
 - ⇒ leads to a model-theoretic or proof-theoretic semantics
- Datalog rules are operational rules that can be used to calculate the answer set.
 - ⇒ 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 for Datalog⁺

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

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

Model-theoretic Semantics for Datalog⁺

Preliminaries (ctd)

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

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

and let \mathcal{I} be an instance over \mathcal{R} .

- \mathcal{I} satisfies ρ , if for each assignment ν of the variables in ρ :

$$\nu(G_1), \dots, \nu(G_k) \in \mathcal{I} \implies \nu(H) \in \mathcal{I}$$

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

Model-theoretic Semantics for Datalog⁺

Example

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

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

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

$$\begin{aligned}\rho_1 &: S(X) \leftarrow T(X), \\ \rho_2 &: T(X) \leftarrow S(X).\end{aligned}$$


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

Model-theoretic Semantics for Datalog⁺

Preliminaries (ctd)

Let Π be a Datalog program, let \mathcal{R} be a database schema that contains exactly the relational symbols appearing in Π ,¹ 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 .

¹From now on, we call such a schema *matching* database schema 

Model-theoretic Semantics for Datalog⁺

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} .

Remark

The definition is not Datalog⁺-specific. Concerning Datalog⁺ programs, though, we will later show that the minimal model is unique.

Lemma

Let Π be a Datalog⁺ program. Whenever $\Pi(\mathcal{I})$ exists, then it holds that $\text{adom}(\Pi(\mathcal{I})) \subseteq \text{adom}(\Pi, \mathcal{I})$. Thus, any model of Π contains only constants from $\text{adom}(\Pi, \mathcal{I})$.

Model-theoretic Semantics for Datalog⁺

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} .

Remark

The definition is not Datalog⁺-specific. Concerning Datalog⁺ programs, though, we will later show that the minimal model is unique.

Lemma

Let Π be a Datalog⁺ program. Whenever $\Pi(\mathcal{I})$ exists, then it holds that $\text{adom}(\Pi(\mathcal{I})) \subseteq \text{adom}(\Pi, \mathcal{I})$. Thus, any model of Π contains only constants from $\text{adom}(\Pi, \mathcal{I})$.

Model-theoretic Semantics for Datalog⁺

Constructing a Model for Datalog⁺ Programs

We can construct a model $\mathcal{M}(\Pi, \mathcal{I})$ of a Datalog⁺ program Π containing \mathcal{I} as follows.

- For the extensional relations, choose \mathcal{I}
- For the intensional relations, choose $adom(\Pi, \mathcal{I}) \times \dots \times adom(\Pi, \mathcal{I})$, depending on the arity of the respective input relation

Note: the so-constructed model is not minimal in the general case

Model-theoretic Semantics for Datalog⁺

Theorem

Let Π be a Datalog⁺ program with input \mathcal{I} . Further let \mathcal{X} be the set of all models of Π that contain \mathcal{I} . Then $\Pi(\mathcal{I})$ exists and $\Pi(\mathcal{I}) = \cap \mathcal{X}$.

Proof

It holds that $\mathcal{M}(\Pi, \mathcal{I}) \in \mathcal{X}$, i.e. there is at least one model.

Now let $H \leftarrow G_1, \dots, G_k$ be a rule ρ of Π and ν a variable assignment. Then it holds that

$$\nu(G_1), \dots, \nu(G_k) \in \cap \mathcal{X} \implies \nu(H) \in \cap \mathcal{X},$$

i.e. $\cap \mathcal{X}$ satisfies ρ .

By definition, every model in \mathcal{X} contains the input \mathcal{I} . Therefore, $\cap \mathcal{X}$ is a model of Π that contains \mathcal{I} . By construction, it is unique and minimal.

Fixpoint Semantics for Datalog⁺

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 defined by the following two rules.
 - $A \in \mathcal{I}(R)$ for some EDB relation R
 - There is an instantiation $A \leftarrow A_1, \dots, A_n$ of a rule from Π such that $A_i \in \mathcal{I}$ for $1 \leq i \leq n$.

Fixpoint Semantics for Datalog⁺

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.

- An instance \mathcal{I} is a model of Π if and only if $T_\Pi(\mathcal{I}) \subseteq \mathcal{I}$. (*)

Fixpoint Semantics for Datalog⁺

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 $\Pi(\mathcal{I})$.

Proof

$\Pi(\mathcal{I})$ is fixpoint of T_Π , because

- $T_\Pi(\Pi(\mathcal{I})) \subseteq \Pi(\mathcal{I})$, since $\Pi(\mathcal{I})$ is a model of Π .
- $T_\Pi(\Pi(\mathcal{I})) \subseteq \Pi(\mathcal{I})$ implies that $T_\Pi(T_\Pi(\Pi(\mathcal{I}))) \subseteq T_\Pi(\Pi(\mathcal{I}))$.

Therefore, it follows from (*) that $T_\Pi(\Pi(\mathcal{I}))$ is a model of Π .

Since $\Pi(\mathcal{I})$ is a minimal model, it follows that $\Pi(\mathcal{I}) \subseteq T_\Pi(\Pi(\mathcal{I}))$.

Recall that every fixpoint is also a model. Therefore T_Π must have a minimal fixpoint, which corresponds exactly to the minimal model $\Pi(\mathcal{I})$.

Fixpoint Semantics for Datalog⁺

Computation of the Minimal Fixpoint

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

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

- 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 $T_{\Pi}^{\infty}(\mathcal{I})$.

Fixpoint Semantics for Datalog⁺

Theorem

Let Π be a Datalog⁺ program. Then $T_{\Pi}^{\infty}(\mathcal{I}) = \Pi(\mathcal{I})$, i.e. $T_{\Pi}^{\infty}(\mathcal{I})$ is the minimal fixpoint containing \mathcal{I} .

Proof

Let \mathcal{J} be an arbitrary fixpoint of T_{Π} that contains \mathcal{I} . Then it holds that

$$\begin{aligned}\mathcal{J} &\supseteq T_{\Pi}^0(\mathcal{I}) = \mathcal{I}, \\ \mathcal{J} &\supseteq T_{\Pi}^i(\mathcal{I}), i \geq 0, \\ \mathcal{J} &\supseteq T_{\Pi}^{\infty}(\mathcal{I}).\end{aligned}$$

Remark

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

Fixpoint Semantics for Datalog⁺

Theorem

Let Π be a Datalog⁺ program. Then $T_{\Pi}^{\infty}(\mathcal{I}) = \Pi(\mathcal{I})$, i.e. $T_{\Pi}^{\infty}(\mathcal{I})$ is the minimal fixpoint containing \mathcal{I} .

Proof

Let \mathcal{J} be an arbitrary fixpoint of T_{Π} that contains \mathcal{I} . Then it holds that

$$\begin{aligned}\mathcal{J} &\supseteq T_{\Pi}^0(\mathcal{I}) = \mathcal{I}, \\ \mathcal{J} &\supseteq T_{\Pi}^i(\mathcal{I}), i \geq 0, \\ \mathcal{J} &\supseteq T_{\Pi}^{\infty}(\mathcal{I}).\end{aligned}$$

Remark

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

Proof-theoretic Semantics for Datalog⁺

Definition Proof Tree

Let A be a fact. A *proof tree* of A w.r.t. a Datalog program Π and input \mathcal{I} is a labeled tree satisfying the following properties:

- (1) Every node of the tree is labeled with a fact
- (2) Every leaf node of the tree is labeled with a fact from \mathcal{I}
- (3) The root is labeled with fact A
- (4) For every inner node of the tree there is an instantiation of a rule from Π of the form

$$A_1 \leftarrow A_2, \dots, A_n$$

such that

- the node is labeled with A_1 ,
- the children of the nodes are labeled with A_2, \dots, A_n .

Proof-theoretic Semantics for Datalog⁺

Theorem

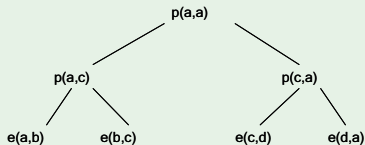
Let Π be a Datalog program with input \mathcal{I} and let A be a fact. It holds that $A \in \Pi(\mathcal{I})$ if and only if there exists a proof tree for A w.r.t. Π and \mathcal{I} .

Example

$\Pi :$ $p(X, Y) \leftarrow e(X, Z), e(Z, Y)$
 $p(X, Y) \leftarrow p(X, Z), p(Z, Y)$

$e :=$

a	b
b	c
c	d
d	a



Summary: Semantics for Datalog⁺

Central Results

- We have studied three semantics for Datalog⁺
 - The model-theoretic semantics
 - The answer of a Datalog program is its minimal model
 - This minimal model is unique for Datalog⁺
 - The fixpoint semantics
 - Derives new facts until a fixpoint is reached
 - This fixpoint always exists for Datalog⁺ programs and is unique
 - The proof-theoretic semantics
 - Defines the output as the set of facts that can be proven
- Key observations
 - For Datalog⁺, all three semantics coincide
 - For Datalog⁺, all three semantics coincide to the naive evaluation algorithm

Datalog⁺: Boundedness and Recursion

Definition: Boundedness

A Datalog⁺ program Π is *bounded* if there is a constant d such that for every input \mathcal{I} it holds that

$$T_{\Pi}^d(\mathcal{I}) = T_{\Pi}^{d+1}(\mathcal{I}).$$

Theorem

In the general case it is undecidable if a Datalog⁺ program is bounded.

Datalog⁺: Boundedness and Recursion

Example: Boundedness

The Datalog⁺ program

$$\begin{aligned} Buys(X, Y) &\leftarrow Trendy(X), Buys(Z, Y) \\ Buys(X, Y) &\leftarrow Likes(X, Y) \end{aligned}$$

is bounded, but the program

$$\begin{aligned} Buys(X, Y) &\leftarrow Knows(X, Z), Buys(Z, Y) \\ Buys(X, Y) &\leftarrow Likes(X, Y) \end{aligned}$$

is not bounded.

Proposition

Let Π be a Datalog⁺ program. If Π is bounded, then there exists a finite equivalent Datalog program that is not recursive.

Datalog⁺: Boundedness and Recursion

Example

$$\text{Buys}(X, Y) \leftarrow \text{Trendy}(X), \text{Buys}(Z, Y)$$

$$\text{Buys}(X, Y) \leftarrow \text{Likes}(X, Y)$$

≡

$$\text{Buys}(X, Y) \leftarrow \text{Likes}(X, Y)$$

$$\text{Buys}(X, Y) \leftarrow \text{Trendy}(X), \text{Likes}(Z, Y)$$

Example

$$\text{Buys}(X, Y) \leftarrow \text{Knows}(X, Z), \text{Buys}(Z, Y)$$

$$\text{Buys}(X, Y) \leftarrow \text{Likes}(X, Y)$$

≡

$$\text{Buys}(X, Y) \leftarrow \text{Likes}(X, Y)$$

$$\text{Buys}(X, Y) \leftarrow \text{Knows}(X, Z), \text{Likes}(Z, Y)$$

$$\text{Buys}(X, Y) \leftarrow \text{Knows}(X, Z), \text{Knows}(Z, Z_1), \text{Likes}(Z_1, Y)$$

$$\text{Buys}(X, Y) \leftarrow \text{Knows}(X, Z), \text{Knows}(Z, Z_1), \text{Knows}(Z_1, Z_2), \text{Likes}(Z_2, Y)$$

⋮



Datalog⁺: Boundedness and Recursion

Example

$$\begin{aligned} \text{Buys}(X, Y) &\leftarrow \text{Trendy}(X), \text{Buys}(Z, Y) \\ \text{Buys}(X, Y) &\leftarrow \text{Likes}(X, Y) \end{aligned}$$

≡

$$\begin{aligned} \text{Buys}(X, Y) &\leftarrow \text{Likes}(X, Y) \\ \text{Buys}(X, Y) &\leftarrow \text{Trendy}(X), \text{Likes}(Z, Y) \end{aligned}$$

Example

$$\begin{aligned} \text{Buys}(X, Y) &\leftarrow \text{Knows}(X, Z), \text{Buys}(Z, Y) \\ \text{Buys}(X, Y) &\leftarrow \text{Likes}(X, Y) \end{aligned}$$

≡

$$\begin{aligned} \text{Buys}(X, Y) &\leftarrow \text{Likes}(X, Y) \\ \text{Buys}(X, Y) &\leftarrow \text{Knows}(X, Z), \text{Likes}(Z, Y) \\ \text{Buys}(X, Y) &\leftarrow \text{Knows}(X, Z), \text{Knows}(Z, Z_1), \text{Likes}(Z_1, Y) \\ \text{Buys}(X, Y) &\leftarrow \text{Knows}(X, Z), \text{Knows}(Z, Z_1), \text{Knows}(Z_1, Z_2), \text{Likes}(Z_2, Y) \end{aligned}$$

⋮



Containment of Datalog⁺ Programs

Definition: Containment of Datalog Programs

Let Π , Π' be two Datalog programs containing the same extensional relations and a common intensional relation \mathcal{T} .

Π is contained in Π' w.r.t. \mathcal{T} , $\Pi \sqsubseteq_{\mathcal{T}} \Pi'$, if for every input \mathcal{I} it holds that

$$\Pi(\mathcal{I})(\mathcal{T}) \subseteq \Pi'(\mathcal{I})(\mathcal{T}).$$

Π and Π' are equivalent w.r.t. \mathcal{T} , $\Pi \equiv_{\mathcal{T}} \Pi'$, iff $\Pi \sqsubseteq_{\mathcal{T}} \Pi'$ and $\Pi' \sqsubseteq_{\mathcal{T}} \Pi$.

Containment of Datalog⁺ Programs: Decidability

Theorem

The containment problem for two Datalog⁺ programs Π , Π' (i.e., the question whether $\Pi \sqsubseteq_T \Pi'$ for some relation T holds or not) is undecidable in the general case.

Proof Idea

By a reduction of the containment problem for context-free grammars.

Theorem

The containment relationship $\Pi \sqsubseteq_T \Pi'$ of two Datalog⁺ programs Π , Π' w.r.t. T (and hence, also the $\Pi \equiv_T \Pi'$ relationship) is decidable if

- Π, Π' can be expressed as finite sets of conjunctive queries (e.g., non-recursive Datalog⁺ programs consisting of a single rule), or
- Π can be expressed as a finite set of conjunctive queries and Π' is a Datalog program.

Containment of Datalog⁺ Programs: Decidability

Theorem

The containment problem for two Datalog⁺ programs Π , Π' (i.e., the question whether $\Pi \sqsubseteq_T \Pi'$ for some relation T holds or not) is undecidable in the general case.

Proof Idea

By a reduction of the containment problem for context-free grammars.

Theorem

The containment relationship $\Pi \sqsubseteq_T \Pi'$ of two Datalog⁺ programs Π , Π' w.r.t. T (and hence, also the $\Pi \equiv_T \Pi'$ relationship) is decidable if

- Π, Π' can be expressed as finite sets of conjunctive queries (e.g., non-recursive Datalog⁺ programs consisting of a single rule), or
- Π can be expressed as a finite set of conjunctive queries and Π' is a Datalog program.

Datalog⁺ vs. Conjunctive Queries

Lemma

For every Datalog⁺ program Π there exists an equivalent (not necessarily finite) set of conjunctive queries.

Proof Sketch

First apply the algorithm `EXPAND` (see next slide) to program Π . Let T be an IDB relation. Further let $\mathcal{S}(T)$ be the set of all conjunctive queries in \mathcal{S} that define T and contain only EDB relations in their body.

Then for every input \mathcal{I} it holds that $\Pi(\mathcal{I})(T) = \mathcal{S}(T)(\mathcal{I})$.

Datalog⁺ vs. Conjunctive Queries

Transformation of a Datalog⁺ Program Π into a set of CQs

Let T be a relevant IDB relation. Initialize \mathcal{S} with the rules from Π defining T ; each of these rules is denoted as *candidate conjunctive query*. The set of conjunctive queries that is equivalent to Π results from the execution of algorithm EXPAND (which expands the set \mathcal{S}) defined as follows.

Algorithm EXPAND:

repeat forever

for each previously unconsidered potential conjunctive query Q in \mathcal{S} do

for each IDB subgoal $R(t_1, \dots, t_k)$ in the body of Q do

for each rule ρ for R do begin

 rename variables of ρ so there are none in common with Q ;

 unify the head H of ρ with $R(t_1, \dots, t_k)$ to get MGU τ ;

 /* MGU τ is the most general substitution such that */

 /* $\tau(H)$ and $\tau(R(t_1, \dots, t_k))$ become syntactically equivalent. */

 add to \mathcal{S} the potential conjunctive query formed from $\tau(Q)$ by

 replacing $\tau(R(t_1, \dots, t_k))$ by τ applied to the body of ρ

end

end EXPAND

Containment Testing

Containment of Conjunctive Queries in Datalog⁺ Programs

■ First Idea:

Expand the Datalog⁺ program into a set of conjunctive queries \mathcal{S} and test containment after each expansion step of \mathcal{S} .

⇒ drawback: using this scheme we cannot determine in finite time if the containment relationship does **not** hold.

■ Second Idea:

For each conjunctive query Q compute the canonical instance \mathcal{I}_Q and consider the minimal model $\Pi(\mathcal{I}_Q)$ of the Datalog⁺ program Π . If it holds that $\tau(R(\vec{U})) \in \Pi(\mathcal{I}_Q)$, where τ is the canonical substitution and $R(\vec{U})$ is the head of Q , then Q is contained in Π .

Containment Testing

Lemma

Let Q be a conjunctive query with head $R(\vec{U})$, canonical instance \mathcal{I}_Q , and associated canonical substitution τ . It holds that $Q \sqsubseteq_R \Pi$ if and only if $\tau(R(\vec{U})) \in \Pi(\mathcal{I}_Q)$.

Proof

If $\tau(R(\vec{U})) \notin \Pi(\mathcal{I}_Q)$, then $Q \not\sqsubseteq_R \Pi$. Assume that $\tau(R(\vec{U})) \in \Pi(\mathcal{I}_Q)$.

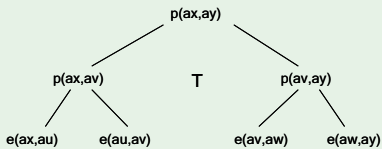
- Let T be an associated proof tree.
- Let \mathcal{I} be an instance and ϕ be a substitution of the variables in Q that satisfies Q w.r.t. \mathcal{I} ; the answer is $\phi(R(\vec{U}))$.
- Replace every constant a in T by $\phi(\tau^{-1}(a))$ and denote the resulting tree as T' . It remains to show that T' is a proof tree for Π w.r.t. \mathcal{I} .
- For every inner node N in T there exists a rule ρ and an instantiation ψ of ρ . Then consider the instantiation of ρ w.r.t. the node N and T' of the form $\phi(\tau^{-1}(\psi(\rho)))$.
- If N is a leaf with label $\tau(G_i)$ in T , then we obtain the label $\phi(\tau^{-1}(\tau(G_i)))$ for N in T' , i.e. $\phi(G_i)$. It holds that $\phi(G_i) \in \mathcal{I}$, so we can conclude that T' is a proof tree.
- The root of T carries the label $\tau(R(\vec{U}))$ and therefore the root of T' carries the label $\phi(\tau^{-1}(\tau(R(\vec{U}))))$, i.e. $\phi(R(\vec{U}))$.
- Consequently, every answer of Q over \mathcal{I} has a proof tree w.r.t. Π .

Datalog Program and Instance

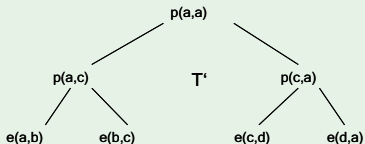
$$\Pi : \begin{aligned} p(X, Y) &\leftarrow e(X, Z), e(Z, Y) \\ p(X, Y) &\leftarrow p(X, Z), p(Z, Y) \end{aligned}$$

$$Q : p(X, Y) \leftarrow e(X, U), e(U, V), e(V, W), e(W, Y)$$

Example: Containment Testing



e	
a_x	a_u
a_u	a_v
a_v	a_w
a_w	a_y



e	
a	b
b	c
c	d
d	a

Datalog[¬]: Inflationary Semantics

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 now 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

Datalog[¬]: Inflationary Semantics

Example: Datalog[¬]

Let the relation G represent the edge relation of a directed graph. The following Datalog[¬] 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' . We thereby assume the distance of unconnected nodes to be ∞ .

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

Datalog[¬]: Inflationary Semantics

Example: Datalog[¬]

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

$$\begin{aligned}T(X, Y) &\leftarrow G(X, Y) \\T(X, Y) &\leftarrow T(X, Z), G(Z, Y) \\Compl(X, Y) &\leftarrow \neg T(X, Y)\end{aligned}$$

Datalog[¬]: Inflationary Semantics

Definition: Immediate Consequence Operator for Datalog[¬]

Let Π be a Datalog[¬] 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
- $A \leftarrow L_1, \dots, L_m$ 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.

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

Datalog⁻: Inflationary Semantics

Inflationary Fixpoint

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

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

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).

Datalog⁻: Inflationary Semantics

Example: Fixpoint and Minimal Models of Datalog⁻

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):

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

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

Datalog⁻: Inflationary Semantics

Example

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

$$T(X, Y) \leftarrow G(X, Y)$$

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

$$\text{old}T(X, Y) \leftarrow T(X, Y)$$

$$\text{old}T\text{exceptFinal}(X, Y) \leftarrow T(X, Y), T(X', Z'), T(Z', Y'), \neg T(X', Y')$$

$$CT(X, Y) \leftarrow \neg T(X, Y), \text{old}T(X', Y'), \neg \text{old}T\text{exceptFinal}(X', Y')$$

Datalog[¬]: Non-inflationary Semantics

Datalog[¬]

- We now consider Datalog programs with negation in both body and head
- Idea: negation in the head of rules removes facts that have been derived in previous steps
- This variant of Datalog is called *Datalog[¬]*

Datalog[¬]: Non-inflationary Semantics

Immediate Consequence Operator for Datalog[¬]

Let Π be a Datalog[¬] program and \mathcal{K} an instance over the relational schema of Π

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

- L is positive and $L \in \mathcal{K}(R)$ for some EDB relation R , or
- $L \leftarrow L_1, \dots, L_m$ is an instantiation (w.r.t. the active domain) of a rule in 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' for Π , Γ'_Π , is defined as follows.

$$\Gamma'_\Pi(\mathcal{K}) := \mathcal{K} \cup \begin{cases} \{A \mid L := A \text{ and } L \text{ is an immediate consequence of } \Pi \text{ w.r.t. } \mathcal{K}\} \\ \setminus \{A \mid L := \neg A \text{ and } L \text{ is an immediate consequence of } \Pi \text{ w.r.t. } \mathcal{K}, \\ \text{but } A \text{ is not an immediate consequence}\} \end{cases}$$

Datalog[¬]: Non-inflationary Semantics

Termination

Let \mathcal{I} be an input for Datalog[¬] program Π . The sequence

$$\Gamma'_{\Pi}(\mathcal{I}), \Gamma'^2_{\Pi}(\mathcal{I}), \Gamma'^3_{\Pi}(\mathcal{I}) \dots$$

does not necessarily have a fixpoint, i.e. the evaluation of Π is not guaranteed to terminate.

Example: Non-terminating Sequence

The following Datalog[¬] program does not terminate for input $T(0)$.

$$\begin{aligned} T(0) &\leftarrow T(1) \\ \neg T(1) &\leftarrow T(1) \\ T(1) &\leftarrow T(0) \\ \neg T(0) &\leftarrow T(0) \end{aligned}$$

Datalog[⊃]: Stratified Semantics

We extend the applicability of the T -operator to Datalog[⊃] (not Datalog^{⊃⊃}) and denote the resulting operator as $T^\#$.

Operator $T^\#$

Let Π be a Datalog[⊃] 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 $A \leftarrow L_1, \dots, L_n$ 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.

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

Datalog[⊖]: Stratified Semantics

Discussion

- 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 $\{T_{\Pi}^{\#,i}(\emptyset)\}_{i>0}$ 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 terminating iterative computation.

Datalog[¬]: Stratified Semantics

Discussion (ctd)

- Consider the program Π :

$$\begin{aligned} p &\leftarrow \neg q \\ q &\leftarrow \neg p \\ p &\leftarrow \neg p, q \end{aligned}$$

The iterative fixpoint computation does not terminate, although $\{p\}$ is the minimal fixpoint.

- Consider Π :

$$\begin{aligned} p &\leftarrow true \\ q &\leftarrow \neg p \\ q &\leftarrow q \end{aligned}$$

The iterative fixpoint computation converges to $\{p, q\}$, but $\{p\}$ is the minimal fixpoint here.

⇒ iteration of the $T^\#$ operator is not a satisfactory solution

Datalog[⊃]: Stratified Semantics

Remark

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

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

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.

Datalog[¬]: 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[¬] 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 $\{\Pi^1, \Pi^2, \dots, \Pi^n\}$ 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 .

Datalog[⊃]: Stratified Semantics

Evaluation of Stratified Programs

- The (stratified) semantics of a stratified Datalog[⊃] program is defined by a stratified composition of minimal models, where each minimal model can be computed 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.

Datalog[⊃]: Stratified Semantics

Stratified Semantics

Let $\sigma := \Pi^1, \Pi^2, \dots, \Pi^n$ be a stratification of some Datalog[⊃] 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* $\Pi^{\text{strat}}(\mathcal{E})$ 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} .

Datalog[⊃]: Stratified Semantics

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.

⇒ due to this reason, the stratified semantics is generally accepted as a semantics for stratified Datalog[⊃] programs

Datalog⁻: Stratified Semantics

Example

Consider the program

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

and input

<i>green</i>	
1	2

<i>red</i>	
1	2
2	3

The following two models are minimal.

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

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

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

Datalog[¬]: Stratified Semantics

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

NR-Datalog[¬]: Non-recursive Datalog with Negation

Non-recursive Datalog with Negation

- An *non-recursive Datalog[¬] program* (NR-Datalog[¬], for short), is a Datalog[¬] program whose dependency graph contains no cycle
- NR-Datalog[¬] is a subset of stratified Datalog: the dependency graph has no cycle and thus no cycle going through a negative edge
- We underlie the stratified semantics for evaluating NR-Datalog[¬]

Outlook

In this subsection we show that NR-Datalog[¬] and relational algebra have exactly the same expressive power, i.e. for every relational algebra query, there is an equivalent NR-Datalog[¬] program and vice versa.

NR-Datalog[¬]: Non-recursive Datalog with Negation

Non-recursive Datalog with Negation

- An *non-recursive Datalog[¬] program* (NR-Datalog[¬], for short), is a Datalog[¬] program whose dependency graph contains no cycle
- NR-Datalog[¬] is a subset of stratified Datalog: the dependency graph has no cycle and thus no cycle going through a negative edge
- We underlie the stratified semantics for evaluating NR-Datalog[¬]

Outlook

In this subsection we show that NR-Datalog[¬] and relational algebra have exactly the same expressive power, i.e. for every relational algebra query, there is an equivalent NR-Datalog[¬] program and vice versa.

Relational Algebra is Contained in NR-Datalog[¬]

Lemma (*₁)

For every relational algebra query there is an equivalent NR-Datalog[¬] program.

Proof Sketch

- Let $\mathcal{R} := \{R_1, \dots, R_n\}$ be a relational schema
- By $ar(R_i)$ we denote the arity of relational symbol R_i ($1 \leq i \leq n$).
- Given a relational symbol $R \in \mathcal{R}$, we denote the attributes associated to R as $att(R) := [A_1^R, \dots, A_{ar(R)}^R]$
- We introduce a function $pr(Q)$ that, for each relational algebra query Q , returns a relational symbol s.t. (i) for all Q : $pr(Q) \notin \mathcal{R}$ and (ii) for all $Q_1 \neq Q_2$ it holds that $pr(Q_1) \neq pr(Q_2)$

We consider algebra expressions built using selection, projection, join, rename, union, difference and inductively define a function $r2d$ that transforms them into NR-Datalog[¬] programs (i.e., sets of rules)

Relational Algebra is Contained in NR-Datalog[¬]

Proof Sketch (ctd)

- If $Q := R_i$ is a relational symbol, define

$$r2d(Q) := \{pr(Q)(A_1^R, \dots, A_{ar(n)}^R) \leftarrow R(A_1^R, \dots, A_{ar(n)}^R)\},$$

where we understand the symbols A_i^R as variables.

- If $Q := \pi_{A_1, \dots, A_m}(Q_1)$ is a projection expression, define

$$r2d(Q) := r2d(Q_1) \cup \{pr(Q)(A_1, \dots, A_m) \leftarrow head(r2d(Q_1))\}$$

- If $Q := Q_1 \bowtie Q_2$, let $\overline{X_1}$ ($\overline{X_2}$) denote the vector of variables appearing in $head(r2d(Q_1))$ ($head(r2d(Q_2))$) and define

$$r2d(Q) := r2d(Q_1) \cup r2d(Q_2) \cup \\ \{pr(Q)(\overline{X_1}, \overline{X_2}) \leftarrow head(r2d(Q_1)), head(r2d(Q_2))\}$$

Relational Algebra is Contained in NR-Datalog[⌊]

Proof Sketch (ctd)

- If $Q := \rho_{A_1 \mapsto A_2}(Q_1)$, let $\rho_{A_1 \mapsto A_2}(\overline{X_1})$ denote the vector of variables appearing in $head(r2d(Q_1))$, where each occurrence of A_1 has been replaced by A_2 , and let (ii) $\rho_{A_1 \mapsto A_2}(head(r2d(Q_1)))$ denote the formula obtained from $head(r2d(Q_1))$ obtained by replacing every occurrence of A_1 by A_2 , and define

$$r2d(Q) := r2d(Q_1) \cup \{pr(Q)(\rho_{A_1 \mapsto A_2}(\overline{X_1})) \leftarrow \rho_{A_1 \mapsto A_2}(head(r2d(Q_1)))\}$$

- If $Q := Q_1 \cup Q_2$, let $\overline{X_1}$ ($\overline{X_2}$) denote the vector of variables appearing in $head(r2d(Q_1))$ ($head(r2d(Q_2))$) and define

$$r2d(Q) := r2d(Q_1) \cup r2d(Q_2) \cup \{pr(Q)(\overline{X_1}) \leftarrow head(r2d(Q_1)), pr(Q)(\overline{X_2}) \leftarrow head(r2d(Q_2))\}$$

Relational Algebra is Contained in NR-Datalog[¬]

Proof Sketch (ctd)

- If $Q := Q_1 - Q_2$, let $\overline{X_1}$ denote the vector of variables appearing in $\text{head}(r2d(Q_1))$ and define

$$r2d(Q) := r2d(Q_1) \cup r2d(Q_2) \cup \\ \{pr(Q)(\overline{X_1}) \leftarrow pr(Q_1)(\overline{X_1}), \neg pr(Q_2)(\overline{X_1})\}$$

Closing remarks:

- By construction, the program $r2d(Q)$ is a NR-Datalog[¬] program
- It can be easily shown by induction that, given Q as input, the resulting Datalog program $r2d(Q)$ is equivalent to Q (i.e., the answer of Q and the result stored in $pr(Q)$ are identical)
- This concludes the proof of Lemma (*₁)

Relational Algebra is Contained in NR-Datalog[⊃]

Example: Translation of Relational Algebra into NR-Datalog[⊃]

Consider the schema $\mathcal{R} := \{R[A, B, C], S[C, D, E]\}$. We translate the relational algebra query $Q := \pi_A(R \bowtie S)$:

- For subquery $Q := R$ we obtain the rule

$$ans_R(A, B, C) \leftarrow R(A, B, C)$$

- For subquery $Q := S$ we obtain the rule

$$ans_S(C, D, E) \leftarrow S(C, D, E)$$

- For subquery $Q := R \bowtie S$, we obtain the rule

$$ans_{R \bowtie S}(A, B, C, C, D, E) \leftarrow ans_R(A, B, C), ans_S(C, D, E)$$

- For query Q we thus obtain: $ans_Q(A) \leftarrow ans_{R \bowtie S}(A, B, C, C, D, E)$

NR-Datalog[¬] is Contained in Relational Algebra

Lemma (*₂)

For every NR-Datalog[¬] query there is an equivalent relational algebra query.

To prove this lemma, we will now show...

... how to translate the NR-Datalog[¬] program into the range-restricted relational calculus. The lemma then follows from Codd's theorem (stating that relational algebra and the range-restricted relational calculus have the same expressive power).

Translation of NR-Datalog[¬] into the Relational Calculus

Example: Translation NR-Datalog[¬] into Relational Calculus

Consider the relations

- *Prof*[Name,Room,Phone]
- *Lecture*[Course,Name]
- *Examiner*[Topic,Name]

The following NR-Datalog[¬] program and the associated translation extract all professors that do neither teach nor examine:

Active(X) ← Lecture(−,X)	{<X> ∃Y ₁ , Y ₂ Prof(X, Y ₁ , Y ₂) ¬[∃Y ₁ Lecture(Y ₁ , X) ∨ ∃Y ₁ Examiner(Y ₁ , X)]}
Active(X) ← Examiner(−,X)	
ans(X) ← ¬ Active(X), Prof(X,−,−)	

Translation of NR-Datalog[¬] into the Relational Calculus

Lemma (*₃)

For each NR-Datalog[¬] program there is an equivalent relational calculus query.

Proof Sketch

- Let Π be a NR-Datalog[¬] program with intensional relations S_1, \dots, S_m , and rules $\rho_{i1}, \dots, \rho_{ik_i}$ for each $i \leq m$
- We inductively define
 - a formula φ_{ij} for each rule q_{ij} , and
 - a formula φ_i for each relation S_i
- Note: in the translation process described subsequently, we may obtain a (syntactically) extended version of Datalog rules containing equality atoms of the form $X = c$ or $X = Y$ in rule bodies

Translation of NR-Datalog[⊃] into the Relational Calculus

Proof Sketch (ctd)

- Let $\rho := \rho_{ij}$ be a rule of the form $S_i(U_1, \dots, U_l) \leftarrow L_1, \dots, L_p$, where the L_k are (possibly negated) literals (for $1 \leq k \leq p$)
- In a first step, transform ρ into a safe rule ρ' :
 - The head is translated into $S_i(X_1, \dots, X_l)$
 - For each $r \leq l$ we modify the rule as follows
 - If $U_r := c$ is a literal, add the literal $X_r = c$ to the body of the rule
 - If $U_r := x$ is a variable, replace x by X_r in the body of the rule
 - If $U_r := U_s$ for $s \neq r$, add a literal $X_r = X_s$ to the body of the rule

Example: Rule Transformation

The rule $\rho : S_2(X_2, 5, X_2, X_3) \leftarrow R_1(Y_1, X_2, X_3), R_2(Y_1, X_2, Y_2)$ is translated into $\rho' : S_2(X_1, X_2, X_3, X_4) \leftarrow R_1(Y_1, X_1, X_4), R_2(Y_1, X_1, Y_2), X_2 = 5, X_1 = X_3$

Translation of NR-Datalog[⊃] into the Relational Calculus

Proof Sketch (ctd)

The formula $\varphi_{ij}(x_1, \dots, x_l)$ is now obtained from ρ' as follows.

- All variables appearing only in the body of ρ' are existentially quantified
- All literals are interconnected through conjunction
- Atoms $S_r(\bar{v})$ are replaced by $\varphi_r(\bar{x}/\bar{v})$

Example: Translation of a Rule

Let R_1 be extensional and R_2 be intensional. Then the rule $\rho' : S_2(X_1, X_2, X_3, X_4) \leftarrow R_1(Y_1, X_1, X_4), R_2(Y_1, X_1, Y_2), X_2 = 5, X_1 = X_3$ is translated into

$$\varphi_{ij}(X_1, \dots, X_4) := \exists Y_1, Y_2 R_1(Y_1, X_1, X_4) \wedge \varphi_2(Y_1, X_1, Y_2) \wedge X_2 = 5 \wedge X_1 = X_3$$

Translation of NR-Datalog[⊃] into the Relational Calculus

Proof Sketch (ctd)

- Having a translation for rules, we're almost done and define

$$\varphi_i := \varphi_{i1} \vee \dots \vee \varphi_{ik_i}$$

- The relational calculus query $\{\langle X_1, \dots, X_l \rangle \mid \varphi_m\}$ is then equivalent to the program Π
- To complete the proof of Lemma (*₃) it remains to be shown that φ_m is domain-independent
- To this end, we show in the following that φ_m is range-restricted, which implies domain-independence

Translation of NR-Datalog[∩] into the Relational Calculus

Lemma (*₄)

- (a) For every NR-Datalog[∩] program Π there is a range-restricted formula φ such that for all matching databases D it holds that $\Pi(D) = q(D)$ with $q := \{\langle \bar{X} \rangle \mid \varphi\}$.
- (b) For every range-restricted formula φ the query $\{\langle \bar{X} \rangle \mid \varphi\}$ is range-restricted.

Note: This lemma, in combination with Codd's Theorem, implies Lemma (*₂)

Proof Sketch

Proof Idea:

- 1 We show that the formula constructed in the proof of Lemma (*₃) (i.e., the translation from NR-Datalog[∩] to the relational calculus) is range-restricted
- 2 By induction on the structure of φ

Translation of NR-Datalog[⊃] into the Relational Calculus

Proof of Part (a)

- Recall:
 - By $rr(\varphi)$ we denote the range-restricted variables of relational calculus formula φ (i.e., those variables that can take only values appearing in the database)
 - By $SRNF(\varphi)$ we denote the safe range normal form of φ
 - By $free(\varphi)$ we denote the free variables of φ
- Let Π be a NR-Datalog[⊃] program and let $S_i, \rho_{ij}, \rho'_{ij}, \varphi_i$ be defined as in the proof of the previous translation
- We prove by induction on i that for all $i \leq m$ it holds that $rr(SRNF(\varphi_i)) = free(\varphi_i)$, which shows that φ_i is range-restricted

Translation of NR-Datalog[¬] into the Relational Calculus

Proof of Part (a) (ctd)

Basic case: $i = 1$

- Every variable X_i appears in positive form in each rule ρ_{ij}
 - Recalling that every φ_{1j} is of the form $\exists \bar{Y} \bigwedge_p L_p$, we conclude that a variable is positive if it appears in at least one positive literal (or is connected with such a variable through equality atoms)
- ⇒ The free variables of φ_{1j} are contained in each set $rr(\varphi_{1j})$
- ⇒ $rr(SRNF(\varphi_{1j})) = free(\varphi_{1j})$
- ⇒ $rr(SRNF(\varphi_1)) = free(\varphi_1)$, which completes the basic case

Translation of NR-Datalog[¬] into the Relational Calculus

Proof of Part (a) (ctd)

Induction step: $i > 1$

- Now it may happen that X is positive in ρ_{ij} , because it appears in an intensional atom $S_r(\bar{V})$
 - Such an atom in φ_{ij} corresponds to a formula φ_r
 - By induction we have $rr(SRNF(\varphi_r)) = free(\varphi_r)$
 - It holds that every variable X_i appears in a positive literal or is connected with such a variable through equality atoms, so it follows (analogously to case $i = 1$) that the free variables of φ_{ij} are contained in each set $rr(\varphi_{ij})$
- ⇒ $rr(SRNF(\varphi_{ij})) = free(\varphi_{ij})$
- ⇒ $rr(SRNF(\varphi_i)) = free(\varphi_i)$, which completes the induction step

From the induction hypothesis it follows by definition that φ_m is range-restricted.

Translation of NR-Datalog[¬] into the Relational Calculus

Proof of Part (b)

To show that for each range-restricted formula φ , the query $\{\langle \bar{U} \rangle \mid \varphi\}$ is range-restricted we start with a range-restricted formula $\varphi(\bar{X})$ in SRNF with $rr(\varphi) \neq \perp$. We show by induction on the structure of φ that

- for each matching database D ,
- each set d s.t. $adom(\varphi, D) \subseteq d \subseteq dom$, and
- each variable assignment $\beta : free(\varphi) \mapsto d$

it holds that

- 1 if $X_i \in rr(\varphi)$ and $D \models_d \varphi[\beta]$, then $\beta(x_i) \in adom(\varphi, D)$, and
- 2 $D \models_d \varphi[\beta] \iff D \models_{adom(\varphi, D) \cup \beta(free(\varphi))} \varphi[\beta]$.

Basic case: the claims are easily verified for atomic formulas of the form $R(\bar{U})$, $X = c$, $X = Y$ (the last one follows from $rr(\varphi) = \emptyset$).

Proof of Part (b) (ctd)

Induction step: we first show that claim 1

if $X_i \in rr(\varphi)$ and $D \models_d \varphi[\beta]$, then $\beta(x_1) \in \text{adom}(\varphi, D)$

holds for composed formulas:

- $\varphi := \neg(\varphi_1)$ follows trivially by induction.
- $\varphi := \varphi_1 \vee \varphi_2$: follows by induction, because $rr(\varphi_1) = rr(\varphi_2) = rr(\varphi)$
- $\varphi := \varphi_1 \wedge \varphi_2$:
 - If φ_2 is of the form $X = Y$ and $rr(\varphi) \cap \{X, Y\} \neq \emptyset$
 - If $D \models_d \varphi[\beta]$, then $\beta(X) = \beta(Y)$
 - Thus, claim 1 follows by induction
 - Otherwise:
 - Let $X_i \in rr(\varphi)$
 - ⇒ $X_1 \in rr(\varphi_1)$ or $X_1 \in rr(\varphi_2)$
 - ⇒ $\beta(X_i) \in \text{adom}(\varphi, D)$ by induction
- $\varphi := \exists Y \varphi_1$:
 - By assumption $rr(\varphi) \neq \perp$, so $rr(\varphi_1) = rr(\varphi) \cup \{Y\}$
 - Therefore $rr(\varphi) \subseteq rr(\varphi_1)$ and claim 1 follows by induction

Proof of Part 2 (ctd)

Having shown claim 1, we now show that claim 2

$$D \models_d \varphi[\beta] \iff D \models_{\text{adom}(\varphi, D) \cup \beta(\text{free}(\varphi))} \varphi[\beta]$$

holds for composed formulas, by induction on φ

- The cases $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, and $\varphi_1 \wedge \varphi_2$ are trivial
- It remains to show claim 2 for $\exists Y\varphi_1$

“ \Leftarrow ” follows by induction and $\text{adom}(\varphi, D) \cup \beta(\text{free}(\varphi)) \subseteq d$

- “ \Rightarrow ”
- Assume that $D \models_d \varphi[\beta]$
 - \Rightarrow there is $a \in d$ with $D \models_d \varphi_1[\beta(Y/a)]$
 - It holds that $Y \in \text{rr}(\varphi_1)$, so claim 1 implies that $a \in \text{adom}(\varphi_1, D)$
 - Now let $\beta' := \beta(Y/a)$
 - $\Rightarrow \text{adom}(\varphi_1, D) \cup \beta'(\text{free}(\varphi_1)) = \text{adom}(\varphi, D) \cup \beta(\text{free}(\varphi))$
 - By induction we then have $D \models_{\text{adom}(\varphi, D) \cup \beta(\text{free}(\varphi))} \varphi_1[\beta']$
 - And also: $D \models_{\text{adom}(\varphi, D) \cup \beta(\text{free}(\varphi))} \varphi[\beta]$
 - This completes claim 2

Proof of Part (b) (ctd)

It remains to show that the two claims

- 1 if $X_i \in rr(\varphi)$ and $D \models_d \varphi[\beta]$, then $\beta(x_1) \in \text{adom}(\varphi, D)$, and
- 2 $D \models_d \varphi[\beta] \iff D \models_{\text{adom}(\varphi, D) \cup \beta(\text{free}(\varphi))} \varphi[\beta]$

imply part (b) of the proposition, i.e. imply that for every range-restricted formula φ the query $\{\langle \bar{X} \rangle \mid \varphi\}$ is range-restricted.


- Let φ be range-restricted and w.l.o.g. in *SRNF*, so $rr(\varphi) = \text{free}(\varphi)$
 - Using claim 1 we conclude that the query result contains only values from $\text{adom}(\varphi, D)$
 - Now consider d_1, d_2 with $\text{adom}(\varphi, D) \subseteq d_i \subseteq \text{dom}$ for $1 \leq i \leq 2$
 - Let $D \models_{d_1} \varphi[\beta]$ for some β
 - Using claim 1 we conclude that $\beta(\text{free}(\varphi)) \subseteq \text{adom}(\varphi, D)$
 - Using claim 2 we conclude that $D \models_{d_1} \varphi[\beta] \iff D \models_{\text{adom}(\varphi, D)} \varphi[\beta]$
 - Hence, $\llbracket \varphi \rrbracket_{d_1}(D) = \llbracket \varphi \rrbracket_{\text{adom}(D)}$
 - Analogously: $\llbracket \varphi \rrbracket_{d_2}(D) = \llbracket \varphi \rrbracket_{\text{adom}(D)}$ and thus $\llbracket \varphi \rrbracket_{d_1}(D) = \llbracket \varphi \rrbracket_{d_2}(D)$
- $\Rightarrow \varphi$ is range-restricted

Equivalence NR-Datalog[¬] and Relational Algebra

When combining Lemma (*₁) and Lemma (*₂) we immediately obtain the central result of this section:²

Theorem

Relational algebra and NR-Datalog[¬] have exactly the same expressive power.

²Recall: Lemmata (*₃) and (*₄) were used to prove Lemma (*₂). 

Datalog[⊃]: Well-founded Semantics

Limitations of the Stratified Semantics

- Drawback: some Datalog[⊃] don't have a stratified semantics
- We now present a well-founded semantics that is defined for every Datalog[⊃] program
- Whenever a Datalog[⊃] program is stratified, its stratified and its well-founded semantics coincide

Datalog[⊥]: Well-founded Semantics

Example: Board Game

(A) Consider the rule

$$\text{win}(X) \leftarrow \text{move}(X, Y), \neg \text{win}(Y)$$

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

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

- The models $\text{win} := \{a, c\}$ and $\text{win} := \{b, d\}$ are minimal.
- The model $\text{win} := \{a, c\}$ is minimal and *supports*; it corresponds to our intuition.

Datalog[⊥]: Well-founded Semantics

Example (ctd)

(B) 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.

Datalog[⊥]: Well-founded Semantics

Example (ctd)

(C) Consider the same rule

$$\text{win}(X) \leftarrow \text{move}(X, Y), \neg \text{win}(Y)$$

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

Are there “intuitive” models? If so, which ones?

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

<i>true:</i>	$\text{win}(d), \text{win}(f)$
<i>false:</i>	$\text{win}(e), \text{win}(g)$
<i>undefined:</i>	$\text{win}(a), \text{win}(b), \text{win}(c)$

Datalog[⊥]: Well-founded Semantics

Example (ctd)

(C) Consider the same rule

$$\text{win}(X) \leftarrow \text{move}(X, Y), \neg \text{win}(Y)$$

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

Are there “intuitive” models? If so, which ones?

The following 3-ary model is intuitively convincing:

<i>true:</i>	$\text{win}(d), \text{win}(f)$
<i>false:</i>	$\text{win}(e), \text{win}(g)$
<i>undefined:</i>	$\text{win}(a), \text{win}(b), \text{win}(c)$

Datalog⁻: Well-founded Semantics

Outlook

We start with programs that contain their input as part of their definition, subsequently discuss

- 3 – *ary* instances together with their notion of truth,
- then 3 – *ary* minimal models of Datalog (without negation),
- then 3 – *ary* minimal (3-stable) models of Datalog⁻,
- and finally present a fixpoint operator for the computation of the semantics.

Datalog[⊥]: Well-founded Semantics

Programs Extended by Their Input

- Let Π be a Datalog[⊥] program and \mathcal{E} be an instance over the EDB relations. Let $\Pi_{\mathcal{E}}$ be Π extended by \mathcal{E} as follows.

$$R(a_1, \dots, a_n) \in \mathcal{E} \implies R(a_1, \dots, a_n) \leftarrow \text{true is a rule in } \Pi_{\mathcal{E}}.$$

It holds that $\Pi(\mathcal{E}) = \Pi_{\mathcal{E}}(\emptyset)$.

- Let Π be a program with input \mathcal{E} . In the following, we only consider $\Pi_{\mathcal{E}}$. When talking about a program Π only, we tacitly assume that Π already contains the respective input.

Datalog[⊥]: Well-founded Semantics

Ground instances

- Let $\mathcal{B}(\Pi)$ be the set of all facts of the form $R(a_1, \dots, a_n)$, where R is a relation in Π and a_1, \dots, a_n are constants appearing in Π .

$ground(\Pi)$, the *ground instance* of Π , is obtained by instantiating the rules from Π with all possible combinations of constants from Π .

The *ground literals* in the body and head of rules in $ground(\Pi)$ therefore are facts in $\mathcal{B}(\Pi)$.

Datalog[⊥]: Well-founded Semantics

3-ary Instances

3-ary instances are represented by specifying the facts that are true and the facts that are false; all unspecified facts have an unknown truth value.

Shortcut notation for truth values:

$$\text{true} : 1 \qquad \text{false} : 0 \qquad \text{undefined} : 1/2$$

To determine the truth value of a formula α that is composed out of a boolean combination of two formulas β, γ w.r.t. to some instance \mathcal{I} , we extend \mathcal{I} to $\hat{\mathcal{I}}$:

$$\begin{aligned} \hat{\mathcal{I}}(\beta \wedge \gamma) &= \min\{\hat{\mathcal{I}}(\beta), \hat{\mathcal{I}}(\gamma)\} \\ \hat{\mathcal{I}}(\beta \vee \gamma) &= \max\{\hat{\mathcal{I}}(\beta), \hat{\mathcal{I}}(\gamma)\} \\ \hat{\mathcal{I}}(\neg\beta) &= 1 - \hat{\mathcal{I}}(\beta) \\ \hat{\mathcal{I}}(\beta \leftarrow \gamma) &= \begin{cases} 1 & \text{if } \hat{\mathcal{I}}(\gamma) \leq \hat{\mathcal{I}}(\beta), \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

$p \leftarrow q$ and $p \vee \neg q$ now have different truth values!

Datalog⁻: Well-founded Semantics

Definition: 3-ary Instances

A *3-ary instance* \mathcal{I} over (the schema of) Π is a total mapping of $\mathcal{B}(\Pi)$ into $\{0, 1, 1/2\}$, where $\mathcal{I}^1, \mathcal{I}^0, \mathcal{I}^{1/2}$ are the corresponding subsets of \mathcal{I} in dependence of their truth value.

Example: 3-ary Instance

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

$ground(\Pi)$:

$$\begin{aligned} & win(a) \leftarrow move(a, d), \neg win(d) \\ & win(a) \leftarrow move(a, b), \neg win(b) \\ & win(d) \leftarrow move(d, e), \neg win(e) \\ & \vdots \end{aligned}$$

- $move := \{(a, b), (b, c), (c, a), (a, d), (d, e), (d, f), (f, g)\}$
- 3-ary Instance: $\mathcal{I}^1(win) := \{d, f\}$, $\mathcal{I}^0(win) := \{e, g\}$, $\mathcal{I}^{1/2}(win) := \{a, b, c\}$

Datalog[⊥]: Well-founded Semantics

Definition: 3-ary Instances

A *3-ary instance* \mathcal{I} over (the schema of) Π is a total mapping of $\mathcal{B}(\Pi)$ into $\{0, 1, 1/2\}$, where $\mathcal{I}^1, \mathcal{I}^0, \mathcal{I}^{1/2}$ are the corresponding subsets of \mathcal{I} in dependence of their truth value.

Example: 3-ary Instance

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

$ground(\Pi)$:

$$\begin{array}{l} win(a) \leftarrow move(a, d), \neg win(d) \\ win(a) \leftarrow move(a, b), \neg win(b) \\ win(d) \leftarrow move(d, e), \neg win(e) \\ \vdots \end{array}$$

- $move := \{(a, b), (b, c), (c, a), (a, d), (d, e), (d, f), (f, g)\}$
- 3-ary Instance: $\mathcal{I}^1(win) := \{d, f\}$, $\mathcal{I}^0(win) := \{e, g\}$, $\mathcal{I}^{1/2}(win) := \{a, b, c\}$

Datalog[⊥]: Well-founded Semantics

Additional Definitions

- Consider a program Π and an 3-ary instance \mathcal{I} .
- \mathcal{I} is called *3-ary model* of Π , if \mathcal{I} satisfies all rules in $ground(\Pi)$.
- A 3-ary instance \mathcal{I} is called *total* (or *2-ary*), if $\mathcal{I}^{1/2} = \emptyset$.
- We define the order \prec , which sorts two 3-ary instances \mathcal{I}, \mathcal{J} as follows.

$\mathcal{I} \prec \mathcal{J}$ exactly if for every $A \in \mathcal{B}(\Pi) : \mathcal{I}(A) \leq \mathcal{J}(A)$.

- The minimal 3-ary instance w.r.t. \prec assigns to each atom the truth value 0. We denote this instance as \perp .

Datalog[⊥]: Well-founded Semantics

3-ary minimal Models of Datalog

A Datalog program Π is called *3-extended*, if in the body of the rules the truth values 0, 1/2, and 1 may appear as literals.

The *3-ary immediate consequence operator* $3T_{\Pi}$ of a Datalog program Π is a mapping over 3-ary instances \mathcal{I} defined as follows. Let $A \in \mathcal{B}(\Pi)$.

$$3T_{\Pi}(\mathcal{I})(A) := \begin{cases} 1 & \text{there is a rule } A \leftarrow \text{body in } \text{ground}(\Pi), \\ & \text{where } \hat{\mathcal{I}}(\text{body}) = 1, \\ 0 & \text{for every rule } A \leftarrow \text{body in } \text{ground}(\Pi) \text{ it} \\ & \text{holds that } \hat{\mathcal{I}}(\text{body}) = 0 \text{ or there is no rule} \\ & \text{in } \text{ground}(\Pi) \text{ having } A \text{ in its head,} \\ 1/2 & \text{otherwise.} \end{cases}$$

Datalog[⊥]: Well-founded Semantics

Theorem

Let Π be a 3-ary Datalog program.

- $3T_{\Pi}$ is monotonic and the sequence $\{3T_{\Pi}^i(\perp)\}_{i>0}$ is increasing and converges towards the minimal fixpoint of $3T_{\Pi}$.
- Π has a unique minimal 3-ary model. This model is identical to the minimal fixpoint.

Datalog[⊥]: Well-founded Semantics

Example

Consider the 3-ary Datalog program Π :

$$\begin{aligned} p &\leftarrow 1/2 \\ p &\leftarrow q, 1/2 \\ q &\leftarrow p, r \\ q &\leftarrow p, s \\ s &\leftarrow q \\ r &\leftarrow 1 \end{aligned}$$

Then

$$\begin{aligned} 3T_{\Pi}(\{\neg p, \neg q, \neg r, \neg s\}) &= \{\neg q, r, \neg s\} \\ 3T_{\Pi}(\{\neg q, r, \neg s\}) &= \{r, \neg s\} \\ 3T_{\Pi}(\{r, \neg s\}) &= \{r\} \\ 3T_{\Pi}(\{r\}) &= \{r\} \end{aligned}$$

Datalog[⊥]: Well-founded Semantics

3-stable Models of Datalog[⊥]

Let Π be a Datalog[⊥] program and \mathcal{I} a 3-ary instance.

- The *positivized ground version* of Π w.r.t. \mathcal{I} , $pg(\Pi, \mathcal{I})$, is the 3-ary Datalog program resulting from $ground(\Pi)$ when replacing every negative literal $\neg A$ by $\hat{\mathcal{I}}(\neg A)$, i.e. by 0, 1, 1/2.
- We denote the minimal fixpoint $pg(\Pi, \mathcal{I})(\perp)$ of $pg(\Pi, \mathcal{I})$ by $conseq_{\Pi}(\mathcal{I})$, where we assume truth values from \mathcal{I} for the negative literals.
- A 3-ary instance \mathcal{I} is called *3-stable model* of a Datalog[⊥] program Π , if

$$conseq_{\Pi}(\mathcal{I}) = \mathcal{I}.$$

Datalog[⊥]: Well-founded Semantics

Example

Consider the Datalog[⊥] program Π :

$$\begin{aligned}p &\leftarrow \neg r \\q &\leftarrow \neg r, p \\s &\leftarrow \neg t \\t &\leftarrow q, \neg s \\u &\leftarrow \neg t, p, s\end{aligned}$$

and the 3-ary instance

$$\mathcal{I} := \{p, q, \neg r\}$$

We obtain for $pg(\Pi, \mathcal{I})$:

$$\begin{aligned}p &\leftarrow 1 \\q &\leftarrow 1, p \\s &\leftarrow 1/2 \\t &\leftarrow q, 1/2 \\u &\leftarrow 1/2, p, s\end{aligned}$$

and finally:



Datalog[⊖]: Well-founded Semantics

Remark

Datalog[⊖] programs may have multiple 3-stable models.

Definition: Well-founded Semantics

Let Π be a Datalog[⊖] program. The *well-founded semantics* of Π is the 3-ary instance that contains exactly those positive and negative facts contained in all 3-stable models of Π .

We write Π^{wf} , or $\Pi^{wf}(\mathcal{E})$, if an input \mathcal{E} is explicitly given for some program Π .

Datalog[⊃]: Well-founded Semantics

Example

Consider the Datalog[⊃] program Π :

$$\begin{aligned}p &\leftarrow \neg r \\q &\leftarrow \neg r, p \\s &\leftarrow \neg t \\t &\leftarrow q, \neg s \\u &\leftarrow \neg t, p, s\end{aligned}$$

This program has three 3-stable models:

$$\begin{aligned}\mathcal{I}_1 &:= \{p, q, t, \neg r, \neg s, \neg u\} \\ \mathcal{I}_2 &:= \{p, q, s, \neg r, \neg t, u\} \\ \mathcal{I}_3 &:= \{p, q, \neg r\}\end{aligned}$$

The well-founded semantics of Π is given by \mathcal{I}_3 .

Datalog⁻: Fixpoint Computation

Alternating Fixpoint

In practice, it is unfeasible to compute all 3-stable models in order to compute the well-founded semantics. In the following we discuss a different approach called *alternating fixpoint computation*.

Consider the sequence $\{\mathcal{I}_i\}_{i \geq 0}$ of 3-ary instance for some program Π :

$$\begin{aligned}\mathcal{I}_0 &:= \perp, \\ \mathcal{I}_{i+1} &:= \text{conseq}_{\Pi}(\mathcal{I}_i).\end{aligned}$$

Example

Consider the program Π consisting of the rule

$$\text{win}(X) \leftarrow \text{move}(X, Y), \neg \text{win}(Y)$$

and the inputs

$$\mathcal{E}_1(\text{move}) := \{(a, b), (b, a), (a, c)\}$$

$$\mathcal{E}_2(\text{move}) := \{(a, b), (b, a), (a, c), (c, d)\}$$

The resulting ground versions are $\text{ground}(\Pi_{\mathcal{E}_1}, \perp)$ and $\text{ground}(\Pi_{\mathcal{E}_2}, \perp)$ are computed as follows:³

$\text{move}(a, b) \leftarrow$
 $\text{move}(b, a) \leftarrow$
 $\text{move}(a, c) \leftarrow$
 $\text{win}(a) \leftarrow \text{move}(a, b), \neg \text{win}(b)$
 $\text{win}(a) \leftarrow \text{move}(a, c), \neg \text{win}(c)$
 $\text{win}(b) \leftarrow \text{move}(b, a), \neg \text{win}(a)$

$\text{move}(a, b) \leftarrow$
 $\text{move}(b, a) \leftarrow$
 $\text{move}(a, c) \leftarrow$
 $\text{move}(c, d) \leftarrow$
 $\text{win}(a) \leftarrow \text{move}(a, b), \neg \text{win}(b)$
 $\text{win}(a) \leftarrow \text{move}(a, c), \neg \text{win}(c)$
 $\text{win}(b) \leftarrow \text{move}(b, a), \neg \text{win}(a)$
 $\text{win}(c) \leftarrow \text{move}(c, d), \neg \text{win}(d)$

³Rules containing unsatisfiable subgoals of *move* are omitted.

Remarks

- If \mathcal{I} is total, then $\text{conseq}_{\Pi}(\mathcal{I})$ is total. Since \perp is total, all \mathcal{I}_i are total.
- $\text{conseq}_{\Pi}(\mathcal{I})$ is antimonotonic. This means, $\mathcal{I} \prec \mathcal{J}$ implies $\text{conseq}_{\Pi}(\mathcal{I}) \succ \text{conseq}_{\Pi}(\mathcal{J})$.
- \mathcal{I}_0 under-estimates (over-estimates) the positive (negative) facts in Π 's answer set; accordingly, \mathcal{I}_1 over-estimates (under-estimates) the corresponding positive (negative) facts in Π 's answer set
- It holds that $\mathcal{I}_0 \prec \mathcal{I}_1$ and $\mathcal{I}_0 \prec \mathcal{I}_2$, so (due to the antimonotonic):

$$\mathcal{I}_0 \prec \mathcal{I}_2 \prec \mathcal{I}_4 \prec \dots$$

$$\mathcal{I}_1 \succ \mathcal{I}_3 \succ \mathcal{I}_5 \succ \dots$$

$$\mathcal{I}_0 \prec \mathcal{I}_1, \mathcal{I}_2 \prec \mathcal{I}_3, \mathcal{I}_4 \prec \mathcal{I}_5, \dots$$

- Let \mathcal{I}_* be the limit of the increasing sequence $\{\mathcal{I}_{2i}\}_{i \geq 0}$ and \mathcal{I}^* be the limit of the decreasing sequence $\{\mathcal{I}_{2i+1}\}_{i \geq 0}$. It holds that $\mathcal{I}_* \prec \mathcal{I}^*$.
- Let $\mathcal{I}_*^* = (\mathcal{I}_*)^1 \cup (\mathcal{I}^*)^0$.

Datalog[¬]: Fixpoint Computation

Example

Consider the Datalog[¬] program Π :

$$\begin{aligned}p &\leftarrow \neg r \\q &\leftarrow \neg r, p \\s &\leftarrow \neg t \\t &\leftarrow q, \neg s \\u &\leftarrow \neg t, p, s\end{aligned}$$

For a given \mathcal{I} we first have to specify the positivized ground version $pg(\Pi, \mathcal{I})$, where $\mathcal{I}_0 = \perp$.

Datalog[¬]: Fixpoint Computation

Example (ctd)

$$\begin{array}{l}
 (1) \text{ pg}(\Pi, \perp) : \\
 p \leftarrow 1 \\
 q \leftarrow 1, p \\
 s \leftarrow 1 \\
 t \leftarrow q, 1 \\
 u \leftarrow 1, p, s
 \end{array}
 \longrightarrow \mathcal{I}_1 = \{p, q, \neg r, s, t, u\}$$

$$\begin{array}{l}
 (2) \text{ pg}(\Pi, \mathcal{I}_1) : \\
 p \leftarrow 1 \\
 q \leftarrow 1, p \\
 s \leftarrow 0 \\
 t \leftarrow q, 0 \\
 u \leftarrow 0, p, s
 \end{array}
 \longrightarrow \mathcal{I}_2 = \{p, q, \neg r, \neg s, \neg t, \neg u\}$$

Datalog⁻: Fixpoint Computation

Example (ctd)

$$\begin{array}{ll}
 (3) \text{ pg}(\Pi, \mathcal{I}_2) : & p \leftarrow 1 \\
 & q \leftarrow 1, p \\
 & s \leftarrow 1 \quad \longrightarrow \mathcal{I}_3 = \{p, q, \neg r, s, t, u\} \\
 & t \leftarrow q, 1 \\
 & u \leftarrow 1, p, s
 \end{array}$$

$$\begin{array}{ll}
 (4) \text{ pg}(\Pi, \mathcal{I}_3) : & p \leftarrow 1 \\
 & q \leftarrow 1, p \\
 & s \leftarrow 0 \quad \longrightarrow \mathcal{I}_4 = \{p, q, \neg r, \neg s, \neg t, \neg u\} \\
 & t \leftarrow q, 0 \\
 & u \leftarrow 0, p, s
 \end{array}$$

$$\mathcal{I}_*^* = \{p, q, \neg r\}$$

Datalog[¬]: Fixpoint Computation

Lemma

Let Π be a Datalog[¬] program.

- \mathcal{I}_*^* is a 3-stable model of Π .
- It holds that $\mathcal{I}_*^* = \Pi^{wf}(\emptyset)$, i.e. \mathcal{I}_*^* implements the *well-founded semantics* of Π .

Theorem

Let Π be a stratified Datalog[¬] program. Then $\Pi^{wf}(\emptyset)$ is total and for every total input \mathcal{E} : $\Pi^{wf}(\mathcal{E}) = \Pi^{strat}(\mathcal{E})$.

Datalog⁻: Fixpoint Computation

Example

Consider the program Π :

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

and the input: $\text{green} := \{(1, 2)\}$, $\text{red} := \{(1, 2), (2, 3)\}$

$\text{pg}(\Pi, \perp)$ realizes the assumption that the facts from stratum 1 that are addressed in stratum 2 are all wrong, by considering all negative literals as true without assuming the corresponding positive literals as false.

Hence, $\mathcal{I}_1 := \text{conseq}_{\Pi}(\perp)$ over-estimates the facts that basically could be derived in stratum 2.

$\text{pg}(\Pi, \mathcal{I}_1)$ considers only those facts from stratum 1 as wrong that could not have been derived in stratum 1. $\mathcal{I}_2 := \text{conseq}_{\Pi}(\mathcal{I}_1)$ thus gives us the stratified semantics of Π .

Example

Consider the program Π :

$$\begin{aligned} p(1, 2) &\leftarrow \\ q(X) &\leftarrow p(X, Y) \\ r(X) &\leftarrow \neg q(X) \\ t(X) &\leftarrow \neg r(X) \end{aligned}$$

together with its ground instance and the positivized ground versions $pg(\Pi, \mathcal{I})$:⁴

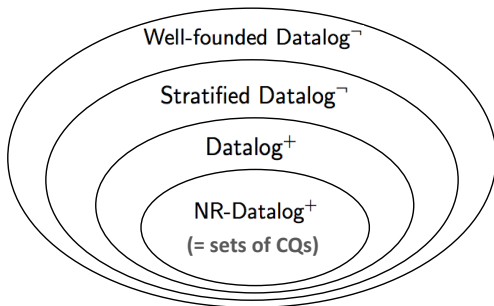
$ground(\Pi)$	$pg(\Pi, \perp)$	$pg(\Pi, \mathcal{I}_1)$	$pg(\Pi, \mathcal{I}_2)$	$pg(\Pi, \mathcal{I}_3)$
$p(1, 2) \leftarrow$	$p(1, 2) \leftarrow$	$p(1, 2) \leftarrow$	$p(1, 2) \leftarrow$	$p(1, 2) \leftarrow$
$q(1) \leftarrow p(1, 2)$	$q(1) \leftarrow p(1, 2)$	$q(1) \leftarrow p(1, 2)$	$q(1) \leftarrow p(1, 2)$	$q(1) \leftarrow p(1, 2)$
$r(1) \leftarrow \neg q(1)$	$r(1) \leftarrow 1$	$r(1) \leftarrow 0$	$r(1) \leftarrow 0$	$r(1) \leftarrow 0$
$r(2) \leftarrow \neg q(2)$	$r(2) \leftarrow 1$	$r(2) \leftarrow 1$	$r(2) \leftarrow 1$	$r(2) \leftarrow 1$
$t(1) \leftarrow \neg r(1)$	$t(1) \leftarrow 1$	$t(1) \leftarrow 0$	$t(1) \leftarrow 1$	$t(1) \leftarrow 1$
$t(2) \leftarrow \neg r(2)$	$t(2) \leftarrow 1$	$t(2) \leftarrow 0$	$t(2) \leftarrow 0$	$t(2) \leftarrow 0$

The well-founded model is total and given by $\{p(1, 2), q(1), r(2), t(1)\}$ ⁵; this corresponds to the stratified semantics.

⁴Rules containing non-satisfiable subgoals are omitted.

⁵All facts that are not listed are wrong.

Overview: Expressiveness of Datalog Fragments



- Note: all inclusions in the diagram are strict
- Not shown in the diagram: NR-Datalog⁻
 - Equivalent to relational algebra
 - Contained in Stratified Datalog⁻
 - Uncomparable to Datalog⁺