Ontology and Database Systems: Foundations of Database Systems

Part 5: Datalog

Werner Nutt

Faculty of Computer Science
Master of Science in Computer Science

A.Y. 2014/2015

Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

Motivation

- Relational Calculus and Relational Algebra were considered to be "*the*" database languages for a long time
- Codd: A query language is "complete," if it yields Relational Calculus
- However, Relational Calculus misses an important feature: *recursion*
- Example: A metro database with relation `links:line, station, nextstation`
  - What stations are reachable from station "Odeon"?
  - Can we go from Odeon to Tuileries?
  - etc.
- It can be proved: such queries cannot be expressed in Relational Calculus
- This motivated a logic-programming extension to conjunctive queries: *datalog*

Example: Metro Database Instance

link	line	station	nextstation
	4	St. Germain	Odeon
	4	Odeon	St. Michel
	4	St. Michel	Chatelet
	1	Chatelet	Louvres
	1	Louvres	Palais Royal
	1	Palais-Royal	Tuileries
	1	Tuileries	Concorde

Datalog program for the first query:

$$\begin{array}{rcl} \mathtt{reach}(\mathtt{X},\mathtt{X}) & \leftarrow & \mathtt{link}(\mathtt{L},\mathtt{X},\mathtt{Y}) \\ \mathtt{reach}(\mathtt{X},\mathtt{X}) & \leftarrow & \mathtt{link}(\mathtt{L},\mathtt{Y},\mathtt{X}) \\ \mathtt{reach}(\mathtt{X},\mathtt{Y}) & \leftarrow & \mathtt{link}(\mathtt{L},\mathtt{X},\mathtt{Z}), \mathtt{reach}(\mathtt{Z},\mathtt{Y}) \\ \mathtt{answer}(\mathtt{X}) & \leftarrow & \mathtt{reach}(\text{`}\mathtt{Odeon}\text{`},\mathtt{X}) \end{array}$$

- Note: this is a recursive definition
- Intuitively, if the part right of "←" is true,
  the rule "fires" and the atom left of "←" is concluded.

The Datalog Language

- Datalog is akin to Logic Programming
- The basic language (considered next) has many extensions
- There exist several approaches to defining the semantics:

  Model-theoretic approach: View rules as logical sentences, which state the query result

  Operational (fixpoint) approach: Obtain query result by applying an inference procedure, until a fixpoint is reached

  Proof-theoretic approach: Obtain proofs of facts in the query result, following a proof calculus (based on resolution)

Datalog vs. Logic Programming

Although datalog is akin to Logic Programming,
there are important differences:

- There are **no functions symbols** in datalog
  $\rightsquigarrow$ no unbounded data structures, such as lists, are supported
- Datalog has a **purely declarative semantics**
  $\rightsquigarrow$ In a datalog program,
  - the *order of clauses* is irrelevant
  - the *order of atoms* in a rule body is irrelevant
- Datalog distinguishes between
  - database relations ("*extensional database*", *edb*) and
  - derived relations ("*intensional database*", *idb*)

Syntax of "plain datalog", or "datalog"

Definition

A *datalog rule* r is an expression of the form

$$R_0(\bar{x}_0) \leftarrow R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n) \tag{1}$$

where

- $n \geq 0$,
- $R_0, \ldots, R_n$ are relations names,
- $\bar{x}_0, \ldots, \bar{x}_n$ are tuples of variables and constants (from **dom**), and
- every variable in $\bar{x}_0$ occurs in $\bar{x}_1, \ldots, \bar{x}_n$ ("safety")

Remark

- The *head* of r, denoted $H(r)$, is $R_0(\bar{x}_0)$
- The *body* of r, denoted $B(r)$, is $\{\ R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)\ \}$
- The rule symbol "$\leftarrow$" is often also written as ":-"

Datalog Programs

Definition

A *datalog program* is a finite set of datalog rules.

Let P be a datalog program.

- An *extensional relation* of P is a relation occurring only in rule bodies of P
- An *intensional relation* of P is a relation occurring in the head of some rule in P
- The *extensional schema* of P, *edb*(P), consists of all extensional relations of P
- The *intensional schema* of P, *idb*(P), consists of all intensional relations of P
- The *schema* of P, *sch*(P), is the union of *edb*(P) and *idb*(P).

The Metro Example /1

Datalog program P on the metro database schema (w/o integrity constraints)

$$\mathcal{M} = \{\texttt{link}(\texttt{line},\ \texttt{station},\ \texttt{nextstation})\}:$$

$$\begin{array}{rcl}
\texttt{reach}(\texttt{X},\texttt{X}) & \leftarrow & \texttt{link}(\texttt{L},\texttt{X},\texttt{Y}) \\
\texttt{reach}(\texttt{X},\texttt{X}) & \leftarrow & \texttt{link}(\texttt{L},\texttt{Y},\texttt{X}) \\
\texttt{reach}(\texttt{X},\texttt{Y}) & \leftarrow & \texttt{link}(\texttt{L},\texttt{X},\texttt{Z}),\texttt{reach}(\texttt{Z},\texttt{Y}) \\
\texttt{answer}(\texttt{X}) & \leftarrow & \texttt{reach('Odeon',X)}
\end{array}$$

Here,

$$\begin{array}{rcl}
\mathit{edb}(P) & = & \{\texttt{link}\} \quad (= \mathcal{M}), \\
\mathit{idb}(P) & = & \{\texttt{reach},\texttt{answer}\}, \\
\mathit{sch}(P) & = & \{\texttt{link},\texttt{reach},\texttt{answer}\}
\end{array}$$

Datalog Syntax (cntd)

- The set of constants occurring in program P is denoted as $\mathit{adom}(P)$
- The *active domain* of P with respect to an instance $\mathbf{I}$ is defined as

$$\mathit{adom}(P, \mathbf{I}) := \mathit{adom}(P) \cup \mathit{adom}(\mathbf{I}),$$

that is, as the set of constants occurring in P and $\mathbf{I}$

Definition (Rule Instantiation)

Let $\alpha\colon \mathit{var}(r) \cup \textbf{dom} \to \textbf{dom}$ be an assignment for the variables in a rule r of form (1). Then the *instantiation* of r with α, denoted $\alpha(r)$, is the rule

$$R_0(\alpha(\bar{x}_0)) \leftarrow R_1(\alpha(\bar{x}_1)), \ldots, R_n(\alpha(\bar{x}_n)),$$

which results from replacing each variable x with $\alpha(x)$.

The Metro Example/2

- For the datalog program P above, we have that $\mathit{adom}(P) = \{\text{ Odeon }\}$
- We consider the database instance $\mathbf{I}$:

link	line	station	nextstation
	4	St. Germain	Odeon
	4	Odeon	St. Michel
	4	St. Michel	Chatelet
	1	Chatelet	Louvre
	1	Louvre	Palais-Royal
	1	Palais-Royal	Tuileries
	1	Tuileries	Concorde

Then $\mathit{adom}(\mathbf{I}) = \{$4, 1, St.Germain, Odeon, St.Michel, Chatelet, Louvres, Palais-Royal, Tuileries, Concorde$\}$

- Also $\mathit{adom}(P, \mathbf{I}) = \mathit{adom}(\mathbf{I})$

The Metro Example/3

- The rule

$$\begin{aligned}\texttt{reach}(\texttt{St.Germain}, \texttt{Odeon}) \quad \leftarrow \quad & \texttt{link}(\texttt{Louvre}, \texttt{St.Germain}, \texttt{Concorde}),\\ & \texttt{reach}(\texttt{Concorde}, \texttt{Odeon})\end{aligned}$$

is an instantiation of the rule

$$\texttt{reach}(\texttt{X}, \texttt{Y}) \quad \leftarrow \quad \texttt{link}(\texttt{L}, \texttt{X}, \texttt{Z}), \texttt{reach}(\texttt{Z}, \texttt{Y})$$

(take $\alpha(\texttt{X}) = \texttt{St.Germain}$, $\alpha(\texttt{L}) = \texttt{Louvre}$, $\alpha(\texttt{Y}) = \texttt{Odeon}$, $\alpha(\texttt{Z}) = \texttt{Concorde}$)

Datalog: Model-Theoretic Semantics

General Idea:

- We view a program as a set of first-order sentences
- Given an instance $\mathbf{I}$ of *edb*(P),
  the result of P is a database instance of *sch*(P)
  that extends $\mathbf{I}$ and satisfies the sentences
  (or, is a *model* of the sentences)
- There can be many models
- The *intended answer* is specified by particular models
- These particular models are selected by "external" conditions

Logical Theory Σ_P

- To every datalog rule r of the form $R_0(\bar{x}_0) \leftarrow R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$, with variables $x_1, \ldots, x_m$, we associate the logical sentence $\sigma(r)$:

$$\forall x_1, \cdots \forall x_m \, (R_1(\bar{x}_1) \wedge \cdots \wedge R_n(\bar{x}_n) \rightarrow R_0(\bar{x}_0))$$

- To a program P, we associate the set of sentences $\Sigma_P = \{\sigma(r) \mid r \in P\}$

Definition

Let P be a datalog program and $\mathbf{I}$ an instance of *edb*(P). Then,

- A *model* of P is an instance of *sch*(P) that satisfies Σ_P
- We compare models wrt set inclusion "$\subseteq$"
  (in the Logic Programming perspective)
- The *semantics* of P on input $\mathbf{I}$, denoted $P(\mathbf{I})$,
  is the *least model* of P containing $\mathbf{I}$, if it exists

Example

For program P and instance $\mathbf{I}$ of the Metro Example, the least model is:

link	line	station	nextstation
	4	St. Germain	Odeon
	4	Odeon	St. Michel
	4	St. Michel	Chatelet
	1	Chatelet	Louvres
	1	Louvres	Palais-Royal
	1	Palais-Royal	Tuileries
	1	Tuileries	Concorde

reach		
	St. Germain	St. Germain
	Odeon	Odeon
	. . .	
	Concorde	Concorde
	St. Germain	Odeon
	St. Germain	St.Michel
	St. Germain	Chatelet
	St. Germain	Louvre
	. . .	

answer	
	Odeon
	St. Michel
	Chatelet
	Louvre
	Palais-Royal
	Tuileries
	Concorde

Questions

1. Is the semantics $P(\mathbf{I})$ well-defined for every input instance $\mathbf{I}$?
2. How can one compute $P(\mathbf{I})$?

Observation: For any $\mathbf{I}$, there is a model of P containing $\mathbf{I}$

- Let $\mathbf{B}(P, \mathbf{I})$ be the instance of *sch*(P) such that

$$\mathbf{B}(P, \mathbf{I})(R) = \begin{cases} \mathbf{I}(R) & \text{for each } R \in \mathit{edb}(P) \\ \mathit{adom}(P, \mathbf{I})^{\mathit{ary}(R)} & \text{for each } R \in \mathit{idb}(P) \end{cases}$$

- Then: $\mathbf{B}(P, \mathbf{I})$ is a model of P containing $\mathbf{I}$
  $\Rightarrow \quad P(\mathbf{I})$ is a subset of $\mathbf{B}(P, \mathbf{I})$ *(if it exists)*
- Naive algorithm: explore all subsets of $\mathbf{B}(P, \mathbf{I})$

Elementary Properties of $P(\mathbf{I})$

Let P be a datalog program, $\mathbf{I}$ an instance of *edb*(P), and $\mathcal{M}(\mathbf{I})$ the set of all models of P containing $\mathbf{I}$.

Theorem

The intersection $\bigcap_{M \in \mathcal{M}(\mathbf{I})} M$ is a model of P.

Corollary

1. $P(\mathbf{I}) = \bigcap_{M \in \mathcal{M}(\mathbf{I})} M$
2. *adom*$(P(\mathbf{I})) \subseteq$ *adom*$(P, \mathbf{I})$, that is, no new values appear
3. $P(\mathbf{I})(R) = \mathbf{I}(R)$, for each $R \in$ *edb*(P)

Consequences:

- $P(\mathbf{I})$ is well-defined for every $\mathbf{I}$
- If P and $\mathbf{I}$ are finite, the $P(\mathbf{I})$ is finite

Why Choose the Least Model?

There are two reasons to choose the least model containing $\mathbf{I}$:

1. The *Closed World Assumption*:
   - If a fact $R(\bar{c})$ is not true in all models of a database $\mathbf{I}$, then infer that $R(\bar{c})$ is false
   - This amounts to considering $\mathbf{I}$ as complete
   - . . . which is customary in database practice
2. The relationship to Logic Programming:
   - Datalog should desirably match Logic Programming (seamless integration)
   - Logic Programming builds on the minimal model semantics

Relating Datalog to Logic Programming

- A logic program makes no distinction between *edb* and *idb*
- A datalog program P and an instance $\mathbf{I}$ of *edb*(P) can be mapped to the logic program

$$\mathcal{P}(P, \mathbf{I}) = P \cup \mathbf{I}$$

  (where $\mathbf{I}$ is viewed as a set of atoms in the Logic Programming perspective)
- Correspondingly, we define the logical theory

$$\Sigma_{P,\mathbf{I}} = \Sigma_P \cup \mathbf{I}$$

- The semantics of the logic program $\mathcal{P} = \mathcal{P}(P, \mathbf{I})$ is defined in terms of *Herbrand interpretations* of the language induced by $\mathcal{P}$:
  - The domain of discourse is formed by the constants occurring in $\mathcal{P}$
  - Each constant occurring in $\mathcal{P}$ is interpreted by itself

Herbrand Interpretations of Logic Programs

Given a rule r, we denote by *Const*(r) the set of all constants in r

Definition

For a (function-free) logic program $\mathcal{P}$, we define

- the *Herbrand universe* of $\mathcal{P}$, by

$$\mathbf{HU}(\mathcal{P}) = \bigcup_{r \in \mathcal{P}} \mathit{Const}(r)$$

- the *Herbrand base* of $\mathcal{P}$, by

$$\mathbf{HB}(\mathcal{P}) = \{R(c_1, \ldots, c_n) \mid R \text{ is a relation in } \mathcal{P}, \\ c_1, \ldots, c_n \in \mathbf{HU}(\mathcal{P}), \text{ and } \mathit{ary}(R) = n\}$$

Example

$$\mathcal{P} = \{ \quad \begin{array}{l} \mathtt{arc}(\mathtt{a}, \mathtt{b}). \\ \mathtt{arc}(\mathtt{b}, \mathtt{c}). \\ \mathtt{reachable}(\mathtt{a}). \\ \mathtt{reachable}(\mathtt{Y}) \leftarrow \mathtt{arc}(\mathtt{X}, \mathtt{Y}), \mathtt{reachable}(\mathtt{X}). \ \} \end{array}$$

$$\begin{array}{rcl} \mathbf{HU}(\mathcal{P}) & = & \{\mathtt{a}, \mathtt{b}, \mathtt{c}\} \\ \mathbf{HB}(\mathcal{P}) & = & \{\mathtt{arc}(\mathtt{a}, \mathtt{a}),\ \mathtt{arc}(\mathtt{a}, \mathtt{b}),\ \mathtt{arc}(\mathtt{a}, \mathtt{c}), \\ & & \ \mathtt{arc}(\mathtt{b}, \mathtt{a}),\ \mathtt{arc}(\mathtt{b}, \mathtt{b}),\ \mathtt{arc}(\mathtt{b}, \mathtt{c}), \\ & & \ \mathtt{arc}(\mathtt{c}, \mathtt{a}),\ \mathtt{arc}(\mathtt{c}, \mathtt{b}),\ \mathtt{arc}(\mathtt{c}, \mathtt{c}), \\ & & \ \mathtt{reachable}(\mathtt{a}),\ \mathtt{reachable}(\mathtt{b}),\ \mathtt{reachable}(\mathtt{c})\} \end{array}$$

Grounding

- A rule r' is a *ground instance* of a rule r with respect to $\mathbf{HU}(\mathcal{P})$,
  if $r' = \alpha(r)$ for an assignment α
  such that $\alpha(x) \in \mathbf{HU}(\mathcal{P})$ for each $x \in \mathit{var}(r)$
- The *grounding* of a rule r with respect to $\mathbf{HU}(\mathcal{P})$,
  denoted $\mathit{Ground}_{\mathcal{P}}(r)$,
  is the set of all ground instances of r wrt $\mathbf{HU}(\mathcal{P})$
- The *grounding* of a logic program $\mathcal{P}$ is

$$\mathit{Ground}(\mathcal{P}) = \bigcup_{r \in \mathcal{P}} \mathit{Ground}_{\mathcal{P}}(r)$$

Example

$$
\begin{aligned}
\mathit{Ground}(\mathcal{P}) = \{ & \mathtt{arc}(\mathrm{a},\mathrm{b}).\ \ \mathtt{arc}(\mathrm{b},\mathrm{c}).\ \ \mathtt{reachable}(\mathrm{a}). \\
& \mathtt{reachable}(\mathrm{a}) \leftarrow \mathtt{arc}(\mathrm{a},\mathrm{a}), \mathtt{reachable}(\mathrm{a}). \\
& \mathtt{reachable}(\mathrm{b}) \leftarrow \mathtt{arc}(\mathrm{a},\mathrm{b}), \mathtt{reachable}(\mathrm{a}). \\
& \mathtt{reachable}(\mathrm{c}) \leftarrow \mathtt{arc}(\mathrm{a},\mathrm{c}), \mathtt{reachable}(\mathrm{a}). \\
& \mathtt{reachable}(\mathrm{a}) \leftarrow \mathtt{arc}(\mathrm{b},\mathrm{a}), \mathtt{reachable}(\mathrm{b}). \\
& \mathtt{reachable}(\mathrm{b}) \leftarrow \mathtt{arc}(\mathrm{b},\mathrm{b}), \mathtt{reachable}(\mathrm{b}). \\
& \mathtt{reachable}(\mathrm{c}) \leftarrow \mathtt{arc}(\mathrm{b},\mathrm{c}), \mathtt{reachable}(\mathrm{b}). \\
& \mathtt{reachable}(\mathrm{a}) \leftarrow \mathtt{arc}(\mathrm{c},\mathrm{a}), \mathtt{reachable}(\mathrm{c}). \\
& \mathtt{reachable}(\mathrm{b}) \leftarrow \mathtt{arc}(\mathrm{c},\mathrm{b}), \mathtt{reachable}(\mathrm{c}). \\
& \mathtt{reachable}(\mathrm{c}) \leftarrow \mathtt{arc}(\mathrm{c},\mathrm{c}), \mathtt{reachable}(\mathrm{c}).\ \}
\end{aligned}
$$

Herbrand Models

- A *Herbrand-interpretation* I of $\mathcal{P}$ is any subset $I \subseteq \mathbf{HB}(\mathcal{P})$
- A *Herbrand-model* of $\mathcal{P}$ is a Herbrand-interpretation that satisfies all sentences in $\Sigma_{P,\mathbf{I}}$
- Equivalently, $M \subseteq \mathbf{HB}(\mathcal{P})$ is a Herbrand model if
  for all $r \in$ *Ground*$(\mathcal{P})$ such that $B(r) \subseteq M$
  we have that $H(r) \subseteq M$

Example

The Herbrand models of program $\mathcal{P}$ above are exactly the following:

- $M_1 = \{\ \mathtt{arc}(\mathtt{a}, \mathtt{b}), \mathtt{arc}(\mathtt{b}, \mathtt{c}),$
  $\mathtt{reachable}(\mathtt{a}), \mathtt{reachable}(\mathtt{b}), \mathtt{reachable}(\mathtt{c})\ \}$
- $M_2 = \mathbf{HB}(\mathcal{P})$
- every interpretation M such that $M_1 \subseteq M \subseteq M_2$

and no others.

Logic Programming Semantics

Proposition

$\mathbf{HB}(\mathcal{P})$ is always a model of $\mathcal{P}$

Theorem

For every logic program there exists a least Herbrand model (wrt "$\subseteq$").

For a program $\mathcal{P}$, this model is denoted $\mathit{MM}(\mathcal{P})$ (for "minimal model").
The model $\mathit{MM}(\mathcal{P})$ is the semantics of $\mathcal{P}$.

Theorem (Datalog $\leftrightarrow$ Logic Programming))

Let P be a datalog program and $\mathbf{I}$ be an instance of $\mathit{edb}(P)$. Then,

$$P(\mathbf{I}) = \mathit{MM}(\mathcal{P}(P, \mathbf{I}))$$

Consequences

Results and techniques for Logic Programming can be exploited for datalog.

For example,

- proof procedures for Logic Programming (e.g., SLD resolution) can be applied to datalog (with some caveats, regarding for instance termination)
- datalog can be reduced by "grounding" to propositional logic programs

Fixpoint Semantics

Another view:

"If all facts in $\mathbf{I}$ hold, which other facts must hold after firing the rules in P?"

Approach:

- Define an *immediate consequence operator* $\mathbf{T}_P(\mathbf{K})$ on db instances $\mathbf{K}$
- Start with $\mathbf{K} = \mathbf{I}$
- Apply $\mathbf{T}_P$ to obtain a new instance: $\mathbf{K}_{\text{new}} := \mathbf{T}_P(\mathbf{K}) = \mathbf{I} \cup \text{new facts}$
- Iterate until nothing new can be produced
- The result yields the semantics

Immediate Consequence Operator

Let P be a datalog program and $\mathbf{K}$ be a database instance of *sch*(P).
A fact $R(\bar{t})$ is an *immediate* consequence for $\mathbf{K}$ and P, if either

- $R \in$ *edb*(P) and $R(\bar{t}) \in \mathbf{K}$, or
- there exists a ground instance r of a rule in P such that $H(r) = R(\bar{t})$ and $B(r) \subseteq \mathbf{K}$.

Definition (Immediate Consequence Operator)

The *immediate consequence operator* of a datalog program P is the mapping

$$\mathbf{T}_P\colon \mathit{inst}(\mathit{sch}(P)) \to \mathit{inst}(\mathit{sch}(P))$$

where

$$\mathbf{T}_P(\mathbf{K}) = \{A \mid A \text{ is an immediate consequence for } \mathbf{K} \text{ and } P\}.$$

Example

Consider

$$P = \{ \quad \texttt{reachable}(\mathtt{a}), \\ \texttt{reachable}(\mathtt{Y}) \leftarrow \texttt{arc}(\mathtt{X},\mathtt{Y}), \texttt{reachable}(\mathtt{X}) \ \}$$

where *edb*$(P) = \{\texttt{arc}\}$ and *idb*$(P) = \{\texttt{reachable}\}$.

Let

$$\begin{aligned}
\mathbf{I} = \mathbf{K}_1 &= \{\texttt{arc}(\mathtt{a},\mathtt{b}),\ \texttt{arc}(\mathtt{b},\mathtt{c})\} \\
\mathbf{K}_2 &= \{\texttt{arc}(\mathtt{a},\mathtt{b}),\ \texttt{arc}(\mathtt{b},\mathtt{c}),\ \texttt{reachable}(\mathtt{a})\} \\
\mathbf{K}_3 &= \{\texttt{arc}(\mathtt{a},\mathtt{b}),\ \texttt{arc}(\mathtt{b},\mathtt{c}),\ \texttt{reachable}(\mathtt{a}),\ \texttt{reachable}(\mathtt{b})\ \} \\
\mathbf{K}_4 &= \{\texttt{arc}(\mathtt{a},\mathtt{b}),\ \texttt{arc}(\mathtt{b},\mathtt{c}),\ \texttt{reachable}(\mathtt{a}),\ \texttt{reachable}(\mathtt{b}),\ \texttt{reachable}(\mathtt{c})\}
\end{aligned}$$

Example (cntd)

Then,

$$\mathbf{T}_P(\mathbf{K}_1) = \{\texttt{arc}(\mathsf{a},\mathsf{b}),\ \texttt{arc}(\mathsf{b},\mathsf{c}),\ \texttt{reachable}(\mathsf{a})\} \ = \ \mathbf{K}_2$$
$$\mathbf{T}_P(\mathbf{K}_2) = \{\texttt{arc}(\mathsf{a},\mathsf{b}),\ \texttt{arc}(\mathsf{b},\mathsf{c}),\ \texttt{reachable}(\mathsf{a}),\ \texttt{reachable}(\mathsf{b})\} \ = \ \mathbf{K}_3$$
$$\mathbf{T}_P(\mathbf{K}_3) = \{\texttt{arc}(\mathsf{a},\mathsf{b}),\ \texttt{arc}(\mathsf{b},\mathsf{c}),\ \texttt{reachable}(\mathsf{a}),\ \texttt{reachable}(\mathsf{b}),\ \texttt{reachable}(\mathsf{c})\} = \mathbf{K}_4$$
$$\mathbf{T}_P(\mathbf{K}_4) = \{\texttt{arc}(\mathsf{a},\mathsf{b}),\ \texttt{arc}(\mathsf{b},\mathsf{c}),\ \texttt{reachable}(\mathsf{a}),\ \texttt{reachable}(\mathsf{b}),\ \texttt{reachable}(\mathsf{c})\} = \mathbf{K}_4$$

Thus, $\mathbf{K}_4$ is a *fixpoint* of $\mathbf{T}_P$.

Definition

$\mathbf{K}$ is a *fixpoint* of operator $\mathbf{T}_P$ if $\mathbf{T}_P(\mathbf{K}) = \mathbf{K}$

Properties

Proposition

Let P be a datalog program.

1. The operator $\mathbf{T}_P$ is monotonic, that is,
$$\mathbf{K} \subseteq \mathbf{K}' \text{ implies } \mathbf{T}_P(\mathbf{K}) \subseteq \mathbf{T}_P(\mathbf{K}');$$
2. For all $\mathbf{K} \in \mathit{inst}(\mathit{sch}(P))$, we have:
$$\mathbf{K} \text{ is a model of } \Sigma_P \text{ if and only if } \mathbf{T}_P(\mathbf{K}) \subseteq \mathbf{K};$$
3. If $\mathbf{T}_P(\mathbf{K}) = \mathbf{K}$ (i.e., $\mathbf{K}$ is a fixpoint), then $\mathbf{K}$ is a model of Σ_P.

Note: The converse of 3. does not hold in general.

Datalog Semantics via Least Fixpoint

The semantics of P on a database instance $\mathbf{I}$ of *edb*(P) is a special fixpoint:

Theorem

Let P be a datalog program and $\mathbf{I}$ be a database instance. Then

1. $\mathbf{T}_P$ has a least (wrt "$\subseteq$") fixpoint containing $\mathbf{I}$, denoted *lfp*$(P, \mathbf{I})$.
2. Moreover, *lfp*$(P, \mathbf{I})$ = *MM*$(\mathcal{P}(P, \mathbf{I})) = P(\mathbf{I})$.

Constructive definition of $P(\mathbf{I})$ by *fixpoint iteration*

Proof (of Claim 2, first equality, sketch).

Let M_1 = *lfp*$(P, \mathbf{I})$ and M_2 = *MM*$(\mathcal{P}(P, \mathbf{I}))$.
Since M_1 is a fixpoint of $\mathbf{T}_P$, it is a model of Σ_P, and since it contains $\mathbf{I}$ it is a model of $\mathcal{P}(P, \mathbf{I})$. Hence, $M_2 \subseteq M_1$. Since M_2 is a model of $\mathcal{P}(P, \mathbf{I})$, it holds that $\mathbf{T}_P(M_2) \subseteq M_2$. Note that for every model M of $\mathcal{P}(P, \mathbf{I})$ we have, due to the monotonicity of $\mathbf{T}_P$, that $\mathbf{T}_P(M)$ is model. Hence, $\mathbf{T}_P(M_2) = M_2$, since M_2 is a minimal model. This implies that M_2 is a fixpoint, hence $M_1 \subseteq M_2$. $\square$

Fixpoint Iteration

For a datalog program P and an instance $\mathbf{I}$, we define the sequence $(\mathbf{I}_i)_{i \geq 0}$ by

$$\begin{aligned} \mathbf{I}_0 &= \mathbf{I} \\ \mathbf{I}_i &= \mathbf{T}_P(\mathbf{I}_{i-1}) \qquad \text{for } i > 0. \end{aligned}$$

We observe:

- By monotoncity of $\mathbf{T}_P$, we have $\mathbf{I}_0 \subseteq \mathbf{I}_1 \subseteq \mathbf{I}_2 \subseteq \cdots \subseteq \mathbf{I}_i \subseteq \mathbf{I}_{i+1} \subseteq \cdots$
- For every $i \geq 0$, we have $\mathbf{I}_i \subseteq \mathbf{B}(P, \mathbf{I})$
- Hence, for some integer $n \leq |\mathbf{B}(P, \mathbf{I})|$, we have $\mathbf{I}_{n+1} = \mathbf{I}_n$ $(=: \mathbf{T}_P^\omega(\mathbf{I}))$
- It holds that $\mathbf{T}_P^\omega(\mathbf{I}) = \mathit{lfp}(P, \mathbf{I}) = P(\mathbf{I})$.

This can be readily implemented by an algorithm.

Example

$$P = \{\, \texttt{reachable(a)}, \\ \texttt{reachable(Y)} \leftarrow \texttt{arc(X,Y)}, \texttt{reachable(X)} \,\}$$

$$\mathbf{I} = \{\texttt{arc}(\texttt{a},\texttt{b}),\ \texttt{arc}(\texttt{b},\texttt{c})\}$$

Then,

$$\begin{aligned}
\mathbf{I}_0 &= \{\texttt{arc}(\texttt{a},\texttt{b}),\ \texttt{arc}(\texttt{b},\texttt{c})\} \\
\mathbf{I}_1 = \mathbf{T}_P^1(\mathbf{I}) &= \{\texttt{arc}(\texttt{a},\texttt{b}),\ \texttt{arc}(\texttt{b},\texttt{c}), \texttt{reachable}(\texttt{a})\} \\
\mathbf{I}_2 = \mathbf{T}_P^2(\mathbf{I}) &= \{\texttt{arc}(\texttt{a},\texttt{b}),\ \texttt{arc}(\texttt{b},\texttt{c}), \texttt{reachable}(\texttt{a}),\ \texttt{reachable}(\texttt{b})\} \\
\mathbf{I}_3 = \mathbf{T}_P^3(\mathbf{I}) &= \{\texttt{arc}(\texttt{a},\texttt{b}),\ \texttt{arc}(\texttt{b},\texttt{c}), \texttt{reachable}(\texttt{a}),\ \texttt{reachable}(\texttt{b}),\ \texttt{reachable}(\texttt{c})\} \\
\mathbf{I}_4 = \mathbf{T}_P^4(\mathbf{I}) &= \{\texttt{arc}(\texttt{a},\texttt{b}),\ \texttt{arc}(\texttt{b},\texttt{c}), \texttt{reachable}(\texttt{a}),\ \texttt{reachable}(\texttt{b}),\ \texttt{reachable}(\texttt{c})\} \\
&= \mathbf{T}_P^3(\mathbf{I})
\end{aligned}$$

Thus, $\mathbf{T}_P^\omega(\mathbf{I}) = \mathit{lfp}(P, \mathbf{I}) = \mathbf{I}_4$.

Excursion: Fixpoint Theory

- Evaluating a datalog program P on $\mathbf{I}$ amounts to evaluating the logic program $\mathcal{P}(P, \mathbf{I})$
- For logic programs, fixpoint semantics is defined by appeal to fixpoint theory
- This provides another possibility to define semantics of datalog programs

Excursion: Fixpoint Theory/2

- A *complete lattice* is a partially ordered set $(U, \leq)$ such that each subset $V \subseteq U$ has a least upper bound *sup*(V) and a greatest lower bound *inf*(V), respectively.
- An operator $T\colon U \to U$ is
  - *monotone*, if for every x, $y \in U$ it holds that $x \leq y$ implies $T(x) \leq T(y)$
  - *continuous*, if $T(\mathit{sup}(V)) = \mathit{sup}(\{T(x) \mid x \in V\}$ for every $V \subseteq U$.

Notice: Continuous operators are monotone

Monotone and continuous operators have nice fixpoint properties

Fixpoint Theorems of Knaster-Tarski and Kleene

Theorem

Every monotone operator T on a complete lattice $(U, \leq)$ has a least fixpoint $\mathit{lfp}(T)$, and $\mathit{lfp}(T) = \mathit{inf}(\{x \in U \mid T(x) \leq x\})$.

A stronger theorem holds for continuous operators.

Theorem

Every continuous operator T on a complete lattice $(U, \leq)$ has a least fixpoint, and $\mathit{lfp}(T) = \mathit{sup}(\{T^i \mid i \geq 0\})$, where $T^0 = \mathit{inf}(U)$ and $T^{i+1} = T(T^i)$, for all $i \geq 0$.

Notation: $T^\infty = sup(\{T^i \mid i \geq 0\})$.

- Finite convergence: $T^k = T^{k-1}$ for some $k \Rightarrow T^\infty = T^k$
- A weaker form of Kleene's theorem holds for all monotone operators (transfinite sequence T^i).

Applying Fixpoint Theory

- For a logic program $\mathcal{P}$, the power set lattice $(P(\mathbf{HB}(\mathcal{P})), \subseteq)$ over the Herbrand base $\mathbf{HB}(\mathcal{P})$ is a complete lattice.
- We can associate with $\mathcal{P}$ an immediate consequence operator $T_{\mathcal{P}}$ on $\mathbf{HB}(\mathcal{P})$ such that $T_{\mathcal{P}}(I) = \{H(r) \mid r \in \mathit{Ground}(\mathcal{P}), B(r) \subseteq I\}$
- $T_{\mathcal{P}}$ is monotonic (in fact, continuous)
- Thus, $T_{\mathcal{P}}$ has the least fixpoint $\mathit{lfp}(T_{\mathcal{P}})$. It coincides with $T_{\mathcal{P}}^{\infty}$ and $\mathit{MM}(\mathcal{P})$

Theorem

Theorem. Given a datalog program P and a database instance $\mathbf{I}$,

$$P(\mathbf{I}) = \mathit{lfp}(T_{\mathcal{P}(P,\mathbf{I})}) = T^{\infty}_{\mathcal{P}(P\mathbf{I})}$$

Remark: Application of fixpoint theory is primarily of interest for infinite sets

Proof-Theoretic Approach

Basic idea: The answer of a datalog program P on $\mathbf{I}$ is given by the set of facts which can be *proved* from P and $\mathbf{I}$.

Definition (Proof tree)

A *proof tree* for a fact A from $\mathbf{I}$ and P is a labeled finite tree T such that

- each vertex of T is labeled by a fact
- the root of T is labeled by A
- each leaf of T is labeled by a fact in $\mathbf{I}$
- if a non-leaf of T is labeled with A_1 and its children are labeled with $A_2, \ldots, A_n$, then there exists a ground instance r of a rule in P such that $H(r) = A_1$ and $B(r) = \{A_2, \ldots, A_n\}$

Example (Same Generation)

Let

$$P = \{ r_1 \colon \mathtt{sgc(X, X)} \;\leftarrow\; \mathtt{person(X)}$$
$$r_2 \colon \mathtt{sgc(X, Y)} \;\leftarrow\; \mathtt{par(X, X1), sgc(X1, Y1), par(Y, Y1)} \}$$

where *edb*$(P) = \{\mathtt{person}, \mathtt{par}\}$ and *idb*$(P) = \{\mathtt{sgc}\}$

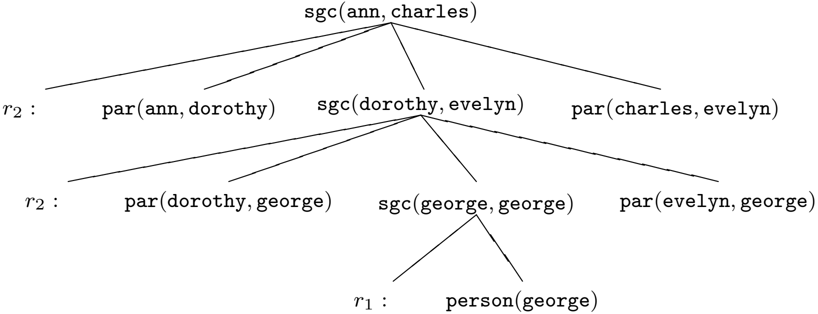
Consider $\mathbf{I}$ as follows:

$$\mathbf{I}(\mathtt{person}) = \{ \langle \mathtt{ann} \rangle,\ \langle \mathtt{bertrand} \rangle,\ \langle \mathtt{charles} \rangle, \langle \mathtt{dorothy} \rangle,$$
$$\langle \mathtt{evelyn} \rangle, \langle \mathtt{fred} \rangle,\ \langle \mathtt{george} \rangle,\ \langle \mathtt{hilary} \rangle \}$$

$$\mathbf{I}(\mathtt{par}) = \{ \langle \mathtt{dorothy}, \mathtt{george} \rangle,\ \langle \mathtt{evelyn}, \mathtt{george} \rangle,\ \langle \mathtt{bertrand}, \mathtt{dorothy} \rangle,$$
$$\langle \mathtt{ann}, \mathtt{dorothy} \rangle,\ \langle \mathtt{hilary}, \mathtt{ann} \rangle,\ \langle \mathtt{charles}, \mathtt{evelyn} \rangle \}.$$

Example (Same Generation)/2

Proof tree for $A = \mathtt{sgc}(\mathtt{ann}, \mathtt{charles})$ from $\mathbf{I}$ and P:



Proof Tree Construction

There are different ways to construct a proof tree for A from P and $\mathbf{I}$:

- *Bottom Up construction:* From leaves to root

  Intimately related to fixpoint approach

  - Define $S \vdash_P B$ to prove fact B from facts S if $B \in S$ or by a rule in P
  - Give $S = \mathbf{I}$ for granted

- *Top Down construction:* From root to leaves

  In Logic Programming view, consider program $\mathcal{P}(P, \mathbf{I})$.

  - This amounts to a set of logical sentences $H_{\mathcal{P}(P,\mathbf{I})}$ of the form

  $$\forall x_1 \cdots \forall x_m (R_1(\bar{x}_1) \vee \neg R_2(\bar{x}_2) \vee \neg R_3(\bar{x}_3) \vee \cdots \vee \neg R_n(\bar{x}_n))$$

  - Prove that $A = R(\bar{t})$ is a logical consequence via resolution refutation, that is, that $H_{\mathcal{P}(P,\mathbf{I})} \cup \{\neg A\}$ is unsatisfiable.

Datalog and SLD Resolution

- Logic Programming uses SLD resolution
- SLD: Selection Rule Driven Linear Resolution for Definite Clauses
- For datalog programs P on $\mathbf{I}$, resp. $\mathcal{P}(P, \mathbf{I})$, things are simpler than for general logic programs (no function symbols, unification is easy)

Let $\mathit{SLD}(\mathcal{P})$ be the set of ground atoms provable with SLD Resolution from $\mathcal{P}$.

Theorem

For any datalog program P and database instance $\mathbf{I}$,

$$\mathit{SLD}(\mathcal{P}(P, \mathbf{I})) = P(\mathbf{I}) = \mathbf{T}^{\infty}_{\mathcal{P}(P,\mathcal{I})} = \mathit{lfp}(\mathbf{T}_{\mathcal{P}(P,\mathcal{I})}) = \mathit{MM}(\mathcal{P}(P, \mathbf{I}))$$

SLD Resolution – Termination

- Notice: Selection rule for next rule/atom to be considered for resolution might affect termination
- Prolog's strategy (leftmost atom/first rule) is problematic

Example:

$$\texttt{child\_of}(\texttt{karl}, \texttt{franz}).$$

$$\texttt{child\_of}(\texttt{franz}, \texttt{frieda}).$$

$$\texttt{child\_of}(\texttt{frieda}, \texttt{pia}).$$

$$\texttt{descendent\_of}(\texttt{X}, \texttt{Y}) \leftarrow \texttt{child\_of}(\texttt{X}, \texttt{Y}).$$

$$\texttt{descendent\_of}(\texttt{X}, \texttt{Y}) \leftarrow \texttt{child\_of}(\texttt{X}, \texttt{Z}), \texttt{descendent\_of}(\texttt{Z}, \texttt{Y}).$$

$$\leftarrow \texttt{descendent\_of}(\texttt{karl}, \texttt{X}).$$

SLD Resolution – Termination/2

Example (cntd.):

$$\mathtt{child\_of}(\mathtt{karl}, \mathtt{franz}).$$

$$\mathtt{child\_of}(\mathtt{franz}, \mathtt{frieda}).$$

$$\mathtt{child\_of}(\mathtt{frieda}, \mathtt{pia}).$$

$$\mathtt{descendent\_of}(\mathtt{X}, \mathtt{Y}) \leftarrow \mathtt{child\_of}(\mathtt{X}, \mathtt{Y}).$$

$$\mathtt{descendent\_of}(\mathtt{X}, \mathtt{Y}) \leftarrow \mathsf{descendent\_of}(\mathsf{X}, \mathsf{Z}), \mathsf{child\_of}(\mathsf{Z}, \mathsf{Y}).$$

$$\leftarrow \mathtt{descendent\_of}(\mathtt{karl}, \mathtt{X}).$$

SLD Resolution – Termination /3

Example (cntd.):

$$\begin{aligned}
&\texttt{child_of}(\texttt{karl}, \texttt{franz}).\\
&\texttt{child_of}(\texttt{franz}, \texttt{frieda}).\\
&\texttt{child_of}(\texttt{frieda}, \texttt{pia}).\\
&\texttt{descendent_of}(\texttt{X}, \texttt{Y}) \leftarrow \texttt{child_of}(\texttt{X}, \texttt{Y}).\\
&\texttt{descendent_of}(\texttt{X}, \texttt{Y}) \leftarrow \texttt{descendent_of}(\texttt{X}, \texttt{Z}),\\
&\qquad\qquad\qquad\qquad\quad\ \texttt{descendent_of}(\texttt{Z}, \texttt{Y}).\\
&\leftarrow \texttt{descendent_of}(\texttt{karl}, \texttt{X}).
\end{aligned}$$

Exercise: Metro Reachability

Over the Metro database, consider the predicates `reachableFromOne`$/3$ and `reachableFromBoth`$/3$, with the following meaning for stations a, b, and c:

1. `reachableFromOne`(a, b, c) holds if c is reachable from one of a or b;
2. `reachableFromBoth`(a, b, c) holds if c is reachable from both of a and b.

Write datalog rules that define these predicates.