Foundations of Databases

Datalog

Free University of Bozen – Bolzano, 2004–2005

Thomas Eiter

Institut für Informationssysteme

Arbeitsbereich Wissensbasierte Systeme (184/3)

Technische Universität Wien

`http://www.kr.tuwien.ac.at/staff/eiter`

(Some slides by Wolfgang Faber)

Motivation

- Relational Calculus and Relational Algebra were considered to be "*the*" database languages for long time
- Codd: A query language is "complete," if it yields precisely 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
- Such queries can provably be not expressed in Relational Calculus
- This motivated a logic-programming extension to conjunctive queries: *datalog*

Example: Metro database instance

links	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 first query:

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

Note: recursive definition

Intuitively, a rule “fires” if the part right of $\leftarrow$ is true, and the atom left of $\leftarrow$ is concluded.

Datalog Language

- datalog is akin to logic programming
- The basic language (considered next) has many extensions
- Different approaches for defining the semantics exist:

  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 versus Logic Programming

- As stated above, Datalog is akin to logic programming, in particular to Prolog
- There are important differences, though:
  - There are no functions symbols in datalog. Consequently, no potentially infinite data structures such as lists are supported
  - Datalog has a purely declarative semantics. In a datalog program,
    - the order of clauses is irrelevant
    - the order of atoms in a rule body is irrelevant
  - Datalog programs adhere to the active domain semantics (like Safe Relational Calculus, Relational Algebra)
  - Datalog distinguishes between databases relations ("*extensional database*", *edb*) and derived relations ("*intensional database*", *idb*)

Datalog Syntax

Consider here the simplest version of datalog ("plain datalog", or "datalog").

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

$$R_1(\vec{x}_1) \leftarrow R_2(\vec{x}_2), \ldots, R_n(\vec{x}_n) \quad (1)$$

where $n \geq 1$ and

- $R_1, \ldots, R_n$ are relations names
- $\vec{x}_1, \ldots, \vec{x}_n$ are vectors of variables and constants (from $\mathbf{dom}$)
- Every variable in $\vec{x}_1$ must occur in $\vec{x}_2, \ldots, \vec{x}_n$ ("safety")
- the head of r is $R(\vec{x}_1)$, denoted $H(r)$.
- the body of r is $\{\ R_2(\vec{x}_2), \ldots, R_n(\vec{x}_n)\ \}$, denoted $B(r)$.
- Remark: The rule atom "$\longleftarrow$" is often also written as "$\mathbf{:}-$"

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

Datalog Syntax /2

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).

Note:

- Sometimes, extensional and intensional relations are explicitly specified. Intensional relations may then also occur only in rule bodies (but are of no use then)
- In a logic programming view, the term “predicate” is used as synonym for “relation (name)”.

Metro Example

Datalog program P on metro database scheme
$\mathcal{M} = \{\texttt{links} : \texttt{line}, \texttt{station}, \texttt{nextstation}\}$:

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

Here,

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

Datalog Syntax /3

- The set of constants occurring in a datalog program P, is denoted by $adom(P)$.
- Given a database instance $\mathbf{I}$, $adom(P, \mathbf{I})$ denotes $adom(P) \cup adom(\mathbf{I})$, i.e., the set of constants occurring in P and $\mathbf{I}$
- $adom(P, \mathbf{I})$ is the *active domain* of P with respect to $\mathbf{I}$.

Defn. Given a valuation $\nu : var(r) \cup \mathbf{dom} \rightarrow \mathbf{dom}$ for a rule r of form (1), the instantiation of r with ν, denoted $\nu(r)$, is the rule

$$R_1(\nu(\vec{x}_1)) \leftarrow R_2(\nu(\vec{x}_2)), \ldots, R_n(\nu(\vec{x}_n))$$

which results by replacing each variable x with $\nu(x)$.

Metro Example

- For datalog program P above, $adom(P)$ = { Odeon }
- Database instance $\mathbf{I}$:

links	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

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

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

Metro Example /2

- The rule

$$\texttt{reach}(\texttt{St.Germain}, \texttt{Odeon}) \quad \leftarrow \quad \texttt{links}(\texttt{Louvres}, \texttt{St.Germain}, \texttt{Concorde}), \texttt{reach}(\texttt{Concorde}, \texttt{Odeon})$$

is an instance of the rule

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

of P:

take $\nu(X)$ = St.Germain, $\nu(L)$ = Louvres, $\nu(Y)$=Odeon, $\nu(Z)$=Concorde

Datalog: Model-Theoretic Semantics

- Key Idea: View program as a set of first-order sentences
- The database instance constituting the result satisfies the sentences (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

- Associate with each datalog rule r of form

$$R_1(\vec{x}_1) \leftarrow R_2(\vec{x}_2), \ldots, R_n(\vec{x}_n)$$

  the logical sentence $\sigma(r)$:

$$\forall x_1, \cdots \forall x_m \, (R_2(\vec{x}_2) \wedge \cdots \wedge R_n(\vec{x}_n) \rightarrow R_1(\vec{x}_1))$$

  where $x_1, \ldots, x_m$ are the variables in r.

- Associate with P the set of sentences $\Sigma_P = \{\sigma(r) \mid r \in P\}$.

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

- A *model* of P is an instance of $sch(P)$ which satisfies Σ_P.
- The semantics of P on input $\mathbf{I}$, denoted $P(\mathbf{I})$, is the least model of P (unique minimal model wrt set inclusion $\subseteq$) containing $\mathbf{I}$, if it exists.

Example

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

links	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	Louvres
	...	

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

Questions

- Is the semantics $P(\mathbf{I})$ well-defined for each input database $\mathbf{I}$?
- How to compute $P(\mathbf{I})$?

Observation: For any $\mathbf{I}$, program P has some model containing $\mathbf{I}$

- Let $\mathbf{B}(P, \mathbf{I})$ be the instance of $sch(P)$ such that
  - for each $R \in edb(P)$, $\mathbf{B}(P, \mathbf{I})(R) = \mathbf{I}(R)$
  - for each $R \in idb(P)$, $\mathbf{B}(P, \mathbf{I})(R) = adom(P, \mathbf{I})^{arity(R)}$
- Then, $\mathbf{B}(P, \mathbf{I})$ is a model of P containing $\mathbf{I}$.
- $\Rightarrow P(\mathbf{I})$ is a subset of $\mathbf{B}(P, \mathbf{I})$.
- Naive algorithm: explore all subsets of $\mathbf{B}(P, \mathbf{I})$.

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

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

1. $\mathbf{M} = \bigcap \mathcal{M}(\mathbf{I})$ is the least model of P, where $\mathcal{M}(\mathbf{I})$ is the set of all models of P containing $\mathbf{I}$.
2. $adom(P(\mathbf{I})) \subseteq adom(P, \mathbf{I})$, i.e., no new values appear
3. For each $R \in edb(P)$, $P(\mathbf{I})(R) = \mathbf{I}(R)$.

Consequence:

- $P(\mathbf{I})$ is well-defined, for each $\mathbf{I}$
- A query $Q(\vec{x})$ defined by a distinguished "output" relation $q \in idb(P)$ has always finite result
- Effective methods exist to compute $Q(\mathbf{I})$

Why choosing the least model?

Two reasons to choose the least model containing $\mathbf{I}$:

1. The *Closed World Assumption*:
   - If a fact $R(\vec{c})$ is not true in all models of a database $\mathbf{I}$, then infer that $R(\vec{c})$ is false.
   - This amounts to considering $\mathbf{I}$ as complete
   - This 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 has no distinction between edb and idb.
- A datalog program P and an instance $\mathbf{I}$ of $edb(P)$ can be mapped to a logic program $\mathcal{P}(P, \mathbf{I})$ given by

$$\mathcal{P}(P, \mathbf{I}) = P \cup \{R(\vec{t}) \mid R \in edb(P), \vec{t} \in \mathbf{I}(R)\}.$$

- As a logical theory, this amounts to

$$\Sigma_{P,\mathbf{I}} = \Sigma_P \cup \{R(\vec{t}) \mid R \in edb(P), \vec{t} \in \mathbf{I}(R)\}.$$

- The semantics of $\mathcal{P} = \mathcal{P}(P, \mathbf{I})$ is defined in terms of *Herbrand interpretations* of the language induced by $\mathcal{P}$:

  The domain of discourse are 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, let $Const(r)$ be the set of all constants in r.

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

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

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

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

$$\mathbf{HB}(\mathcal{P}) = \{R(x_1, \ldots, x_n) \mid R \text{ is a relation (predicate) in } \mathcal{P}, \\ x_1, \ldots, x_n \in \mathbf{HU}(\mathcal{P}), ar(R) = n\}$$

Example

$$\mathcal{P} = \{ \quad \texttt{arc}(\mathtt{a}, \mathtt{b}).$$

$$\texttt{arc}(\mathtt{b}, \mathtt{c}).$$

$$\texttt{reachable}(\mathtt{a}).$$

$$\texttt{reachable}(\mathtt{Y}) \leftarrow \texttt{arc}(\mathtt{X}, \mathtt{Y}), \texttt{reachable}(\mathtt{X}). \}$$

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

Grounding

- A rule r' is a *ground instance* of a rule r with respect to $\mathbf{HU}(\mathcal{P})$, if $r' = \nu(r)$ for a valuation ν such that $\nu(x) \in \mathbf{HU}(\mathcal{P})$ for each $x \in var(r)$.
- The grounding of a rule r with respect to $\mathbf{HU}(\mathcal{P})$, denoted $Ground_{\mathcal{P}}(r)$, is the set of all ground instances of r wrt $\mathbf{HU}(\mathcal{P})$.
- The grounding of a logic program $\mathcal{P}$ is

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

Example

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

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 any $M \subseteq \mathbf{HB}(\mathcal{P})$ such that

$$\forall r \in Ground(\mathcal{P}) : (H(r) \subseteq M) \vee (B(r) \not\subseteq M)$$

Equivalently:

$$\forall r \in Ground(\mathcal{P}) : (B(r) \subseteq M) \rightarrow (H(r) \subseteq M)$$

Example

Herbrand models of program $\mathcal{P}$ above:

- $M_1 = \{ \quad \texttt{arc}(\texttt{a}, \texttt{b}), \texttt{arc}(\texttt{b}, \texttt{c}),$
  $\texttt{reachable}(\texttt{a}), \texttt{reachable}(\texttt{b}), \texttt{reachable}(\texttt{c})\}$
- $M_2 = \mathbf{HB}(\mathcal{P})$
- Every interpretation M such that $M_1 \subseteq M \subseteq M_2$ (and no others)

Logic Programming Semantics

- **Theorem**. $\mathbf{HB}(\mathcal{P})$ is always a model of $\mathcal{P}$.

- **Theorem**. Each logic program $\mathcal{P}$ has the least (wrt $\subseteq$) model, denoted $MM(\mathcal{P})$.

  The model $MM(\mathcal{P})$ is the semantics of $\mathcal{P}$.

- **Theorem (Datalog $\leftrightarrow$ Logic Programming)**. Let P be a datalog program and $\mathbf{I}$ be an instance of $edb(P)$. Then,

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

Implications

- Results and techniques for logic programming can be exploited for datalog

  E.g.,

  - proof procedures for logic programming (e.g., SLD resolution) can be used to datalog (with some caveats)
  - Datalog can be reduced by “grounding” to propositional logic programs (utilized e.g. by the systems DLV and Smodels)

Fixpoint Semantics

Different view:

> "If we assume that all facts in $\mathbf{I}$ are true, which other facts must be true by firing rules in P?"

Approach:

- Define an *immediate consequence operator* $\mathbf{T}_P(\mathbf{K})$ on db instances.
- Start with $\mathbf{K} = \mathbf{I}$.
- Apply $\mathbf{T}_P$: $\mathbf{K}_{new} := \mathbf{T}_P(\mathbf{K}) = \mathbf{K} \cup$ 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 $R(\vec{t})$ is an *immediate* consequence for $\mathbf{K}$ and P, if either

- $R \in edb(P)$ and $R(\vec{t}) \in \mathbf{K}$, or
- there exists some ground instance r of a rule in P such that $Head(r) = R(\vec{t})$ and $Body(r) \subseteq \mathbf{K}$.

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

$$\mathbf{T}_P : inst(sch(P)) \rightarrow inst(sch(P))$$

such that

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

Example

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

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

$$\begin{array}{lcl}
\mathbf{K}_1 & = & \{\texttt{arc}(\texttt{a}, \texttt{b}),\ \texttt{arc}(\texttt{b}, \texttt{c})\} \\
\mathbf{K}_2 & = & \{\texttt{arc}(\texttt{a}, \texttt{b}),\ \texttt{arc}(\texttt{b}, \texttt{c}),\ \texttt{reachable}(b)\} \\
\mathbf{K}_3 & = & \{\texttt{arc}(\texttt{a}, \texttt{b}),\ \texttt{arc}(\texttt{b}, \texttt{c}), \texttt{reachable}(\texttt{a}),\ \texttt{reachable}(\texttt{b}),\ \texttt{reachable}(\texttt{c})\}
\end{array}$$

Then,

$$\begin{array}{rcl}
\mathbf{T}_P(\emptyset) & = & \{\texttt{reachable}(\texttt{a})\} \\
\mathbf{T}_P(\mathbf{K}_1) & = & \{\texttt{arc}(\texttt{a}, \texttt{b}),\ \texttt{arc}(\texttt{b}, \texttt{c}), \texttt{reachable}(\texttt{a})\} \ = \ \mathbf{K}_2 \\
\mathbf{T}_P(\mathbf{K}_2) & = & \{\texttt{arc}(\texttt{a}, \texttt{b}),\ \texttt{arc}(\texttt{b}, \texttt{c}), \texttt{reachable}(\texttt{a}),\ \texttt{reachable}(\texttt{b})\} \\
\mathbf{T}_P(\mathbf{K}_3) & = & \{\texttt{arc}(\texttt{a}, \texttt{b}),\ \texttt{arc}(\texttt{b}, \texttt{c}), \texttt{reachable}(\texttt{a}),\ \texttt{reachable}(\texttt{b}),\ \texttt{reachable}(\texttt{c})\} \\
 & = & \mathbf{K}_3
\end{array}$$

Thus, $\mathbf{K}_3$ is a *fixpoint* of $\mathbf{T}_P$ (i.e., $\mathbf{T}_P(\mathbf{K}) = \mathbf{K}$).

Properties

Lemma. For every datalog program P,

1. the operator $\mathbf{T}_P$ is monotonic, i.e., $\mathbf{K} \subseteq \mathbf{K}'$ implies $\mathbf{T}_P(\mathbf{K}) \subseteq \mathbf{T}_P(\mathbf{K}')$;
2. $\mathbf{K} \in inst(sch(P))$ is a model of $\Sigma_P \Leftrightarrow \mathbf{T}_P(\mathbf{K}) \subseteq \mathbf{K}$;
3. If $\mathbf{T}_P(\mathbf{K}) = \mathbf{K}$ then $\mathbf{K}$ is a model of Σ_P.

Note: The converse of 3. fails in general.

Datalog Semantics via Least Fixpoint

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

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

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

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

Fixpoint Iteration

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

Define sequence $\{\mathbf{I}_i\}_{i \geq 0}$ by

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

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

This can be readily implemented by an algorithm

Example

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

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

Then,

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

Thus, $\mathbf{T}_P^\omega(\mathbf{I}) = lfp(P, \mathbf{I}) = \mathbf{I}_3$.

Excursion: Fixpoint Theory

- Evaluating a datalog program P on $\mathbf{I}$ amounts to evaluation 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 $T : U \rightarrow U$ is
  - *monotone*, if for every $x, y \in U$ it holds that $x \leq y \Rightarrow T(x) \leq T(y)$,
  - *continuous*, if $T(sup(V)) = sup(\{T(x) \mid x \in V\}$ 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 (Knaster-Tarksi). Every monotone operator T on a complete lattice $(U, \leq)$ has a least fixpoint $lfp(T)$, and $lfp(T) = inf(\{x \in U \mid T(x) \leq x\})$.

A stronger theorem holds for continuous operators.

Theorem (Kleene). Every continuous operator T on a complete lattice $(U, \leq)$ has a least fixpoint, and $lfp(T) = sup(\{T^i \mid i \geq 0\})$, where $T^0 = inf(U)$ 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 $T_{\mathcal{P}}(I) = \{H(r) \mid r \in Ground(\mathcal{P}), B(r) \subseteq I\}$
- $T_{\mathcal{P}}$ is monotonic (in fact, continuous)
- Thus, $T_{\mathcal{P}}$ has the least fixpoint $lfp(T_{\mathcal{P}})$. It coincides with $T_{\mathcal{P}}^{\infty}$ and $MM(\mathcal{P})$

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

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

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 be the set of facts which can be *proved* from P and $\mathbf{I}$.

Defn. A *proof tree* of 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)

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

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

Consider $\mathbf{I}$ as follows:

$$\begin{aligned}
\mathbf{I}(person) \quad &= \{ \quad \langle ann\rangle, \langle bertrand\rangle, \langle charles\rangle, \langle dorothy\rangle, \\
&\qquad \langle evelyn\rangle, \langle fred\rangle, \langle george\rangle, \langle hilary\rangle\} \\
\mathbf{I}(par) \quad &= \{ \quad \langle dorothy, george\rangle, \langle evelyn, george\rangle, \langle bertrand, dorothy\rangle, \\
&\qquad \langle ann, dorothy\rangle, \langle ann, hilary\rangle, \langle charles, evelyn\rangle\}.
\end{aligned}$$

Example (Same Generation)/2

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

$\texttt{sgc}(\texttt{ann}, \texttt{charles})$

r_2 : $\texttt{par}(\texttt{ann}, \texttt{dorothy})$ $\texttt{sgc}(\texttt{dorothy}, \texttt{evelyn})$ $\texttt{par}(\texttt{evelyn}, \texttt{george})$

r_2 : $\texttt{par}(\texttt{dorothy}, \texttt{george})$ $\texttt{sgc}(\texttt{george}, \texttt{george})$ $\texttt{par}(\texttt{evelyn}, \texttt{george})$

r_1 : $\texttt{person}(\texttt{george})$

Proof Tree Construction

Different ways to construct a proof tree for A from P and $\mathbf{I}$ exist

- ***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

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

  – Prove $A = R(\vec{t})$ via resolution refutation, i.e., 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)
- Also non-ground atoms can be handled (e.g., $\mathtt{sgc}(\mathtt{ann}, \mathtt{X})$)

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

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

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

SLD Resolution – Termination

- Notice: Selection rule for next rule / atom to be considered for resolution might effect 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

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

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

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

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

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

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

SLD Resolution – Termination /3

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

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

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

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

$$\begin{aligned}\texttt{descendent\_of}(\texttt{X}, \texttt{Y}) \leftarrow\ & \texttt{descendent\_of}(\texttt{X}, \texttt{Z}),\\ & \texttt{descendent\_of}(\texttt{Z}, \texttt{Y}).\end{aligned}$$

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

Bibliography

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems – The Complete Book*. Prentice Hall, 2002.