# Computational Logic

## Datalog

Free University of Bozen-Bolzano, 2010

Werner Nutt

(Based on slides by Thomas Eiter and Wolfgang Faber)

# 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

| 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:

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

Note: 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. 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
  - 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

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

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

**Remarks.**

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

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

# Datalog Programs

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*($P$).

**Remarks.**

- Sometimes, extensional and intensional relations are explicitly specified. It is possible then for intensional relations to occur only in rule bodies (but such relations are of no use then).
- In a Logic Programming view, the term “predicate” is used as synonym for “relation” or “relation name.”

# The Metro Example /1

Datalog program $P$ on metro database scheme

$\mathcal{M} = \{\mathtt{links} : \mathtt{line},\ \mathtt{station},\ \mathtt{nextstation}\}$:

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

Here,

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

# Datalog Syntax (cntd)

- The set of constants occurring in a datalog program $P$ is denoted as $adom(P)$
- Given a database instance $\mathbf{I}$, we define the *active domain* of $P$ with respect to $I$ as

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

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

**Definition.** Let $\nu\colon\, var(r) \cup \mathbf{dom} \rightarrow \mathbf{dom}$ be a valuation for a rule $r$ of form (1). Then the *instantiation* of $r$ with $\nu$, denoted $\nu(r)$, is the rule

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

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

# The Metro Example /2

- For the datalog program $P$ above, we have that $adom(P)$ = $\{$ Odeon $\}$
- We consider the 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 |

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

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

## The Metro Example /3

- The rule

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

is an instance of the rule

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

of $P$:

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

# Datalog: Model-Theoretic Semantics

**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 $\Sigma_P$

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

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

- 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 $\Sigma_P$
- 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:

| 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 every input instance $\mathbf{I}$?
- 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

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

- Then: $\mathbf{B}(P, \mathbf{I})$ is a model of $P$ containing $\mathbf{I}$

  $\Rightarrow$ $P(\mathbf{I})$ is a subset of $\mathbf{B}(P, \mathbf{I})$ *(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 $R(\vec{c})$ 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
   - . . . 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

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

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

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

## Example

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

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

## 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 $\nu$ 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

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

## Example

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

## 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 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:

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

and no others.

# Logic Programming Semantics

- **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 *MM*$(\mathcal{P})$ (for "minimal model").

  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,

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

# 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: $\mathbf{K}_{new} := \mathbf{T}_P(\mathbf{K}) = \mathbf{I} \cup$ 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 a ground instance $r$ of a rule in $P$ such that $H(r) = R(\vec{t})$ and $B(r) \subseteq \mathbf{K}$.

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

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

where

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

## Example

Consider

$$P = \{ \quad \texttt{reachable}(\texttt{a})$$

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

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

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

## Example (cntd)

Then,

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

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.** For every datalog program $P$ we have:

1. The operator $\mathbf{T}_P$ is monotonic, that is, $\mathbf{K} \subseteq \mathbf{K}'$ implies $\mathbf{T}_P(\mathbf{K}) \subseteq \mathbf{T}_P(\mathbf{K}')$;
2. For any $\mathbf{K} \in inst(sch(P))$ 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 $\Sigma_P$.

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

# Datalog Semantics via Least Fixpoint

The semantics of $P$ on 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, $lfp(P, \mathbf{I}) = \mathit{MM}(\mathcal{P}(P, \mathbf{I})) = P(\mathbf{I})$.

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

**Proof** of Claim 2, first equality (Sketch): Let $M_1 := lfp(P, \mathbf{I})$ and $M_2 := \mathit{MM}(\mathcal{P}(P, \mathbf{I}))$.

Since $M_1$ is a fixpoint of $\mathbf{T}_P$, it is a model of $\Sigma_P$, 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 $M_1 \subseteq M_2$.

## Fixpoint Iteration

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

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

- 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

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

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

Then,

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

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

## 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.** 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)

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

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

Consider $\mathbf{I}$ as follows:

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

## Example (Same Generation)/2

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

- $\texttt{sgc}(\texttt{ann}, \texttt{charles})$
  - $r_2$ :
  - $\texttt{par}(\texttt{ann}, \texttt{dorothy})$
  - $\texttt{sgc}(\texttt{dorothy}, \texttt{evelyn})$
    - $r_2$ :
    - $\texttt{par}(\texttt{dorothy}, \texttt{george})$
    - $\texttt{sgc}(\texttt{george}, \texttt{george})$
      - $r_1$ :
      - $\texttt{person}(\texttt{george})$
    - $\texttt{par}(\texttt{evelyn}, \texttt{george})$
  - $\texttt{par}(\texttt{charles}, \texttt{evelyn})$

## 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, 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)
- 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 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

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

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

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

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

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

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

## SLD Resolution – Termination /3

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

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

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

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

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

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