Review: Datalog

A rule-based recursive query language

father(alice, bob)
mother(alice, carla)
Parent(x, y) ← father(x, y)
Parent(x, y) ← mother(x, y)
SameGeneration(x, x)
SameGeneration(x, y) ← Parent(x, v) ∧ Parent(y, w) ∧ SameGeneration(v, w)

- Datalog is more complex than FO query answering
- Datalog is more expressive than FO query answering
- Semipositive Datalog with a successor ordering captures P
- Datalog containment is undecidable

Remaining question: How can Datalog query answering be implemented?
Implementing Datalog

FO queries (and thus also CQs and UCQs) are supported by almost all DBMS
→ many specific implementation and optimisation techniques

How can Datalog queries be answered in practice?
→ techniques for dealing with recursion in DBMS query answering

There are two major paradigms for answering recursive queries:
  • **Bottom-up**: derive conclusions by applying rules to given facts
  • **Top-down**: search for proofs to infer results given query
Computing Datalog Query Answers Bottom-Up

We already saw a way to compute Datalog answers bottom-up: the step-wise computation of the consequence operator $T_P$.

Bottom-up computation is known under many names:

- **Forward-chaining** since rules are “chained” from premise to conclusion (common in logic programming)
- **Materialisation** since inferred facts are stored (“materialised”) (common in databases)
- **Saturation** since the input database is “saturated” with inferences (common in theorem proving)
- **Deductive closure** since we “close” the input under entailments (common in formal logic)
Naive Evaluation of Datalog Queries

A direct approach for computing $T_P^\infty$

```
01 $T_P^0 := \emptyset$
02 $i := 0$
03 repeat :
04 $T_{i+1}^P := \emptyset$
05 for $H \leftarrow B_1 \land \ldots \land B_\ell \in P$ :
06   for $\theta \in B_1 \land \ldots \land B_\ell(T_i^P)$ :
07     $T_{i+1}^P := T_{i+1}^P \cup \{H\theta\}$
08     $i := i + 1$
09 until $T_{i-1}^P = T_i^P$
10 return $T_i^P$
```

Notation for line 06/07:

- a substitution $\theta$ is a mapping from variables to database elements
- for a formula $F$, we write $F\theta$ for the formula obtained by replacing each free variable $x$ in $F$ by $\theta(x)$
- for a CQ $Q$ and database $I$, we write $\theta \in Q(I)$ if $I \models Q\theta$
What’s Wrong with Naive Evaluation?

An example Datalog program:

\[
\text{e}(1, 2) \quad \text{e}(2, 3) \quad \text{e}(3, 4) \quad \text{e}(4, 5)
\]

\[
(R1) \quad \text{T}(x, y) \leftarrow \text{e}(x, y)
\]

\[
(R2) \quad \text{T}(x, z) \leftarrow \text{T}(x, y) \land \text{T}(y, z)
\]

How many body matches do we need to iterate over?

\[
T_P^0 = \emptyset \quad \text{initialisation}
\]

\[
T_P^1 = \{\text{T}(1, 2), \text{T}(2, 3), \text{T}(3, 4), \text{T}(4, 5)\} \quad 4 \text{ matches for (R1)}
\]

\[
T_P^2 = T_P^1 \cup \{\text{T}(1, 3), \text{T}(2, 4), \text{T}(3, 5)\} \quad 4 \times (R1) + 3 \times (R2)
\]

\[
T_P^3 = T_P^2 \cup \{\text{T}(1, 4), \text{T}(2, 5), \text{T}(1, 5)\} \quad 4 \times (R1) + 8 \times (R2)
\]

\[
T_P^4 = T_P^3 = T_P^\infty \quad 4 \times (R1) + 10 \times (R2)
\]

In total, we considered 37 matches to derive 11 facts
Does it really matter how often we consider a rule match? After all, each fact is added only once . . .

In practice, finding applicable rules takes significant time, even if the conclusion does not need to be added – iteration takes time!

\( \leadsto \) huge potential for optimisation

**Observation:**
we derive the same conclusions over and over again in each step

**Idea:** apply rules only to newly derived facts

\( \leadsto \) semi-naive evaluation
Semi-Naive Evaluation

The computation yields sets $T^0_P \subseteq T^1_P \subseteq T^2_P \subseteq \ldots \subseteq T^\infty_P$

- For an IDB predicate $R$, let $R^i$ be the “predicate” that contains exactly the $R$-facts in $T^i_P$
- For $i \leq 1$, let $\Delta^i_R$ be the collection of facts $R^i \setminus R^{i-1}$

We can restrict rules to use only some computations.

Some options for the computation in step $i + 1$:

\[
\begin{align*}
T(x, z) &\leftarrow T^i(x, y) \land T^i(y, z) & \text{same as original rule} \\
T(x, z) &\leftarrow \Delta^i_T(x, y) \land \Delta^i_T(y, z) & \text{restrict to new facts} \\
T(x, z) &\leftarrow \Delta^i_T(x, y) \land T^i(y, z) & \text{partially restrict to new facts} \\
T(x, z) &\leftarrow T^i(x, y) \land \Delta^i_T(y, z) & \text{partially restrict to new facts}
\end{align*}
\]

What to choose?
Semi-Naive Evaluation (2)

Inferences that involve new and old facts are necessary:

\[ e(1, 2) \quad e(2, 3) \quad e(3, 4) \quad e(4, 5) \]

\[ (R1) \quad T(x, y) \leftarrow e(x, y) \]

\[ (R2) \quad T(x, z) \leftarrow T(x, y) \land T(y, z) \]

\[ T_0^P = \emptyset \]

\[ T_1^P = \Delta_1^T \]

\[ \Delta_1^T = \{ T(1, 2), T(2, 3), T(3, 4), T(3, 4), T(4, 5) \} \]

\[ T_2^P = T_1^P \cup \Delta_2^T \]

\[ \Delta_2^T = \{ T(1, 3), T(2, 4), T(3, 5) \} \]

\[ T_3^P = T_2^P \cup \Delta_3^T \]

\[ \Delta_3^T = \{ T(1, 4), T(2, 5), T(1, 5) \} \]

\[ T_4^P = T_3^P = T_\infty^P \]

\[ \Delta_4^T = \emptyset \]

To derive \( T(1, 4) \) in \( \Delta_3^T \), we need to combine
\[ T(1, 3) \in \Delta_2^T \text{ with } T(3, 4) \in \Delta_1^T \text{ or } T(1, 2) \in \Delta_1^T \text{ with } T(2, 4) \in \Delta_2^T \]

\( \rightarrow \) rule \( T(x, z) \leftarrow \Delta_1^T(x, y) \land \Delta_1^T(y, z) \) is not enough
Semi-Naive Evaluation (3)

**Correct approach:** consider only rule application that use at least one newly derived IDB atom

For example program:

\[
\begin{align*}
&\text{e}(1, 2) \quad \text{e}(2, 3) \quad \text{e}(3, 4) \quad \text{e}(4, 5) \\
&(R1) \quad T(x, y) \leftarrow \text{e}(x, y) \\
&(R2.1) \quad T(x, z) \leftarrow \Delta_T^i(x, y) \land T^i(y, z) \\
&(R2.2) \quad T(x, z) \leftarrow T^i(x, y) \land \Delta_T^i(y, z)
\end{align*}
\]

There is still redundancy here: the matches for \(T(x, z) \leftarrow \Delta_T^i(x, y) \land \Delta_T^i(y, z)\) are covered by both \((R2.1)\) and \((R2.2)\)

\(\mapsto\) replace \((R2.2)\) by the following rule:

\[
(R2.2') \quad T(x, z) \leftarrow T^{i-1}(x, y) \land \Delta_T^i(y, z)
\]

EDB atoms do not change, so their \(\Delta\) would be \(\emptyset\)

\(\mapsto\) ignore such rules after the first iteration
Semi-Naive Evaluation: Example

\[
\begin{align*}
\text{e}(1, 2) & \quad \text{e}(2, 3) & \quad \text{e}(3, 4) & \quad \text{e}(4, 5) \\
(R1) & \quad T(x, y) & \leftarrow & \text{e}(x, y) \\
(R2.1) & \quad T(x, z) & \leftarrow & \Delta^i_T(x, y) \land T^i(y, z) \\
(R2.2') & \quad T(x, z) & \leftarrow & T^{i-1}(x, y) \land \Delta^i_T(y, z)
\end{align*}
\]

How many body matches do we need to iterate over?

\[
\begin{align*}
T^0_P & = \emptyset & & \text{initialisation} \\
T^1_P & = \{T(1, 2), T(2, 3), T(3, 4), T(4, 5)\} & & 4 \times (R1) \\
T^2_P & = T^1_P \cup \{T(1, 3), T(2, 4), T(3, 5)\} & & 3 \times (R2.1) \\
T^3_P & = T^2_P \cup \{T(1, 4), T(2, 5), T(1, 5)\} & & 3 \times (R2.1), 2 \times (R2.2') \\
T^4_P & = T^3_P = T^\infty_P & & 1 \times (R2.1), 1 \times (R2.2')
\end{align*}
\]

In total, we considered 14 matches to derive 11 facts
Semi-Naive Evaluation: Full Definition

In general, a rule of the form

\[ H(\vec{x}) \leftarrow e_1(\vec{y}_1) \land \ldots \land e_n(\vec{y}_n) \land I_1(\vec{z}_1) \land I_2(\vec{z}_2) \land \ldots \land I_m(\vec{z}_m) \]

is transformed into \( m \) rules

\begin{align*}
H(\vec{x}) & \leftarrow e_1(\vec{y}_1) \land \ldots \land e_n(\vec{y}_n) \land \Delta^i_{11}(\vec{z}_1) \land I^i_2(\vec{z}_2) \land \ldots \land I^i_m(\vec{z}_m) \\
H(\vec{x}) & \leftarrow e_1(\vec{y}_1) \land \ldots \land e_n(\vec{y}_n) \land I^{i-1}_1(\vec{z}_1) \land \Delta^i_{12}(\vec{z}_2) \land \ldots \land I^i_m(\vec{z}_m) \\
& \quad \ldots \\
H(\vec{x}) & \leftarrow e_1(\vec{y}_1) \land \ldots \land e_n(\vec{y}_n) \land I^{i-1}_1(\vec{z}_1) \land I^{i-1}_2(\vec{z}_2) \land \ldots \land \Delta^i_m(\vec{z}_m)
\end{align*}

Advantages and disadvantages:

- Huge improvement over naive evaluation
- Some redundant computations remain (see example)
- Some overhead for implementation (store level of entailments)
Summary and Outlook

Datalog queries can be evaluated bottom-up or top-down

Simplest practical bottom-up technique: semi-naive evaluation

Next question:

- Can we improve Datalog evaluation further?
- What about practical implementations?