Review: Datalog

A rule-based recursive query language

father(alice, bob)
mother(alice, carla)

\[
\begin{align*}
\text{Parent}(x, y) & \leftarrow \text{father}(x, y) \\
\text{Parent}(x, y) & \leftarrow \text{mother}(x, y)
\end{align*}
\]

\[
\begin{align*}
\text{SameGeneration}(x, x) \\
\text{SameGeneration}(x, y) & \leftarrow \text{Parent}(x, v) \land \text{Parent}(y, w) \land \text{SameGeneration}(v, w)
\end{align*}
\]

• 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 \( T_r \)

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 \( T_p^{i+1} := \emptyset \)
05 for \( H \leftarrow B_1 \land \ldots \land B_r \in P \):
06 for \( \theta \in B_1 \land \ldots \land B_r(T_p^i) \):
07 \( T_p^{i+1} := T_p^{i+1} \cup \{ H \theta \} \)
08 i := i + 1
09 until \( T_p^{i+1} = T_p^i \)
10 return \( T_p^i \)
```

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 \( I \models Q\theta \)

Less Naive Evaluation Strategies

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!
\( \sim \) 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
\( \sim \) semi-naive evaluation

What’s Wrong with Naive Evaluation?

An example Datalog program:

```
\begin{align*}
\text{e}(1, 2) & \text{ e}(2, 3) \text{ e}(3, 4) \text{ e}(4, 5) \\
(R1) & T(x, y) \leftarrow e(x, y) \\
(R2) & T(x, z) \leftarrow T(x, y) \land T(y, z)
\end{align*}
```

How many body matches do we need to iterate over?

\[
\begin{align*}
T_p^0 &= 0 & \text{initialisation} \\
T_p^1 &= [T(1, 2), T(2, 3), T(3, 4), T(4, 5)] & 4 \text{ matches for (R1)} \\
T_p^2 &= T_p^1 \cup [T(1, 3), T(2, 4), T(3, 5)] & 4 \times (R1) + 3 \times (R2) \\
T_p^3 &= T_p^2 \cup [T(1, 4), T(2, 5), T(1, 5)] & 4 \times (R1) + 8 \times (R2) \\
T_p^4 &= T_p^3 = T_p^\infty & 4 \times (R1) + 10 \times (R2)
\end{align*}
\]

In total, we considered 37 matches to derive 11 facts

Semi-Naive Evaluation

The computation yields sets $T_p^0 \subseteq T_p^1 \subseteq T_p^2 \subseteq \ldots \subseteq T_p^\infty$
- For an IDB predicate \( R \), let \( R' \) be the “predicate” that contains exactly the R-facts in \( T_p^i \)
- For \( i \leq 1 \), let \( \Delta^R_{i} \) be the collection of facts \( R' \setminus R'^{-1} \)

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'(x, y) \land T'(y, z) & \text{same as original rule} \\
T(x, z) & \leftarrow \Delta^R_{i}(x, y) \land \Delta^R_{i}(y, z) & \text{restrict to new facts} \\
T(x, z) & \leftarrow \Delta^R_{i}(x, y) \land T'(y, z) & \text{partially restrict to new facts} \\
T(x, z) & \leftarrow T'(x, y) \land \Delta^R_{i}(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:

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

\[
\begin{align*}
\Delta^1_1 & = (\mathbf{T}(1, 2), \mathbf{T}(2, 3), \mathbf{T}(3, 4), \mathbf{T}(4, 5)) \\
\Delta^2_1 & = (\mathbf{T}(1, 3), \mathbf{T}(2, 4), \mathbf{T}(3, 5)) \\
\Delta^3_1 & = (\mathbf{T}(1, 4), \mathbf{T}(2, 5), \mathbf{T}(1, 5)) \\
\Delta^4_1 & = \emptyset
\end{align*}
\]

\[
\begin{align*}
\mathbf{T}_p^0 & = \emptyset \\
\Delta^1_2 & = \Delta^1_1 \\
\Delta^2_2 & = \Delta^2_1 \\
\Delta^3_2 & = \Delta^3_1 \\
\Delta^4_2 & = \emptyset
\end{align*}
\]

To derive \(\mathbf{T}(1, 4)\) in \(\Delta^4_2\), we need to combine \(\mathbf{T}(3, 4) \in \Delta^3_1\) with \(\mathbf{T}(1, 2) \in \Delta^1_1\) or \(\mathbf{T}(1, 2) \in \Delta^1_1\) with \(\mathbf{T}(2, 4) \in \Delta^2_1\)

\[
\sim \quad \text{rule } \mathbf{T}(x, z) \leftarrow \Delta^1_1(x, y) \land \Delta^2_1(y, z)
\]

\[
\sim \quad \text{replace } (R2.2) \text{ by the following rule:}
\]

\[
(R2.2') \quad \mathbf{T}(x, z) \leftarrow \mathbf{T}^{-1}(x, y) \land \Delta^1_1(y, z)
\]

Semi-Naive Evaluation: Example

\[
\begin{align*}
\mathbf{e}(1, 2) & \quad \mathbf{e}(2, 3) \quad \mathbf{e}(3, 4) \quad \mathbf{e}(4, 5) \\
(R1) & \quad \mathbf{T}(x, y) \leftarrow \mathbf{e}(x, y) \\
(R2.1) & \quad \mathbf{T}(x, z) \leftarrow \Delta^1_1(x, y) \land \mathbf{T}^{-1}(y, z) \\
(R2.2') & \quad \mathbf{T}(x, z) \leftarrow \mathbf{T}^{-1}(x, y) \land \Delta^1_1(y, z)
\end{align*}
\]

How many body matches do we need to iterate over?

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

In total, we considered 14 matches to derive 11 facts.

Semi-Naive Evaluation: Full Definition

In general, a rule of the form

\[
\mathbf{H}(\mathbf{z}) \leftarrow \mathbf{e}_1(\mathbf{y}_1) \land \ldots \land \mathbf{e}_n(\mathbf{y}_n) \land \mathbf{I}_1(\mathbf{z}_1) \land \mathbf{I}_2(\mathbf{z}_2) \land \ldots \land \mathbf{I}_m(\mathbf{z}_m)
\]

is transformed into \(m\) rules

\[
\begin{align*}
\mathbf{H}^0(\mathbf{z}) & \leftarrow \mathbf{e}_1(\mathbf{y}_1) \land \ldots \land \mathbf{e}_n(\mathbf{y}_n) \land \mathbf{I}_1(\mathbf{z}_1) \\
\mathbf{H}^1(\mathbf{z}) & \leftarrow \mathbf{e}_1(\mathbf{y}_1) \land \ldots \land \mathbf{e}_n(\mathbf{y}_n) \land \mathbf{I}_1^{-1}(\mathbf{z}_1) \land \mathbf{I}_2(\mathbf{z}_2) \\
& \quad \ldots \\
\mathbf{H}^m(\mathbf{z}) & \leftarrow \mathbf{e}_1(\mathbf{y}_1) \land \ldots \land \mathbf{e}_n(\mathbf{y}_n) \land \mathbf{I}_1^{-1}(\mathbf{z}_1) \land \mathbf{I}_2^{-1}(\mathbf{z}_2) \land \ldots \land \mathbf{I}_m(\mathbf{z}_m)
\end{align*}
\]

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:
- How can we implement Datalog in practice?