FOUNDATIONS OF DATABASES AND QUERY LANGUAGES

Lecture 11: Optimisation an Evaluation of Datalog

Markus Krötzsch

TU Dresden, 29 June 2015

Overview

See course homepage [⇒ link] for more information and materials

Review: Datalog Expressivity and Complexity

A rule-based recursive query language

$$
\begin{aligned}
&\text{father}(\text{alice}, \text{bob})\\
&\text{mother}(\text{alice}, \text{carla})\\
\text{Parent}(x, y) &\leftarrow \text{father}(x, y)\\
\text{Parent}(x, y) &\leftarrow \text{mother}(x, y)\\
&\text{SameGeneration}(x, x)\\
\text{SameGeneration}(x, y) &\leftarrow \text{Parent}(x, v) \wedge \text{Parent}(y, w) \wedge \text{SameGeneration}(v, w)
\end{aligned}
$$

Datalog is more complex than FO query answering:

- ExpTime-complete for query and combined complexity
- P-complete for data complexity

Datalog cannot express all query mappings in P
but semipositive Datalog with a successor ordering can

Datalog Implementation and Optimisation

How can Datalog query answering be implemented?
How can Datalog queries be optimised?

Recall: static query optimisation

- Query equivalence
- Query emptiness
- Query containment

⇝ all undecidable for FO queries, but decidable for (U)CQs

Learning from CQ Containment?

How did we manage to decide the question $Q_1 \overset{?}{\sqsubseteq} Q_2$ for conjunctive queries Q_1 and Q_2?

Key ideas were:

- We want to know if all situations where Q_1 matches are also matched by Q_2.
- We can simply view Q_1 as a database $\mathcal{I}_{Q_1}$: the most general database that Q_1 can match to
- Containment $Q_1 \overset{?}{\sqsubseteq} Q_2$ holds if Q_2 matches the database $\mathcal{I}_{Q_1}$.

$\leadsto$ decidable in NP

A CQ $Q[x_1, \ldots, x_n]$ can be expressed as a Datalog query with a single rule $\mathsf{Ans}(x_1, \ldots, x_n) \leftarrow Q$
$\leadsto$ Could we apply a similar technique to Datalog?

Checking Rule Entailment

The containment decision procedure for CQs suggests a procedure for single Datalog rules:

- Consider a Datalog program P and a rule $H \leftarrow B_1 \wedge \ldots \wedge B_n$.
- Define a database $\mathcal{I}_{B_1 \wedge \ldots \wedge B_n}$ as for CQs:
  - For every variable x in $H \leftarrow B_1 \wedge \ldots \wedge B_n$, we introduce a fresh constant c_x, not used anywhere yet
  - We define H^c to be the same as H but with each variable x replaced by c_x; similarly we define B_i^c for each $1 \leq i \leq n$
  - The database $\mathcal{I}_{B_1 \wedge \ldots \wedge B_n}$ contains exactly the facts B_i^c $(1 \leq i \leq n)$
- Now check if $H^c \in T_P^\infty(\mathcal{I}_{B_1 \wedge \ldots \wedge B_n})$:
  - If no, then there is a database on which $H \leftarrow B_1 \wedge \ldots \wedge B_n$ produces an entailment that P does not produce.
  - If yes, then $P \models H \leftarrow B_1 \wedge \ldots \wedge B_n$

Example: Rule Entailment

Let P be the program

$$\mathsf{Ancestor}(x, y) \leftarrow \mathsf{parent}(x, y)$$
$$\mathsf{Ancestor}(x, z) \leftarrow \mathsf{parent}(x, y) \wedge \mathsf{Ancestor}(y, z)$$

and consider the rule $\mathsf{Ancestor}(x, z) \leftarrow \mathsf{parent}(x, y) \wedge \mathsf{parent}(y, z)$.

Then $\mathcal{I}_{\mathsf{parent}(x,y) \wedge \mathsf{parent}(y,z)} = \{\mathsf{parent}(c_x, c_y), \mathsf{parent}(c_y, c_z)\}$ (abbr. as $\mathcal{I}$).
We can compute $T_P^\infty(\mathcal{I})$:

$$\begin{aligned}
T_P^0(\mathcal{I}) &= \mathcal{I} \\
T_P^1(\mathcal{I}) &= \{\mathsf{Ancestor}(c_x, c_y), \mathsf{Ancestor}(c_y, c_z)\} \cup \mathcal{I} \\
T_P^2(\mathcal{I}) &= \{\mathsf{Ancestor}(c_x, c_z) \cup T_P^1(\mathcal{I}) \\
T_P^3(\mathcal{I}) &= T_P^2(\mathcal{I}) = T_P^\infty(\mathcal{I})
\end{aligned}$$

Therefore, $\mathsf{Ancestor}(x, z)^c = \mathsf{Ancestor}(c_x, c_z) \in T_P^\infty(\mathcal{I})$,
so P entails $\mathsf{Ancestor}(x, z) \leftarrow \mathsf{parent}(x, y) \wedge \mathsf{parent}(y, z)$.

Deciding Datalog Containment?

Idea for two Datalog programs P_1 and P_2:

- If $P_2 \models P_1$, then every entailment of P_1 is also entailed by P_2
- In particular, this means that P_1 is contained in P_2
- We have $P_2 \models P_1$ if $P_2 \models H \leftarrow B_1 \wedge \ldots \wedge B_n$ for every rule $H \leftarrow B_1 \wedge \ldots \wedge B_n \in P_1$
- We can decide $P_2 \models H \leftarrow B_1 \wedge \ldots \wedge B_n$.

Can we decide Datalog containment this way?
$\leadsto$ No! In fact, Datalog containment is undecidable. What's wrong?

Implication Entailment vs. Datalog Entailment

P_1 :

$\mathsf{A}(x, y) \leftarrow \mathsf{parent}(x, y)$

$\mathsf{A}(x, z) \leftarrow \mathsf{parent}(x, y) \wedge \mathsf{A}(y, z)$

P_2 :

$\mathsf{B}(x, y) \leftarrow \mathsf{parent}(x, y)$

$\mathsf{B}(x, z) \leftarrow \mathsf{parent}(x, y) \wedge \mathsf{B}(y, z)$

Consider the Datalog queries $\langle A, P_1\rangle$ and $\langle B, P_2\rangle$:

- Clearly, $\langle A, P_1\rangle$ and $\langle B, P_2\rangle$ are equivalent (and mutually contained in each other).
- However, P_2 entails no rule of P_1 and P_1 entails no rule of P_2.

$\rightsquigarrow$ IDB predicates do not matter in Datalog, but predicate names matter in first-order implications

Datalog as Second-Order Logic

Datalog is a fragment of second-order logic:
IDB pred's are like variables that can take any set of tuples as value!

Example: the query $\langle A, P_1\rangle$ can be expressed by the formula

$$\forall \mathsf{A}. \left(\begin{array}{lll} \forall x, y.\mathsf{A}(x, y) & \leftarrow \mathsf{parent}(x, y) & \wedge \\ \forall x, y, z.\mathsf{A}(x, z) & \leftarrow \mathsf{parent}(x, y) \wedge \mathsf{A}(y, z) & \end{array} \right) \rightarrow \mathsf{A}(v, w)$$

- This is a formula with two free variables v and w.
  $\rightsquigarrow$ query with two result variables
- Intuitive semantics: "$\langle c, d\rangle$ is a query result if $\mathsf{A}(c, d)$ holds for all possible values of A that satisfy the rules"
  $\rightsquigarrow$ Datalog semantics in other words

We can express any Datalog query like this, with one second-order variable per IDB predicate.

First-Order vs. Second-Order Logic

A Datalog program looks like a set of first-order implications, but it has a second-order semantics

We have already seen that Datalog can express things that are impossible to express in FO queries – that's why we introduced it![1]

Consequences for query optimisation:

- Entailment between sets of first-order implications is decidable (shown above)
- Containment between Datalog queries is not decidable (shown next)

[1] Possible confusion when comparing of FO and Datalog: entailments of first-order implications agree with answers of Datalog queries, so it seems we can break the FO locality restrictions; but query answering is model checking not entailment; FO model checking is much weaker than second-order model checking

Undecidability of Datalog Query Containment

A classical undecidable problem: Post Correspondence Problem

- Input: two lists of words $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$
- Output: "yes" if there is a sequence of indices $i_1, i_2, i_3, \ldots, i_m$ such that $\alpha_{i_1}\alpha_{i_2}\alpha_{i_3}\cdots\alpha_{i_m} = \beta_{i_1}\beta_{i_2}\beta_{i_3}\cdots\beta_{i_m}$.

$\rightsquigarrow$ we will reduce PCP to Datalog containment

We need to define Datalog programs that work on databases that encode words:

- We represent words by chains of binary predicates
- Binary EDB predicates represent a letters
- For each letter σ, we use a binary EDB predicate $\mathsf{letter}[\sigma]$
- We assume that the words α_i have the form $a^i_1 \cdots a^i_{|\beta_i|}$, and that the words β_i have the form $b^i_1 \cdots b^i_{|\beta_i|}$

Solving PCP with Datalog Containment

A program P_1 to recognise potential PCP solutions.

Rules to recognise words α_i and β_i for every $i \in \{1, \ldots, m\}$:

$$\mathsf{A}_i(x_0, x_{|\alpha_i|}) \leftarrow \mathsf{letter}[a^i_1](x_0, x_1) \wedge \ldots \wedge \mathsf{letter}[a^i_{|\alpha_i|}](x_{|\alpha_i|-1}, x_{|\alpha_i|})$$
$$\mathsf{B}_i(x_0, x_{|\beta_i|}) \leftarrow \mathsf{letter}[b^i_1](x_0, x_1) \wedge \ldots \wedge \mathsf{letter}[b^i_{|\beta_i|}](x_{|\beta_i|-1}, x_{|\beta_i|})$$

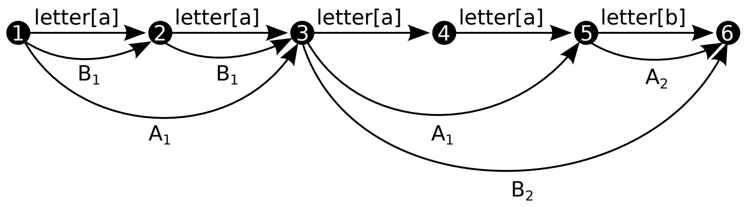
Rules to check for synchronised chairs (for all $i \in \{1, \ldots, m\}$):

$$\mathsf{PCP}(x, y_1, y_2) \leftarrow A_i(x, y_1) \wedge B_i(x, y_2)$$
$$\mathsf{PCP}(x, z_1, z_2) \leftarrow \mathsf{PCP}(x, y_1, y_2) \wedge A_i(y_1, z_1) \wedge B_i(y_2, z_2)$$
$$\mathsf{Accept}() \leftarrow \mathsf{PCP}(x, z, z)$$

Solving PCP with Datalog Containment (2)

Example: $\alpha_1 = aa, \beta_1 = a, \alpha_2 = b, \beta_2 = aab$

Example for an indented database and least model (selected parts):



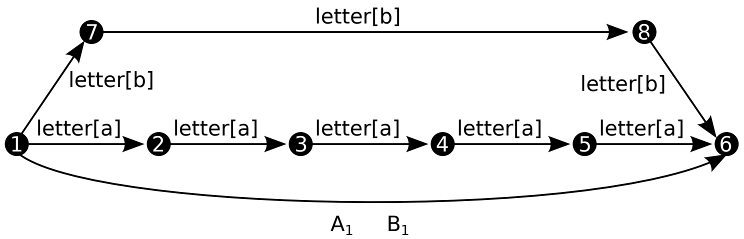
Additional IDB facts that are derived (among others):

$\mathsf{PCP}(1, 3, 2)$ $\mathsf{PCP}(1, 5, 3)$ $\mathsf{PCP}(1, 6, 6)$ $\mathsf{Accept}()$

Solving PCP with Datalog Containment (3)

Example: $\alpha_1 = aaaaa, \beta_1 = bbb$

Problem: P_1 also accepts some unintended cases



Additional IDB facts that are derived:

$\mathsf{PCP}(1, 6, 6)$ $\mathsf{Accept}()$

Solving PCP with Datalog Containment (4)

Solution: specify a program P_2 that recognises all unwanted cases

P_2 consists of the following rules (for all letters σ, σ'):

$$\mathsf{EP}(x, x) \leftarrow$$
$$\mathsf{EP}(y_1, y_2) \leftarrow \mathsf{EP}(x_1, x_2) \wedge \mathsf{letter}[\sigma](x_1, y_1) \wedge \mathsf{letter}[\sigma](x_2, y_2)$$
$$\mathsf{Accept}() \leftarrow \mathsf{EP}(x_1, x_2) \wedge \mathsf{letter}[\sigma](x_1, y_1) \wedge \mathsf{letter}[\sigma'](x_2, y_2) \qquad \sigma \neq \sigma'$$
$$\mathsf{NEP}(x_1, y_2) \leftarrow \mathsf{EP}(x_1, x_2) \wedge \mathsf{letter}[\sigma](x_2, y_2)$$
$$\mathsf{NEP}(x_1, y_2) \leftarrow \mathsf{NEP}(x_1, x_2) \wedge \mathsf{letter}[\sigma](x_2, y_2)$$
$$\mathsf{Accept}() \leftarrow \mathsf{NEP}(x, x)$$

Intuition:

- EP defines equal paths (forwards, from one starting point)
- NEP defines paths of different length (from one starting point to the same end point)

$\leadsto$ P_2 accepts all databases with distinct parallel paths

Solving PCP with Datalog Containment (5)

What does it mean if $\langle \text{Accept}, P_1 \rangle$ is contained in $\langle \text{Accept}, P_2 \rangle$?

The following are equivalent:

- All databases with potential PCP solutions also have distinct parallel paths.
- Databases without distinct parallel paths have no PCP solutions.
- Linear databases (words) have no PCP solutions.
- The answer to the PCP is "no".

$\leadsto$ If we could decide Datalog containment, we could decide PCP

Theorem

Containment and equivalence of Datalog queries are undecidable.

(Note that emptiness of Datalog queries is trivial)

Implementation of Datalog

Implementing Datalog

FO queries (and thus also CQs and UCQs) are supported by almost all DMBS
$\leadsto$ many specific implementation and optimisation techniques

How can Datalog queries be answered in practice?
$\leadsto$ 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^∞

```
01  T_P^0 := ∅
02  i := 0
03  repeat :
04      T_P^{i+1} := ∅
05      for H ← B_1 ∧ ... ∧ B_ℓ ∈ P :
06          for θ ∈ B_1 ∧ ... ∧ B_ℓ(T_P^i) :
07              T_P^{i+1} := T_P^{i+1} ∪ {Hθ}
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 θ 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 $\mathcal{I}$, we write $\theta \in Q(\mathcal{I})$ if $\mathcal{I} \models Q\theta$

What's Wrong with Naive Evaluation?

An example Datalog program:

$$\mathsf{e}(1,2) \quad \mathsf{e}(2,3) \quad \mathsf{e}(3,4) \quad \mathsf{e}(4,5)$$
$$(R1) \quad \mathsf{T}(x,y) \leftarrow \mathsf{e}(x,y)$$
$$(R2) \quad \mathsf{T}(x,z) \leftarrow \mathsf{T}(x,y) \wedge \mathsf{T}(y,z)$$

How many body matches do we need to iterate over?

$T_P^0 = \emptyset$	initialisation
$T_P^1 = \{\mathsf{T}(1,2), \mathsf{T}(2,3), \mathsf{T}(3,4), \mathsf{T}(4,5)\}$	4 matches for $(R1)$
$T_P^2 = T_P^1 \cup \{\mathsf{T}(1,3), \mathsf{T}(2,4), \mathsf{T}(3,5)\}$	$4 \times (R1) + 3 \times (R2)$
$T_P^3 = T_P^2 \cup \{\mathsf{T}(1,4), \mathsf{T}(2,5), \mathsf{T}(1,5)\}$	$4 \times (R1) + 8 \times (R2)$
$T_P^4 = T_P^3 = T_P^\infty$	$4 \times (R1) + 10 \times (R2)$

In total, we considered 37 matches to derive 11 facts

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!
$\leadsto$ 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 $T_P^0 \subseteq T_P^1 \subseteq T_P^2 \subseteq \ldots \subseteq T_P^\infty$

- For an IDB predicate R, let R^i be the "predicate" that contains exactly the R-facts in T_P^i
- For $i \leq 1$, let Δ_{R}^i be the collection of facts $\mathsf{R}^i \setminus \mathsf{R}^{i-1}$

We can restrict rules to use only some computations.
Some options for the computation in step $i+1$:

$\mathsf{T}(x,z) \leftarrow \mathsf{T}^i(x,y) \wedge \mathsf{T}^i(y,z)$	same as original rule
$\mathsf{T}(x,z) \leftarrow \Delta_{\mathsf{T}}^i(x,y) \wedge \Delta_{\mathsf{T}}^i(y,z)$	restrict to new facts
$\mathsf{T}(x,z) \leftarrow \Delta_{\mathsf{T}}^i(x,y) \wedge \mathsf{T}^i(y,z)$	partially restrict to new facts
$\mathsf{T}(x,z) \leftarrow \mathsf{T}^i(x,y) \wedge \Delta_{\mathsf{T}}^i(y,z)$	partially restrict to new facts

What to chose?

Semi-Naive Evaluation (2)

Inferences that involve new and old facts are necessary:

$$\mathsf{e}(1,2) \quad \mathsf{e}(2,3) \quad \mathsf{e}(3,4) \quad \mathsf{e}(4,5)$$

$$(R1) \qquad \mathsf{T}(x,y) \leftarrow \mathsf{e}(x,y)$$

$$(R2) \qquad \mathsf{T}(x,z) \leftarrow \mathsf{T}(x,y) \wedge \mathsf{T}(y,z)$$

$$\begin{array}{ll}
 & T_P^0 = \emptyset \\
\Delta_\mathsf{T}^1 = \{\mathsf{T}(1,2), \mathsf{T}(2,3), \mathsf{T}(3,4), \mathsf{T}(3,4), \mathsf{T}(4,5)\} & T_P^1 = \Delta_\mathsf{T}^1 \\
\Delta_\mathsf{T}^2 = \{\mathsf{T}(1,3), \mathsf{T}(2,4), \mathsf{T}(3,5)\} & T_P^2 = T_P^1 \cup \Delta_\mathsf{T}^2 \\
\Delta_\mathsf{T}^3 = \{\mathsf{T}(1,4), \mathsf{T}(2,5), \mathsf{T}(1,5)\} & T_P^3 = T_P^2 \cup \Delta_\mathsf{T}^3 \\
\Delta_\mathsf{T}^4 = \emptyset & T_P^4 = T_P^3 = T_P^\infty
\end{array}$$

To derive $\mathsf{T}(1,4)$ in Δ_T^3, we need to combine
$\mathsf{T}(1,3) \in \Delta_\mathsf{T}^2$ with $\mathsf{T}(3,4) \in \Delta_\mathsf{T}^1$ or $\mathsf{T}(1,2) \in \Delta_\mathsf{T}^1$ with $\mathsf{T}(2,4) \in \Delta_\mathsf{T}^2$
$\leadsto$ rule $\mathsf{T}(x,z) \leftarrow \Delta_\mathsf{T}^i(x,y) \wedge \Delta_\mathsf{T}^i(y,z)$ 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:

$$\mathsf{e}(1,2) \quad \mathsf{e}(2,3) \quad \mathsf{e}(3,4) \quad \mathsf{e}(4,5)$$

$$(R1) \qquad \mathsf{T}(x,y) \leftarrow \mathsf{e}(x,y)$$

$$(R2.1) \qquad \mathsf{T}(x,z) \leftarrow \Delta_\mathsf{T}^i(x,y) \wedge \mathsf{T}^i(y,z)$$

$$(R2.2) \qquad \mathsf{T}(x,z) \leftarrow \mathsf{T}^i(x,y) \wedge \Delta_\mathsf{T}^i(y,z)$$

There is still redundancy here: the matches for
$\mathsf{T}(x,z) \leftarrow \Delta_\mathsf{T}^i(x,y) \wedge \Delta_\mathsf{T}^i(y,z)$ are covered by both $(R2.1)$ and $(R2.2)$
$\leadsto$ replace $(R2.2)$ by the following rule:

$$(R2.2') \qquad \mathsf{T}(x,z) \leftarrow \mathsf{T}^{i-1}(x,y) \wedge \Delta_\mathsf{T}^i(y,z)$$

EDB atoms do not change, so their Δ would be $\emptyset$
$\leadsto$ ignore such rules after the first iteration

Semi-Naive Evaluation: Example

$$\mathsf{e}(1,2) \quad \mathsf{e}(2,3) \quad \mathsf{e}(3,4) \quad \mathsf{e}(4,5)$$

$$(R1) \qquad \mathsf{T}(x,y) \leftarrow \mathsf{e}(x,y)$$

$$(R2.1) \qquad \mathsf{T}(x,z) \leftarrow \Delta_\mathsf{T}^i(x,y) \wedge \mathsf{T}^i(y,z)$$

$$(R2.2') \qquad \mathsf{T}(x,z) \leftarrow \mathsf{T}^{i-1}(x,y) \wedge \Delta_\mathsf{T}^i(y,z)$$

How many body matches do we need to iterate over?

$T_P^0 = \emptyset$	initialisation
$T_P^1 = \{\mathsf{T}(1,2), \mathsf{T}(2,3), \mathsf{T}(3,4), \mathsf{T}(4,5)\}$	$4 \times (R1)$
$T_P^2 = T_P^1 \cup \{\mathsf{T}(1,3), \mathsf{T}(2,4), \mathsf{T}(3,5)\}$	$3 \times (R2.1)$
$T_P^3 = T_P^2 \cup \{\mathsf{T}(1,4), \mathsf{T}(2,5), \mathsf{T}(1,5)\}$	$3 \times (R2.1), 2 \times (R2.2')$
$T_P^4 = T_P^3 = T_P^\infty$	$1 \times (R2.1), 1 \times (R2.2')$

In total, we considered 14 matches to derive 11 facts

Semi-Naive Evaluation: Full Definition

In general, a rule of the form

$$\mathsf{H}(\vec{x}) \leftarrow \mathsf{e}_1(\vec{y}_1) \wedge \ldots \wedge \mathsf{e}_n(\vec{y}_n) \wedge \mathsf{I}_1(\vec{z}_1) \wedge \mathsf{I}_2(\vec{z}_2) \wedge \ldots \wedge \mathsf{I}_m(\vec{z}_m)$$

is transformed into m rules

$$\mathsf{H}(\vec{x}) \leftarrow \mathsf{e}_1(\vec{y}_1) \wedge \ldots \wedge \mathsf{e}_n(\vec{y}_n) \wedge \Delta_{\mathsf{I}_1}^i(\vec{z}_1) \wedge \mathsf{I}_2^i(\vec{z}_2) \wedge \ldots \wedge \mathsf{I}_m^i(\vec{z}_m)$$

$$\mathsf{H}(\vec{x}) \leftarrow \mathsf{e}_1(\vec{y}_1) \wedge \ldots \wedge \mathsf{e}_n(\vec{y}_n) \wedge \mathsf{I}_1^{i-1}(\vec{z}_1) \wedge \Delta_{\mathsf{I}_2}^i(\vec{z}_2) \wedge \ldots \wedge \mathsf{I}_m^i(\vec{z}_m)$$

$$\ldots$$

$$\mathsf{H}(\vec{x}) \leftarrow \mathsf{e}_1(\vec{y}_1) \wedge \ldots \wedge \mathsf{e}_n(\vec{y}_n) \wedge \mathsf{I}_1^{i-1}(\vec{z}_1) \wedge \mathsf{I}_2^{i-1}(\vec{z}_2) \wedge \ldots \wedge \Delta_{\mathsf{I}_m}^i(\vec{z}_m)$$

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

Perfect Datalog optimisation is impossible

- same situation as for FO queries
- but for somewhat different reasons

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

Simplest practical bottom-up technique: semi-naive evaluation

Next topics:

- More on Datalog implementation
- Further query languages
- Applications