2. Introduction to Datalog

Reinhard Pichler
Institute of Logic and Computation
DBAI Group
TU Wien
13 March, 2018

Motivation

- SQL, relational algebra, relational calculus (both tuple and domain relational calculus) are "relational complete", i.e., they have the full expressive power of relational algebra.
- But: many interesting queries cannot be formulated in these languages
- Example: no recursive queries (SQL now offers a recursive construct)

Example

- Relation parent(PARENT, CHILD), gives information on the parent-child relationship of a certain group of people.
- Problem: look for all ancestors of a certain person.
- Solution: define relation ANCESTOR(X, Y): X is ancestor of Y by generating one generation after the other (one join and one projection each) and finally merge all generations (union):

  \[
  \text{grandparent}(\text{GRANDPARENT}, \text{GRANDCHILD}) := \pi_{1,4}(\text{parent}[\text{CHILD} = \text{PARENT}] \text{parent}) \\
  \text{grandgrandparent}(\text{GRANDGRANDPARENT}, \text{GRANDGRANDCHILD}) := \pi_{1,4}(\text{parent}[\text{CHILD} = \text{GRANDPARENT}] \text{grandparent}) \\
  \ldots
  \]

  \[
  \text{ancestor}(\text{ANCESTOR}, \text{NAME}) := \text{parent} \cup \text{grandparent} \cup \text{grandgrandparent} \cup \ldots
  \]
Possible Solution

- Use of a programming language with an embedded relational complete query language:

```haskell
begin
  result := {};
  newtuples := parent;
  while newtuples ⊈ result do
    begin
      result := result ∪ newtuples;
      newtuples := π₁,₄(newtuples[2 = 1]parent);
    end;
  ancestor := result
end.
```

- Procedural, needs knowledge of a programming language, leaves little room for query optimization.

Better Solution: Datalog

- Prolog-like logical query language,
- allows recursive queries in a declarative way

Example:
- compute all ancestors on the basis of the relation parent
  ```prolog
  ancestor(X,Y) :- parent(X,Y).
  ancestor(X,Z) :- parent(X,Y), ancestor(Y,Z).
  ```
- use the ancestor predicate to compute the ancestors of a certain person (Hans):
  ```prolog
  hans_ancestor(X) :- ancestor(X,hans).
  ```
- compute the ancestors of a certain person (Hans) directly:
  ```prolog
  hans_ancestor(X) :- parent(X,hans).
  hans_ancestor(X) :- hans_ancestor(Y), parent(X,Y).
  ```

Datalog - Syntax

```
<datalog_program> ::= <datalog_rule> | <datalog_program><datalog_rule>
<datalog_rule> ::= <head> :- <body>
<head> ::= <literal> | <body>, <literal>
<body> ::= <relation_id>(<list_of_args>) | <term> | <list_of_args>, <term>
<relation_id> ::= <symb_const> | <symb_var>
<symb_const> ::= <number> | <lcc> | <lcc><string>
<symb_var> ::= <ucc> | <ucc><string>
(list_of_args) ::= <term> | <list_of_args>, <term>
```

(lcc = lower_case_character; ucc = upper_case_character)

Restrictions on the Datalog Syntax

- name of an existing relation of the database (parent) - can be used only in rule bodies
- name of a new relation defined by the datalog program (ancestor)
- has always the same number of arguments.

comparison predicates:
- =, <>, <, > are treated like known database relations.

variables:
- each variable that appears in the head of a rule has to be bound in the body
- variables that appear as arguments of comparison predicates must appear in the same body in literals without comparison predicates

A datalog query is also called datalog program
Logical Semantics of Datalog

We consider

- *R.* datalog rule of the form \( L_0 \leftarrow L_1, L_2, \ldots, L_n. \)
- \( L_i \) literal of the form \( p_i(t_1, \ldots, t_n) \)
- \( x_1, x_2, \ldots, x_r \) variables in \( R \)
- *P.* datalog program with the rules \( R_1, R_2, \ldots, R_m \)

We construct

\[
R^* = \forall x_1 \forall x_2 \ldots \forall x_r ((L_1 \land L_2 \land \cdots \land L_n) \Rightarrow L_0).
\]

We assign to each datalog program \( P \) the (semantically) well-defined formula \( P^* \) as follows:

\[
P^* = R_1^* \land R_2^* \land \cdots \land R_m^*
\]

We have:

- \( DB^* \) is a conjunction of ground atoms (i.e., the facts)
- \( P^* \) is a conjunction of formulas with implication

Let \( G \) be a conjunction of facts and formulas with implication. Then the set \( \text{cons}(G) \) of facts that follow from \( G \) is uniquely defined.

In other words, we have \( \text{cons}(G) = \{ A \mid A \text{ is a fact with } G \models A \} \).

**Definition**

The semantics of a datalog program \( P \) is defined as the function \( M[P] \), that assigns to each database \( DB \) the set of all facts that follow from the formula \( "P^* \land DB^*" \):

\[
M[P] : DB \to \text{cons}(P^* \land DB^*)
\]

We consider now

- *REL.* a relation of the database.
- \( \langle t_1, \ldots, t_n \rangle \) a tuple of the relation \( REL \).
- \( rel(t_1, \ldots, t_n) \) a fact
- *DB.* database with relations \( REL_1, REL_2, \ldots, REL_k \)

We assign to each database relation \( REL \) the formula

\[
REL^* = \text{conjunction of all facts}
\]

- a relation is an unordered set of tuples
- the assignment \( REL \mapsto REL^* \) is therefore not uniquely defined.
- take an arbitrary order (e.g. lexicographical order) since conjunction is associative and commutative.

We assign to each database \( DB \) the (semantically) well-defined formula \( DB^* \) as follows:

\[
DB^* = REL_1^* \land REL_2^* \land \cdots \land REL_k^*.
\]

**Example**

Consider the database \( DB \) with relations \( \text{woman(NAME)}, \text{man(NAME)}, \text{parent(PARENT, CHILD)} \) and the datalog program:

\[
\text{grandpa}(X,Y) :- \text{man}(X), \text{parent}(X,Z), \text{parent}(Z,Y).
\]

<table>
<thead>
<tr>
<th>woman (NAME)</th>
<th>man (NAME)</th>
<th>parent (PARENT, CHILD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grete</td>
<td>Hans</td>
<td>Hans, Linda</td>
</tr>
<tr>
<td>Linda</td>
<td>Karl</td>
<td>Grete, Linda</td>
</tr>
<tr>
<td>Gerti</td>
<td>Michael</td>
<td>Karl, Michael</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Linda, Michael, Karl, Gerti</td>
</tr>
</tbody>
</table>
Let us compute $DB^*$, $P^*$ and $\text{cons}(P^* \land DB^*)$:

$$DB^* = REL_1^* \land \cdots \land REL_k^*$$ with $REL_i^* = \text{conjunction of all facts}

$$DB^* = \text{woman(grete)} \land \text{woman(linda)} \land \text{woman(gerti)} \land \text{man(hans)} \land \text{man(karl)} \land \text{man(michael)} \land \text{parent(hans, linda)} \land \text{parent(grete, linda)} \land \text{parent(karl, michael)} \land \text{parent(linda, michael)} \land \text{parent(karl, gerti)} \land \text{parent(linda, gerti)}.$$

$$P^* = R_1^* \land \cdots \land R_m^*$$ with $R_i^* = \forall X_1 \forall X_2 \ldots \forall X_\ell ((L_1 \land \cdots \land L_n) \Rightarrow L_0)$.

$$P^* = \forall X \forall Y \forall Z : ((\text{man}(X) \land \text{parent}(X, Z) \land \text{parent}(Z, Y)) \Rightarrow \text{grandpa}(X, Y)).$$

The new facts in $\text{cons}(P^* \land DB^*)$:

$$\text{grandpa}(\text{hans}, \text{michael}), \text{grandpa}(\text{hans}, \text{gerti}).$$

The datalog program $P$ with

$P = \text{grandpa}(X, Y) :- \text{man}(X), \text{parent}(X, Z), \text{parent}(Z, Y)$

defines a new relation $\text{grandpa}$ with the following tuples:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hans</td>
<td>Michael</td>
</tr>
<tr>
<td>Hans</td>
<td>Gerti</td>
</tr>
</tbody>
</table>

Operational Semantics of Datalog

- Datalog rules are seen as inference rules,
- a fact that appears in the head of a rule can be deduced, if the facts in the body of the rule can be deduced.

Example:

facts: parent(linda,michael), parent(linda,gerti)
rule: siblings(michael,gerti) :- parent(linda,michael), parent(linda,gerti).

the following fact can be deduced:

siblings(michael,gerti)

Rules with variables

- A rule $R$ with variables represents all variable-free rules we get from $R$ by substituting the variables with the constant symbols.
- The constant symbols are taken from the database $DB$ and the program $P$.
- A variable-free rule resulting from such a substitution is called ground instance of $R$ with respect to $P$ and $DB$.
- We write $\text{Ground}(R, P, DB)$ to denote the set of all ground instances over $P$ and $DB$ of $R$. 
Example:

Compute all relations between siblings with the following rule:

\[ \text{siblings}(Y, Z) : = \text{parent}(X, Y), \text{parent}(X, Z), Y < > Z. \]

All 6 ground instances of this rule with respect to \( P \) and \( DB \) from above are (Note that there are 6 constant symbols: \{Grete, Linda, Gerti, Hans, Michael, Karl\}):

\[
\begin{align*}
\text{siblings(grete, grete)} & : = \text{parent(grete, grete)}, \text{parent(grete, grete)}, \\
& \quad \text{grete} < > \text{grete} \quad (X = Y = Z = \text{grete}) \\
\text{siblings(grete, linda)} & : = \text{parent(grete, grete)}, \text{parent(grete, linda)}, \\
& \quad \text{grete} < > \text{linda} \quad (X = Y = \text{grete}, Z = \text{linda}) \\
\text{siblings(karl, karl)} & : = \text{parent(karl, karl)}, \text{parent(karl, karl)}, \\
& \quad \text{karl} < > \text{karl} \quad (X = Y = Z = \text{karl})
\end{align*}
\]

Idea: execution of a datalog program \( P \) on a database \( DB \): iterative deduction of facts until saturation is reached (fixpoint)

Formalization: define a fixpoint operator

- define Operator \( T_P(DB) \): augments \( DB \) with all facts, that can be deduced in one step by applying the rules from \( P \) to \( DB \).

\[
T_P(DB) = DB \cup \bigcup_{R \in P} \{ L_0 \mid L_0 : - L_1, \ldots, L_n \in \text{Ground}(R; P, DB), \\
L_1, \ldots, L_n \in DB \}
\]

- \( T_P \) is called the immediate consequence operator.
- \( T^i_P(DB) = T_P(T^{i-1}_P(DB)) \) iterated application of \( T_P \).

Properties of \( T_P(DB) \)

- The set of facts is monotonically increasing i.e.:

\[
T^i_P(DB) \subseteq T^{i+1}_P(DB)
\]

- the sequence \( \langle T^i_P(DB) \rangle \) converges finitely: there exists \( n \) with \( T^m_P(DB) = T^n_P(DB) \) for all \( m \geq n \).

- \( T^i_P(DB) \) set of facts, to which \( \langle T^i_P(DB) \rangle \) converges is the result of the application of \( P \) to \( DB \).

- The operational semantics of a datalog program \( P \) assigns to each database \( DB \) the set of facts \( T^i_P(DB) \):

\[
\]

Theorem (Equivalence of semantics)

Assume a program \( P \). Then it holds that \( M[P] = O[P] \). In other words, for any database \( DB \), we have:

\[
\text{cons}(P^* \land DB^*) = T^*_P(DB)
\]
Proof of Theorem

Let \( P \) be a program and \( DB \) a database. We show

\[
\text{cons}(P^* \land DB^*) = T_{P}^{\omega}(DB).
\]

(1) We first show \( T_{P}^{\omega}(DB) \subseteq \text{cons}(P^* \land DB^*) \). By induction on \( i \), we show that \( T_{P}^{i}(DB) \subseteq \text{cons}(P^* \land DB^*) \) for every \( i \geq 0 \). Note that this includes the case where \( i = \omega \).

Base case. Assume \( i = 0 \). Take a fact \( L \in T_{P}^{0}(DB) \). Then by definition of \( T_{P}^{0}(DB) \), \( L \in DB \). By definition, \( DB^* \) is a conjunction of literals and \( L \) occurs in it. Hence, by classical logic, \( L \in \text{cons}(P^* \land DB^*) \).

The inductive step. Suppose \( T_{P}^{i}(DB) \subseteq \text{cons}(P^* \land DB^*) \) for \( i \geq 0 \). We show that \( T_{P}^{i+1}(DB) \subseteq \text{cons}(P^* \land DB^*) \). Recall that \( T_{P}^{i+1}(DB) = T_{P}(T_{P}^{i}(DB)) \). Thus by the definition of \( T_{P} \),

\[
T_{P}^{i+1}(DB) = T_{P}^{i}(DB) \cup \bigcup_{R \in P} \{ L_0 : L_0 := L_1, \ldots, L_n \in \text{Ground}(R, P, DB),
L_1, \ldots, L_n \in T_{P}^{i}(DB) \}
\]

(2) We show \( \text{cons}(P^* \land DB^*) \subseteq T_{P}^{\omega}(DB) \). To this end, we prove that \( L \notin T_{P}^{\omega}(DB) \) implies \( L \notin \text{cons}(P^* \land DB^*) \), for any fact \( L \). We thus simply show that \( T_{P}^{\omega}(DB) \) is a model of \( P^* \land DB^* \).

This suffices because of the following simple property: if \( M \) is a model of a formula \( F \), then any fact \( L \notin M \) is not a logical consequence of \( F \) (as witnessed by \( M \) itself).

By the induction hypothesis, \( T_{P}^{\omega}(DB) \subseteq \text{cons}(P^* \land DB^*) \). Thus it remains to show that \( L_0 \in \text{cons}(P^* \land DB^*) \) for any rule \( R \in P \) such that there is \( L_0 := L_1, \ldots, L_n \in \text{Ground}(R, P, DB) \) with \( L_1, \ldots, L_n \in T_{P}^{\omega}(DB) \).

Assume such a rule \( R = L_0^{'} := L_1^{'} , \ldots , L_n^{'} \) in \( P \), and suppose \( \pi \) is the substitution of variables with constants such that applying \( \pi \) to \( R \) results in \( L_0 := L_1, \ldots, L_n \), i.e., \( \pi(L_j^{'}) = L_j \) for \( j = 0, \ldots, n \).

By construction, in \( P^* \land DB^* \) we have the conjunct

\[
R^* = \forall x_1 \forall x_2 \ldots \forall x_i(\{ L_1^{'}, L_2^{'}, \ldots , L_n^{'} \} \Rightarrow L_0^{'}). 
\]

Thus, by employing the semantics of classical logic, for any variable substitution \( \pi' \) such that \( \{ \pi'(L_1^{'}) , \ldots , \pi'(L_n^{'}) \} \subseteq \text{cons}(P^* \land DB^*) \) we also have \( \pi'(L_0^{'}) \in \text{cons}(P^* \land DB^*) \). Since \( \pi \) is a substitution such that \( \{ \pi(L_1) , \ldots , \pi(L_n) \} \subseteq \text{cons}(P^* \land DB^*) \) by the induction hypothesis, we get \( \pi(L_0^{'}) = L_0 \in \text{cons}(P^* \land DB^*) \).

\[
T_{P}^{\omega}(DB) \text{ is a model of } DB^* \text{ because } DB = T_{P}^{\omega}(DB) \subseteq T_{P}^{\omega}(DB) \text{ by the definition of } T_{P}^{\omega}(DB). 
\]

It remains to show that \( T_{P}^{\omega}(DB) \) is also a model of \( P^* \). Consider an arbitrary rule \( R \in P \). We have to show that \( T_{P}^{\omega}(DB) \) is a model of \( R^* \) with \( R^* = \forall x_1 \forall x_2 \ldots \forall x_i(\{ L_1 \land L_2 \land \ldots \land L_n \} \Rightarrow L_0) \).

Consider an arbitrary (ground) variable assignement \( \pi \) on the variables \( x_1 , \ldots , x_i \). The only non-trivial case is that all facts \( \pi(L_1) , \ldots , \pi(L_n) \) are true in \( T_{P}^{\omega}(DB) \), i.e., \( \{ \pi(L_1) , \ldots , \pi(L_n) \} \subseteq T_{P}^{\omega}(DB) \).

We have to show that then also \( \pi(L_0) \) is true in \( T_{P}^{\omega}(DB) \), i.e., \( \pi(L_0) \in T_{P}^{\omega}(DB) \).

We know \( \pi(L_0) : = \pi(L_1) , \ldots , \pi(L_n) \in \text{Ground}(R, P, DB) \). Thus by the definition of \( T_{P} , \pi(L_0) \in T_{P}(T_{P}^{\omega}(DB)) \). Since \( T_{P}(T_{P}^{\omega}(DB)) = T_{P}^{\omega}(DB) \) by the definition of \( T_{P}^{\omega}(DB) \), we obtain \( \pi(L_0) \in T_{P}^{\omega}(DB) \).
Algorithm: INFER

**INPUT**: datalog program $P$, database $DB$

**OUTPUT**: $T_P^*(DB) \ (= \ cons(P^* \land DB^*))$

1. **STEP 1.** $GP := \bigcup_{R \in P} \text{Ground}(R; P, DB)$
   (* $GP$ . . . set of all ground instances *)

2. **STEP 2.** $OLD := \{\}; \ NEW := DB$

3. **STEP 3.** while $NEW \neq OLD$ do begin
   $OLD := NEW$;
   $NEW := \text{ComputeTP}(OLD)$;
   end;

4. **STEP 4.** output $OLD$.

Subroutine ComputeTP

**INPUT**: Set of facts $OLD$

**OUTPUT**: $T_P(OLD)$

1. **STEP 1.** $F := OLD$

2. **STEP 2.** for each rule $L_0 := L_1, \ldots, L_n$ in $GP$
   if $L_1, \ldots, L_n \in OLD$
   then $F := F \cup \{ L_0 \}$
   end;

3. **STEP 3.** return $F$;

Example

Apply the following program $P$ to calculate all ancestors of the above given database $DB$.

- $\text{ancestor}(X, Y) :- \text{parent}(X, Y)$.
- $\text{ancestor}(X, Z) :- \text{parent}(X, Y), \text{ancestor}(Y, Z)$.

Step 1. (INFER) build $GP$

$$GP = \{ \text{ancestor}(\text{grete}, \text{grete}) :- \text{parent}(\text{grete}, \text{grete}), \text{ancestor}(\text{grete}, \text{linda}) :- \text{parent}(\text{grete}, \text{linda}), \ldots, \text{ancestor}(\text{grete}, \text{grete}) :- \text{parent}(\text{grete}, \text{grete}), \text{ancestor}(\text{grete}, \text{grete}) :- \text{parent}(\text{grete}, \text{linda}), \text{ancestor}(\text{linda}, \text{grete}), \ldots \}.$$  (There are $6^2 + 6^3 = 252$ ground instances.)

Step 2. $OLD := \{\}, \ NEW := DB$

Step 3. $OLD \neq NEW$

Cycle 1: $OLD := DB, NEW := TP(OLD) = TP(DB)$

$$TP(OLD) = OLD \cup \{ \text{ancestor}(A, B) \mid \text{parent}(A, B) \in DB \}.$$  

Cycle 2: $OLD := TP(DB), NEW := TP(OLD) = TP(TP(DB))$

$$TP(OLD) = \text{OLD} \cup \{ \text{ancestor}(\text{hans}, \text{michael}), \text{ancestor}(\text{hans}, \text{gerti}), \text{ancestor}(\text{grete}, \text{michael}), \text{ancestor}(\text{grete}, \text{gerti}) \}.$$  

Cycle 3: $TP(OLD) = OLD$, there are no new facts

Step 4. Output of $OLD$.

The result corresponds to the extension of $DB$ with the new table $\text{ancestor}$.
Datalog with negation

Without negation, datalog is not relational complete because set difference \( R - S \) cannot be expressed.

We introduce the negation (\textit{not}) in bodies of rules.

Restriction on the application of the negation:

\begin{quote}
A relation \( R \) must not be defined on the basis of its negation.
\end{quote}

Check for this constraint: with graph-theoretic methods.

---

Example

The following program \( P \) with the rules:

\begin{align*}
\text{husband}(X) & : - \text{man}(X), \text{married}(X). \\
\text{bachelor}(X) & : - \text{man}(X), \text{not husband}(X).
\end{align*}

is stratified.
The program $P$ with the rules:

\[
\begin{align*}
\text{husband}(X) & : \text{man}(X), \text{not bachelor}(X). \\
\text{bachelor}(X) & : \text{man}(X), \text{not husband}(X).
\end{align*}
\]

is not stratified.

**Algorithm**

**INPUT:** A set of datalog rules.

**OUTPUT:** the decision whether the program is stratified and the classification of the predicates into strata.

**METHOD:**

1. initialize the strata for all predicates with 1.
2. do for all rules $R$ with predicate $p$ in the head:
   - if (i) the body of $R$ contains a negated predicate $q$,
     (ii) the stratum of $p$ is $i$, and
     (iii) the stratum of $q$ is $i \leq j$, then set $i := j + 1$.
   - if (i) the body of $R$ contains an unnegated predicate $q$,
     (ii) the stratum of $p$ is $i$, and
     (iii) the stratum of $q$ is $j$ with $i < j$, then set $i := j$.

until:

- status is stable $\Rightarrow$ program is stratified.
- stratum $n > \#$ predicates $\Rightarrow$ not stratified.

**Example**

Consider $R, S$ relations of the database $DB, P$:

\[
\begin{align*}
\text{v}(X,Y) & : \text{r}(X,X), \text{r}(Y,Y). \\
\text{u}(X,Y) & : \text{s}(X,Y), \text{s}(Y,Z), \text{not v}(X,Y). \\
\text{w}(X,Y) & : \text{not u}(X,Y), \text{v}(Y,X).
\end{align*}
\]

- $\text{w} \rightarrow \text{w}$ level 3
- $\text{u} \rightarrow \text{u}$ level 2
- $s, v \rightarrow \text{r, s, v}$ level 1
Semantics of datalog with negation

**Note:** when calculating the strata of a datalog program with negation the following holds:

1. **Step 1:** computation of all relations of the first stratum.
2. **Step i:** computation of all relations that belong to stratum i.
   - the relations computed in step \(i-1\) are known in step \(i\).

**Semantics** of datalog with negation is therefore uniquely defined.

Computation of \(P\) from the last example above:

1. **Step 1:** compute \(V\) from \(R\)
2. **Step 2:** compute \(U\) from \(S\) and \(V\)
3. **Step 3:** compute \(W\) from \(U\) and \(V\)

---

Properties of datalog with negation

- Datalog with negation is relational complete:
  - The difference \(D = R - S\) of two (e.g. binary) relations \(R\) and \(S\):
    \[
    d(X,Y) :- r(X,Y), \text{ not } s(X,Y).
    \]

- **Syntactical restrictions of datalog with negation:**
  - all variables that appear in the body within a negated literal must also appear in a positive (unnegated) literal

---

Example

Let \(DB\) be a database that contains information on graphs, with relations \(v(X)\), saying \(X\) is a node and \(e(X,Y)\) saying there is an edge from \(X\) to \(Y\).

Write a datalog program that computes all pairs of nodes \((X,Y)\), where \(X\) is a source, \(Y\) is a sink and \(X\) is connected to \(Y\).

\[
\begin{align*}
  p(X,Y) & :- source(X), sink(Y), connection(X,Y). \\
  connection(X,X) & :- v(X). \\
  connection(X,Y) & :- e(X,Z), connection(Z,Y). \\
  n_source(X) & :- e(Y,X). \\
  source(X) & :- v(X), \text{ not } n_source(X). \\
  n_sink(X) & :- e(X,Y). \\
  sink(X) & :- v(X), \text{ not } n_sink(X).
\end{align*}
\]
Learning objectives

- Motivation for Datalog (recursive queries)
- Syntax of Datalog
- Semantics of Datalog:
  - logical semantics,
  - operational semantics.
- Datalog with negation:
  - the need for negation,
  - the notions of dependency graph and stratification,
  - semantics of Datalog with negation.