Deductive databases

Motivation: Deductive DB

• Motivation is two-fold:
  – add deductive capabilities to databases; the database contains:
    • facts (intensional relations)
    • rules to generate derived facts (extensional relations)
    Database is knowledge base
  – Extend the querying
    • datalog allows for recursion
Motivation: Deductive DB

- Datalog as engine of deductive databases
  - similarities with Prolog
  - has facts and rules
  - rules define -possibly recursive- views
- Semantics not always clear
  - safety
  - negation
  - recursion

Outline

- Syntax of the Datalog language
- Semantics of a Datalog program
- Relational algebra = safe Datalog with negation and without recursion
- Optimization techniques
- Conclusions
Syntax of Datalog

• Datalog query/program:
  – facts → traditional relational tables
  – rules → define intensional views

• Rules
  – if-then rules
  – can contain recursion
  – can contain negations

• Semantics of program can be ambiguous

Example

father(X,Y) :- person(X,m), parent(X,Y).
grandson(X,Y) :- parent(Y,Z), parent(Z,X), person(X,m).
hbrothers(X,Y) :- person(X,m), person(Y,m),
                 parent(Z,X), parent(Z,Y).

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Parent</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>ingrid</td>
<td>f</td>
<td>ingrid</td>
<td>toon</td>
</tr>
<tr>
<td>alex</td>
<td>m</td>
<td>alex</td>
<td>toon</td>
</tr>
<tr>
<td>rita</td>
<td>f</td>
<td>rita</td>
<td>an</td>
</tr>
<tr>
<td>jan</td>
<td>m</td>
<td>jan</td>
<td>an</td>
</tr>
<tr>
<td>toon</td>
<td>m</td>
<td>an</td>
<td>bernd</td>
</tr>
<tr>
<td>an</td>
<td>f</td>
<td>an</td>
<td>mattijs</td>
</tr>
<tr>
<td>bernd</td>
<td>m</td>
<td>toon</td>
<td>bernd</td>
</tr>
<tr>
<td>mattijs</td>
<td>m</td>
<td>toon</td>
<td>mattijs</td>
</tr>
</tbody>
</table>
Syntax of Datalog

• Variables: X, Y
• Constants: m, f, rita, ...
• Positive literal: p(t1,...,tn)
  – p is the name of a relation (EDB or IDB)
  – t1, ..., tn constants or variables
• Negative literal: not p(t1, ..., tn)
• Rule: h :- l1, ..., ln
  – h positive literal, l1, ..., ln literals

Syntax of Datalog

• Rule can be recursive
• Arithmetic operations considered as special predicates
  – A<B : smaller(A,B)
  – A+B=C : plus(A,B,C)
Outline

• Syntax of the Datalog language
• **Semantics of a Datalog program**
  – non-recursive
  – recursive datalog
  – aggregation
• Relational algebra = safe Datalog with negation and without recursion
• Optimization techniques
• Conclusions

Semantics of **Non-Recursive** Datalog Programs

• *Ground instantiation of a rule*
  \( h :- l_1, \ldots, l_n \) : replace every variable in the rule by a constant

**Example:**

father\((X,Y)\) \(:=\) person\((X,m)\), parent\((X,Y)\)

instantiation:

father\((toon,an)\) \(:=\) persoon\((toon,m)\),
  parent\((toon,an)\).
Semantics of Non-Recursive Datalog Programs

Let I be a set of facts
The body of a rule instantiation R’ is satisfied by I if:
– every positive literal in the body of R’ is in I
– no negative literal in the body of R’ is in I

Example:
persoon(toon,m), parent(toon,an) not satisfied by the facts given before

Semantics of Non-Recursive Datalog Programs

Let I be a set of facts
R is a rule h:-l1, …, ln

\[ \text{Infer}(R, I) = \{ h' : \]
– h’:-l1’, …, ln’ ground instantiation of R
– l1’ … ln’ satisfied by I \}

\( R = \{ R_1, \ldots, R_n \} \)
\[ \text{Infer}(R, I) = \text{Infer}(R_1, I) \cup \ldots \cup \text{Infer}(R_n, I) \]
Semantics of **Non-Recursive**
Datalog Programs

• A rule \( h : - l_1, \ldots, l_n \) is in layer 1:
  – \( l_1, \ldots, l_n \) only involve extensional predicates

• A rule \( h : - l_1, \ldots, l_n \) is in layer \( i \)
  – for all \( 0 < j < i \), it is not in layer \( j \)
  – \( l_1, \ldots, l_n \) only involve predicates that are
    extensional and in the layers \( 1, \ldots, i-1 \)

Semantics of **Non-Recursive**
Datalog Programs

• Let \( I_0 \) be the facts in a datalog program
  Let \( R_1 \) be the rules at layer 1

  …

  Let \( R_n \) be the rules at layer \( n \)

• \( I_1 = I_0 \cup \text{Infer}(R_1, I_0) \)
  \( I_2 = I_1 \cup \text{Infer}(R_2, I_1) \)

  …

  \( I_n = I_{n-1} \cup \text{Infer}(R_n, I_{n-1}) \)
Semantics of **Non-Recursive Datalog Programs**

**Example:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Father</th>
<th>Grandson</th>
</tr>
</thead>
<tbody>
<tr>
<td>ingrid</td>
<td>f</td>
<td>ingrid</td>
<td>toon</td>
</tr>
<tr>
<td>alex</td>
<td>m</td>
<td>alex</td>
<td>toon</td>
</tr>
<tr>
<td>rita</td>
<td>f</td>
<td>rita</td>
<td>an</td>
</tr>
<tr>
<td>jan</td>
<td>m</td>
<td>jan</td>
<td>an</td>
</tr>
<tr>
<td>toon</td>
<td>m</td>
<td>toon</td>
<td>bernd</td>
</tr>
<tr>
<td>an</td>
<td>f</td>
<td>an</td>
<td>mattijs</td>
</tr>
<tr>
<td>bernd</td>
<td>m</td>
<td>toon</td>
<td>bernd</td>
</tr>
<tr>
<td>mattijs</td>
<td>m</td>
<td>toon</td>
<td>mattijs</td>
</tr>
</tbody>
</table>

- father(X,Y) :- person(X,m), parent(X,Y).
- grandson(X,Y) :- parent(Y,Z), parent(Z,X), person(X,m).
- hbrothers(X,Y) :- person(X,m), person(Y,m), parent(Z,X), parent(Z,Y).

**Safety**

- A rule can make no sense if variables appear in funny ways

Examples:

- S(x) :- R(y)
- S(x) :- **not** R(x)
- S(x) :- R(y), x<y

In each of these cases the result is infinite even if the relation R is finite
Safety

- Therefore, we will only consider rules that are safe.
- A rule \( h : - l_1, \ldots, l_n \) is safe if:
  - every variable in the head of the rule: also in a non-arithmetic positive literal in body
  - every variable in a negative literal of the body: also in some positive literal of the body

Semantics of Non-Recursive Datalog Programs

- For non-recursive, safe datalog programs semantics is well defined
  - all facts that can be derived from the program
  - smallest « model » consistent with the program (unique!)
- Closed-World Assumption: fact is only « true » if it can be derived from the program
Outline

• Syntax of the Datalog language
• **Semantics of a Datalog program**
  – non-recursive
  – recursive datalog
  – aggregation
• Relational algebra = safe Datalog with negation and without recursion
• Optimization techniques
• Conclusions

Semantics of **Recursive** Datalog Programs

• Semantics of recursive datalog is less clear.

Example:

T(a).
R(X) :- T(X), not S(X).
S(X) :- T(X), not R(X).

What about R(a)? S(a)?
Semantics of **Recursive** Datalog Programs

- Therefore: notion of a *stratified* program
- T **depends on** S if some rule with T in the head contains S or (recursively) some predicate that depends on S, in the body.
- **Stratified program:** If T depends on **not** S, then S cannot depend on T (or **not** T).

Semantics of **Recursive** Datalog Programs

- If a program is stratified, the tables in the program can be partitioned into strata:
  - Stratum 0: All database tables.
  - Stratum I: Tables defined in terms of tables in Stratum I and lower strata.
  - If T depends on **not** S, S is in lower stratum than T.
Semantics of **Recursive** Datalog Programs

• Semantics of a stratified program given by:
  – First, compute the *least fixpoint* of all tables in Stratum 1. (Stratum 0 tables are fixed.)
  – Then, compute the *least fixpoint* of tables in Stratum 2; then the lfp of tables in Stratum 3, and so on, stratum-by-stratum.

Semantics of **Recursive** Datalog Programs

• Fixpoint of a set of rules \( R \), starting with set of facts \( I \):  
  
  \[
  \text{repeat} \\
  \phantom{\text{repeat}} \quad \text{Old}_I := I \\
  \phantom{\text{repeat}} \quad I := I \cup \text{infer}(R,I) \\
  \text{until} \quad I = \text{Old}_I
  \]

• Fixpoint **within** one stratum always terminates
The program:

\[
\begin{align*}
T(a). \\
R(X) &: T(X), \textbf{not} S(X). \\
S(X) &: T(X), \textbf{not} R(X).
\end{align*}
\]

is not stratified;

R depends negatively on S
S depends negatively on R

---

\[
\begin{align*}
g(a,b). & \quad g(b,c). \quad g(a,d). \\
gerach(X,X) &: g(X,Y). \\
gerach(X,Y) &: g(X,Y). \\
gerach(X,Z) &: \text{reach}(X,Y), \text{reach}(Y,Z). \\
node(X) &: g(X,Y). \\
nnode(Y) &: g(X,Y). \\
unreach(X,Y) &: \text{node}(X), \text{node}(Y), \textbf{not} \text{reach}(X,Y).
\end{align*}
\]
Semantics of Recursive Datalog Programs

Stratum 0: g(a,b). g(b,c). g(a,d).

Stratum 1:
node(a), node(b), node(c), node(d),
reach(a,a), reach(b,b), reach(c,c),
reach(d,d), reach(a,b), reach(b,c), …

Stratum 2:
unreach(b,a), unreach(c,a), …

Outline

• Syntax of the Datalog language
• Semantics of a Datalog program
  – non-recursive
  – recursive datalog
  – aggregation
• Relational algebra = safe Datalog with
  negation and without recursion
• Optimization techniques
• Conclusions
Aggregate Operators

\[
\text{Degree}(X, \text{SUM}(<Y>)) \leftarrow g(X,Y).
\]

• The \(< \ldots >\) notation in the head indicates grouping; the remaining arguments (\(X, \text{in this example}\)) are the GROUP BY fields.
• In order to apply such a rule, must have all of relation \(g\) available.
• Stratification with respect to use of \(< \ldots >\) is similar to negation.

Outline

• Syntax of the Datalog language
• Semantics of a Datalog program
• Relational algebra = Safe Datalog with negation and without recursion
• Optimization techniques
• Conclusions
RA = Non-Recursive Datalog

• Every operator of RA can be simulated by non-recursive datalog
  – Project out attribute account_name from account.
    
    \[ \text{query} (A) :- \text{account} (A, N, B). \]
  – Cartesian product of relations \( r_1 \) and \( r_2 \).
    
    \[ \text{query} (X_1, X_2, ..., X_n, Y_1, Y_2, ..., Y_m) :- r_1 (X_1, X_2, ..., X_n), r_2 (Y_1, Y_2, ..., Y_m). \]
  – Union of relations \( r_1 \) and \( r_2 \).
    
    \[ \text{query} (X_1, X_2, ..., X_n) :- r_1 (X_1, X_2, ..., X_n), \]
    \[ \text{query} (X_1, X_2, ..., X_n) :- r_2 (X_1, X_2, ..., X_n), \]
  – Set difference of \( r_1 \) and \( r_2 \).
    
    \[ \text{query} (X_1, X_2, ..., X_n) :- r_1 (X_1, X_2, ..., X_n), \]
    \[ \text{not } r_2 (X_1, X_2, ..., X_n) \]

RA = Non-Recursive Datalog

• Every rule can be expressed by one RA expression: expression layer-by-layer
• Recursive datalog with negation is more powerful than relational algebra
  – Transitive closure:
    
    reach(X,X) :- g(X,Y).
    reach(Y,Y) :- g(X,Y).
    reach(X,Y) :- g(X,Y).
    reach(X,Z) :- reach(X,Y), reach(Y,Z).
Outline

- Syntax of the Datalog language
- Semantics of a Datalog program
- Relational algebra = Datalog with negation and without recursion
- **Optimization techniques**
- Conclusions

Evaluation of Datalog Programs

Running example:

```prolog
root(r). child(r,a). child(r,b). child(a,c).
child(a,d). child(c,e). child(d,f). child(b,h).

sg(X,Y) :- root(X),root(Y).
sg(X,Y) :- child(X,U), sg(U,V),
        child(Y,V).
```

![Datalog tree diagram]
Evaluation of Datalog Programs

• **Repeated inferences:** recursive rules are repeatedly applied in the naïve way; same inferences in several iterations.

• **Unnecessary inferences:** if we just want to find sg of a particular node, say \( e \), computing the fixpoint of the sg program and then selecting tuples with \( e \) in the first column is wasteful, in that we compute many irrelevant facts.

Evaluation of Datalog Programs

Running example:

Query: \(?\ sg(e,X)\)

1. \((r, r)\)
2. \((a,a), (b,b), (a,b), (b,a)\)
3. \((c,c), (c,d), (c,h), (d,c), (d,d), \ldots\)
4. \((e,e), (f,f), (e,f), (f,e)\)
Avoiding Repeated Inferences

- **Semiaive Fixpoint Evaluation**: Avoid repeated inferences: at least one of the body facts generated in the most recent iteration.
  - For each recursive table $P$, use a table $\delta_P$.
  - Rewrite the program to use the delta tables.

Avoiding Unnecessary Inferences

- Still, in the running example:
  - many unnecessary deductions when query is $\text{? sg(e, X)}$

- Compare with top-down
  - as in Prolog
  - only facts that are connected to the ultimate goal are being considered
"Magic Sets" Idea

- **Idea**: Define a "filter" table: computes all relevant values, restricts the computation of $\text{sg}(e,X)$.

$$
\text{sg}(X,Y) \ :- \ m(X), \ \text{root}(X), \ \text{root}(Y).
$$

$$
\text{sg}(X,Y) \ :- \ m(X), \ \text{child}(X,U), \ \text{sg}(U,V), \ \text{child}(Y,V).
$$

$$
m(X) \ :- \ m(Y), \ \text{child}(Y,X).
$$

$$
m(e).
$$

Magic Sets

- It is *always* possible to do this in such a way that bottom-up becomes as efficient as top-down!

- Different proposals exist in literature
  - how to introduce the magic filters
Outline

• Syntax of the Datalog language
• Semantics of a Datalog program
• Relational algebra = Datalog with negation and without recursion
• Optimization techniques
• Conclusions

Conclusions

• Datalog adds deductive capabilities to databases
  – extensional relations
  – intensional relations
• Datalog without recursion
  – safety requirement
  – semantics based on layers, minimal model
  – equal in power to relational algebra
Conclusions

• Datalog with recursion
  – semantics not always clear
  – stratified negation: least fixpoints interpretation

• Evaluation of datalog queries:
  – without negation = RA-optimization
  – with recursion:
    • semi-naive recursion
    • magic sets

• Very nice idea, but …
• Deductive databases did not make it as a database paradigm

• Yet, many ideas survived
  – recursion in SQL …
• And others may re-surface in future.