Scalable Uncertainty Management

03 – Provenance

Rainer Gemulla

May 18, 2012
Overview

In this lecture

- Introduction to datalog
- What is provenance?
- Which types of provenance do exist?
  - Lineage
  - Why-provenance
  - How-provenance
- How to compute provenance?
- How do the types of provenance relate to each other?
- How to derive provenance information for datalog?

Not in this lecture

- Uncertainty
- Where-provenance
Outline

1 Datalog

2 Introduction to Provenance
   - Lineage
   - Why-provenance
   - How-provenance

3 Provenance Semirings

4 How-Provenance for nr-datalog

5 Summary
**Datalog**

- Datalog is a declarative language
- Datalog program is collection of if-then rules
- Supports recursion (in contrast to relational algebra)
- *Datalog* is a logic for relations (“database logic”)
- Datalog is based on Prolog
  - No function symbols + safety condition
  - Unique and finite minimum model
  - Unique and finite minimum fixpoint
  - Expressive power in PTIME

**Example**

\[
\text{ancestor}(x, z) \leftarrow \text{parent}(x, z)
\]

\[
\text{ancestor}(x, z) \leftarrow \text{ancestor}(x, y), \text{parent}(y, z)
\]

Straightforward translation to first-order logic:

\[
(\forall x)(\forall z) \quad \text{parent}(x, z) \rightarrow \text{ancestor}(x, z)
\]

\[
(\forall x)(\forall y)(\forall z) \quad \text{ancestor}(x, y) \land \text{parent}(y, z) \rightarrow \text{ancestor}(x, z)
\]
Predicates and atoms

- Relations are represented by *predicates* of same arity
  - For relation name $R$, we use predicate name $R$
  - Order of predicate arguments = natural order of relation attributes
- Predicate with arguments is called a *relational atom*
  - $R(a_1, \ldots, a_k)$ returns TRUE if $(a_1, \ldots, a_k) \in I(R)$
  - FALSE otherwise (*closed word assumption*)
- Predicate can take *constants* and *variables* as arguments
  - Atom with variables = function that takes values for variables and returns TRUE/FALSE

**Example**

For simplicity, we denote both predicate and its interpretation by $R$.

- $R(a_1, b_1) = \text{TRUE}$
- $R(a_2, b_2) = \text{TRUE}$
- $R(a_3, b_3) = \text{FALSE}$
- $R(x, b_1) = f(x) = \begin{cases} \text{TRUE} & \text{if } x = a_1 \\ \text{FALSE} & \text{otherwise} \end{cases}$
Extended datalog: arithmetic atoms

- Comparison between two arithmetic expressions
  - Arithmetic predicates: =, <, >, ≤, ≥, ...
  - Arithmetic expressions: constants, variables, +, −, ×, /, ...

- Arithmetic predicates are like infinite relations
  - Database relations are finite and may change
  - Arithmetic relations are infinite and unchanging

Example

- \( x < y \)
- \( x + 1 \geq y + 4 \times z \)
- \( x < 5 = f(x) = \begin{cases} \text{TRUE} & \text{if } x < 5 \\ \text{FALSE} & \text{otherwise} \end{cases} \)
- \( "<" = \{ (1, 2), (-1.5, 65.4), \ldots \} \)
Datalog rules

- Operations are described by *datalog rules*
  1. A relational atom called *head*
  2. The symbol \( \leftarrow \) (read as “if”)
  3. A *body* consisting of one or more atoms, called *subgoals*
     (connected by \( \land \); in datalog \( \neg \): optionally preceded by \( \neg \))

Example

A movie schema:

\[
\text{Movies(Title, Year, Length, Genre, StudioName, Producer).}
\]

A \( \mathcal{RA} \) expression:

\[
\text{LongMovie} := \pi_{\text{Title, Year}}(\sigma_{\text{Length} \geq 100}(\text{Movies})).
\]

Corresponding datalog rule:

\[
\text{LongMovie}(t, y) \leftarrow \text{Movies}(t, y, l, g, s, p), \quad l \geq 100.
\]
Semantics of rules

1. Possible assignments
   - Let the variables in the rule range over all possible values
   - When all subgoals are TRUE, insert tuple into the head’s relation

2. Nonnegated relational subgoals
   - Consider sets of tuples for each nonnegated relational subgoal
   - Check whether assignment is consistent (same variable, same value)
   - If so, check negated subgoals and arithmetic subgoals
   - If all checks successful, insert tuple into the head’s relation

Example

\[ P(x, z) \leftarrow Q(x, y), R(y, z), \neg Q(x, z) \]

<table>
<thead>
<tr>
<th>Q(x, y)</th>
<th>R(y, z)</th>
<th>Consistent?</th>
<th>\neg Q(x, z)?</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) (1, 2)</td>
<td>(2, 3)</td>
<td>Yes</td>
<td>No</td>
<td>—</td>
</tr>
<tr>
<td>2) (1, 2)</td>
<td>(3, 1)</td>
<td>No; \ y = 2, 3</td>
<td>Irrelevant</td>
<td>—</td>
</tr>
<tr>
<td>3) (1, 3)</td>
<td>(2, 3)</td>
<td>No; \ y = 3, 2</td>
<td>Irrelevant</td>
<td>—</td>
</tr>
<tr>
<td>4) (1, 3)</td>
<td>(3, 1)</td>
<td>Yes</td>
<td>Yes</td>
<td>( P(1, 1) )</td>
</tr>
</tbody>
</table>

CWA
Safe rules

Not all rules give a meaningful (i.e., finite) result → safety condition.

Example

- Safe:
  \[ \text{LongMovie}(t, y) \leftarrow \text{Movies}(t, y, l, g, s, p), \; l \geq 100 \]

- In safe rules, abbreviation \_ for variables that occur only once
  \[ \text{LongMovie}(t, y) \leftarrow \text{Movies}(t, y, l, \_, \_, \_, \_), \; l \geq 100 \]

- Unsafe: \( P(x) \leftarrow Q(y) \)
- Unsafe: \( P(x) \leftarrow \neg Q(x) \)
- Unsafe: \( P(x, y) \leftarrow Q(y), \; x > y \)

Definition

A rule is safe if every variable that appears anywhere in the rule also appears in some nonnegated, relational subgoal of the body. This condition is called the safety condition.
Extensional and intensional predicates

**Definition**

- **Extensional predicates** (EDB) are predicates whose relations are stored in a database. They can only occur in the bodies of datalog rules.
- **Intensional predicates** (IDB) are predicates whose relations is computed by applying datalog rules. They can occur in heads and bodies of datalog rules.

- “Extension” is another name for “instance of a relation”
- “Intensional” relations are defined by the programmer’s “intent”

**Example**

\[ \text{LongMovie}(t, y) \leftarrow \text{Movies}(t, y, l, -, -, -), l \geq 100 \]

- Movies is an EDB predicate (or relation)
- LongMovie is an IDB predicate (or relation)
Datalog queries

A *datalog query* is a collection of one or more rules (often with a designated output relation).

**Example**

Schema (EDB):
- Hotel(HotelNo, Name, City)
- Room(RoomNo, HotelNo, Type, Price)

RA query:
\[
\pi_{\text{HotelNo, Name, City}}(\text{Hotel} \Join \sigma_{\text{Price} > 500 \lor \text{Type} = 'suite'}(\text{Room}))
\]

Datalog query:
- \(\text{ExpensiveRoom}(r, h, t, p) \leftarrow \text{Room}(r, h, t, p), \ p > 500\)
- \(\text{ExpensiveRoom}(r, h, t, p) \leftarrow \text{Room}(r, h, t, p), \ t = 'suite'\)
- \(\text{ExpensiveHotelRoom}(h, n, c, r, t, p) \leftarrow \text{Hotel}(h, n, c), \ \text{ExpensiveRoom}(r, h, t, p)\)
- \(\text{ExpensiveHotel}(h, n, c) \leftarrow \text{ExpensiveHotelRoom}(h, n, c, _, _, _)\)
Datalog and relational algebra

Example (Recursive query)

\[
\text{ancestor}(x, z) \leftarrow \text{parent}(x, z) \\
\text{ancestor}(x, z) \leftarrow \text{ancestor}(x, y), \text{parent}(y, z)
\]

- **Nonrecursive** if the rules can be ordered such that the head predicate of each rule does not occur in a body of the current or a previous rule
- \textit{nr-datalog}: nonrecursive, no negation
- \textit{nr-datalog}^{-}: nonrecursive, with negation

**Theorem**

- \textit{nr-datalog} and \textit{SPJRU} queries have equivalent expressive power.
- \textit{nr-datalog}^{-} and relational algebra have equivalent expressive power.

We will switch between datalog and (subsets of) RA as convenient.
Outline

1 Datalog

2 Introduction to Provenance
   - Lineage
   - Why-provenance
   - How-provenance

3 Provenance Semirings

4 How-Provenance for nr-datalog

5 Summary
Provenance and annotation management

- *Provenance* describes origins and history of data
- *Annotations* describe auxiliary information associated with the data

### NYRestaurants

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Cost</th>
<th>Type</th>
<th>Zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peacock Alley</td>
<td>$$$</td>
<td>French</td>
<td>10022</td>
</tr>
<tr>
<td>Bull &amp; Bear</td>
<td>$$$</td>
<td>Seafood</td>
<td>10022</td>
</tr>
<tr>
<td>Pacifica</td>
<td>$</td>
<td>Chinese</td>
<td>10013</td>
</tr>
<tr>
<td>Soho Kitchen &amp; Bar</td>
<td>$</td>
<td>American</td>
<td>10022</td>
</tr>
</tbody>
</table>

Serves fine French Cuisine in elegant setting. Formal attire.

Extensive wine list!

Yummy chicken curry!!

### Cheap Restaurants

<table>
<thead>
<tr>
<th>Restaurant</th>
<th>Cost</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pacifica</td>
<td>$</td>
<td>Chinese</td>
</tr>
<tr>
<td>Soho Kitchen &amp; Bar</td>
<td>$</td>
<td>American</td>
</tr>
</tbody>
</table>

Outline

1. Datalog

2. Introduction to Provenance
   - Lineage
     - Why-provenance
     - How-provenance

3. Provenance Semirings

4. How-Provenance for nr-datalog

5. Summary
Tuple location

**Definition**

A tuple $t$ tagged with a relation name $R$ is called a *tuple location* and denoted $(R, t)$ or simply $R(t)$. We can view a database instance $I(R)$ on $R$ as a set $\{ (R, t) \mid R \in R, \ t \in I(R) \}$.

**Example**

<table>
<thead>
<tr>
<th>Agencies (A)</th>
<th>ExternalTours (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Dest.</strong></td>
</tr>
<tr>
<td>$t_1$ BayTours</td>
<td>SFO</td>
</tr>
<tr>
<td>$t_2$ HarborCruz</td>
<td>SC</td>
</tr>
<tr>
<td>$t_3$ BayTours</td>
<td>SC</td>
</tr>
<tr>
<td>$t_4$ BayTours</td>
<td>MRY</td>
</tr>
<tr>
<td>$t_5$ HarborCruz</td>
<td>Carmel</td>
</tr>
</tbody>
</table>

- Tuple locations: $A(t_1), A(t_2), A(\langle\text{FunTravel, SJ, 415-2400}\rangle), \ldots$
- Database instance: $\{ A(t_1), A(t_2), E(t_3), E(t_4), \ldots, E(t_8) \}$
Lineage

Definition (informal)

The \textit{lineage} of a tuple $t$ (w.r.t. a query) consists of all tuples of the input data that “contributed to” or “helped produce” $t$.

Example

<table>
<thead>
<tr>
<th>Agencies (A)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>BasedIn</td>
<td>Phone</td>
</tr>
<tr>
<td>$t_1$</td>
<td>BayTours</td>
<td>SFO</td>
</tr>
<tr>
<td>$t_2$</td>
<td>HarborCruz</td>
<td>SC</td>
</tr>
</tbody>
</table>

| ExternalTours (E) |          |          |          |
| Name         | Dest.    | Type     | Price    |
| $t_3$        | BayTours | SFO      | Cable    | $50     |
| $t_4$        | BayTours | SC       | Bus      | $100    |
| $t_5$        | BayTours | SC       | Boat     | $250    |
| $t_6$        | BayTours | MRY      | Boat     | $400    |
| $t_7$        | HarborCruz| MRY     | Boat     | $200    |
| $t_8$        | HarborCruz| Carmel  | Train    | $90     |

BoatAgencies($n, p$) $\leftarrow$ Agencies($n$, $\_$, $p$), ExternalTours($n$, $\_$, ’Boat’, $\_$).

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>BayTours</td>
<td>415-1200</td>
</tr>
<tr>
<td>HarborCruz</td>
<td>831-3000</td>
</tr>
</tbody>
</table>

Lineage

\{ $A(t_1), E(t_5), E(t_6)$ \}
\{ $A(t_2), E(t_7)$ \}
Lineage & query rewriting

Example

Two equivalent queries:

\[ q(x, y) \leftarrow R(x, y) \]
\[ q'(x, y) \leftarrow R(x, y), R(x, z). \]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Theorem

Lineage is sensitive to query rewriting.
Application: Lineage tracing in data warehouses

- Data warehouses integrates data from multiple sources
- Warehouse directly used for coarse-grained analysis
- In-depth analysis requires access to source data

→ view data lineage problem

Lineage tracing in the WHIPS data warehouse system
Outline

1. Datalog

2. Introduction to Provenance
   - Lineage
   - Why-provenance
   - How-provenance

3. Provenance Semirings

4. How-Provenance for nr-datalog

5. Summary
**Definition**

Let \( I \) be a database instance over \( R \), \( q \) a query over \( R \), and \( t \in q(I) \). An instance \( J \subseteq I \) is a **witness for** \( t \) **with respect to** \( q \) if \( t \in q(J) \). The set of all witnesses is given by \( \text{Wit}(q, I, t) = \{ J \subseteq I \mid t \in q(J) \} \).

**Example**

<table>
<thead>
<tr>
<th>Agencies (A)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>BasedIn</td>
<td>Phone</td>
</tr>
<tr>
<td>BayTours</td>
<td>SFO</td>
<td>415-1200</td>
</tr>
<tr>
<td>HarborCruz</td>
<td>SC</td>
<td>831-3000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ExternalTours (E)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Dest.</td>
<td>Type</td>
</tr>
<tr>
<td>BayTours</td>
<td>SFO</td>
<td>Cable</td>
</tr>
<tr>
<td>BayTours</td>
<td>SC</td>
<td>Bus</td>
</tr>
<tr>
<td>BayTours</td>
<td>SC</td>
<td>Boat</td>
</tr>
<tr>
<td>BayTours</td>
<td>MRY</td>
<td>Boat</td>
</tr>
<tr>
<td>HarborCruz</td>
<td>MRY</td>
<td>Boat</td>
</tr>
<tr>
<td>HarborCruz</td>
<td>Carmel</td>
<td>Train</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BoatAgencies</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Phone</td>
</tr>
<tr>
<td>BayTours</td>
<td>415-1200</td>
</tr>
<tr>
<td>HarborCruz</td>
<td>831-3000</td>
</tr>
</tbody>
</table>

**Lineage**

- \( t_9 \):
  - \( \{ A(t_1), E(t_5), E(t_6) \} \)
  - \( \{ A(t_2), E(t_7) \} \)

**Witnesses for**

1. \( t_9 \):
   - \( \{ A(t_1), E(t_5) \} \)
   - \( \{ A(t_1), E(t_6) \} \)
   - \( \{ A(t_1), E(t_5), E(t_6) \} \), . . .

2. \( t_{10} \):
   - \( \{ A(t_2), E(t_7) \} \)
   - \( \{ A(t_1), A(t_2), E(t_7) \} \)
   - . . .

\( I \) is a witness for both \( t_9 \) and \( t_{10} \).
Minimal why-provenance

**Definition**

A *minimal witness* is a minimal element of \(\text{Wit}(q, I, t)\). The set of minimal witnesses is called *minimal why-provenance* and is given by

\[ \text{MWhy}(q, I, t) = \{ J \in \text{Wit}(q, I, t) \mid (\forall J' \in \text{Wit}(q, I, t)) J' = J \lor J' \not\subset J \} \]

**Example**

<table>
<thead>
<tr>
<th>Agencies (A)</th>
<th>ExternalTours (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>BasedIn</strong></td>
</tr>
<tr>
<td>(t_1)</td>
<td>BayTours</td>
</tr>
<tr>
<td>(t_2)</td>
<td>HarborCruz</td>
</tr>
<tr>
<td>(t_3)</td>
<td>BayTours</td>
</tr>
<tr>
<td>(t_4)</td>
<td>BayTours</td>
</tr>
<tr>
<td>(t_5)</td>
<td>BayTours</td>
</tr>
<tr>
<td>(t_6)</td>
<td>BayTours</td>
</tr>
<tr>
<td>(t_7)</td>
<td>HarborCruz</td>
</tr>
<tr>
<td>(t_8)</td>
<td>HarborCruz</td>
</tr>
</tbody>
</table>

**BoatAgencies**

<table>
<thead>
<tr>
<th><strong>Name</strong></th>
<th><strong>Phone</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_9)</td>
<td>BayTours</td>
</tr>
<tr>
<td>(t_{10})</td>
<td>HarborCruz</td>
</tr>
</tbody>
</table>

**Minimal why-provenance**

\[ \{ \{ A(t_1), E(t_5) \}, \{ A(t_1), E(t_6) \} \}, \{ A(t_2), E(t_7) \} \]
Minimal why-provenance & query rewriting

Example

Two equivalent queries:

\[ q(x, y) \leftarrow R(x, y) \]
\[ q'(x, y) \leftarrow R(x, y), R(x, z). \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>R</th>
<th>q(R)</th>
<th>q'(R)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>Min. why</td>
<td>Min. why</td>
</tr>
<tr>
<td>t₁</td>
<td>1</td>
<td>2</td>
<td>{ { R(t₁) } }</td>
<td>{ { R(t₁) } }</td>
</tr>
<tr>
<td>t₂</td>
<td>1</td>
<td>3</td>
<td>{ { R(t₂) } }</td>
<td>{ { R(t₂) } }</td>
</tr>
<tr>
<td>t₃</td>
<td>4</td>
<td>2</td>
<td>{ { R(t₃) } }</td>
<td>{ { R(t₃) } }</td>
</tr>
</tbody>
</table>

Theorem

*Minimal why-provenance is insensitive to query rewriting.*
Application: View deletion problem

- Let $I$ be a database instance and consider view $V = q(I)$
- **View deletion problem:** Find the set of tuples $\Delta I$ to remove from $I$ so that a tuple $t$ is removed from $V$
- Intuitively, all minimal witnesses must be destroyed; many ways, e.g.,
  1. **Source side-effect problem:** Minimize changes to the source ($|\Delta I|$)
  2. **View side-effect problem:** Minimize changes to the view ($|\Delta V|$)
- Both NP-hard for PJ and JU queries!

**Example**

BayTours does not offer boat tours anymore → delete $t_9$.

<table>
<thead>
<tr>
<th>BoatAgencies</th>
<th>Phone</th>
<th>Min. why</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_9$</td>
<td>BayTours</td>
<td>415-1200</td>
</tr>
<tr>
<td>$t_{10}$</td>
<td>HarborCruz</td>
<td>831-3000</td>
</tr>
</tbody>
</table>

Examples:

- delete $A(t_1)$: optimum for both problems
- delete $E(t_5)$ and $E(t_6)$: optimum for (1) when $A \bowtie E$ is taken as source

Buneman et al., PODS, 2002.
Outline

1. Datalog

2. Introduction to Provenance
   - Lineage
   - Why-provenance
   - How-provenance

3. Provenance Semirings

4. How-Provenance for nr-datalog

5. Summary
How-provenance

Definition (informal)

The *how-provenance* of a tuple $t$ describes how $t$ is derived according to the query. It makes use of two “operations”: combine ($\cdot$) and merge ($+$).

Example

<table>
<thead>
<tr>
<th>Agencies (A)</th>
<th>Name</th>
<th>BasedIn</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>BayTours</td>
<td>SFO</td>
<td>415-1200</td>
</tr>
<tr>
<td>$t_2$</td>
<td>HarborCruz</td>
<td>SC</td>
<td>831-3000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ExternalTours (E)</th>
<th>Name</th>
<th>Dest.</th>
<th>Type</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_3$</td>
<td>BayTours</td>
<td>SFO</td>
<td>Cable</td>
<td>$50</td>
</tr>
<tr>
<td>$t_4$</td>
<td>BayTours</td>
<td>SC</td>
<td>Bus</td>
<td>$100</td>
</tr>
<tr>
<td>$t_5$</td>
<td>BayTours</td>
<td>SC</td>
<td>Boat</td>
<td>$250</td>
</tr>
<tr>
<td>$t_6$</td>
<td>BayTours</td>
<td>MRY</td>
<td>Boat</td>
<td>$400</td>
</tr>
<tr>
<td>$t_7$</td>
<td>HarborCruz</td>
<td>MRY</td>
<td>Train</td>
<td>$90</td>
</tr>
<tr>
<td>$t_8$</td>
<td>HarborCruz</td>
<td>Carmel</td>
<td>Train</td>
<td>$200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BoatAgencies</th>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>BayTours</td>
<td>415-1200</td>
<td></td>
</tr>
<tr>
<td>HarborCruz</td>
<td>831-3000</td>
<td></td>
</tr>
</tbody>
</table>

How-provenance

$A(t_1) \cdot E(t_5) + A(t_1) \cdot E(t_6)$

$A(t_2) \cdot E(t_7)$
How-provenance & query rewriting

Example

Two equivalent queries:

\[ q(x, y) \leftarrow R(x, y) \]
\[ q'(x, y) \leftarrow R(x, y), R(x, z). \]

<table>
<thead>
<tr>
<th>( R )</th>
<th>( q(R) )</th>
<th>( q'(R) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( 1 )</td>
<td>( 2 )</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( 1 )</td>
<td>( 3 )</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>( 4 )</td>
<td>( 2 )</td>
</tr>
</tbody>
</table>

Theorem

How-provenance is sensitive to query rewriting.
Application: Debugging of schema mappings

- Data exchange between two applications (source and target)
- Schema mapping relates data from source application to data from target application
- Schema debuggers help in developing such a mapping

Source-to-target dependencies:

\[ D_1 \text{. foreach } x \text{ in } \text{Manhattan-Credit.Cards} \]
\[ \text{exists } x_1 \text{ in } \text{Fargo-Finance.Clients}, x_1 \text{ in } x_1.\text{AccOwned}, \]
\[ x_1 \text{ in } \text{Fargo-Finance.Accounts} \]
\[ \text{where } x_1.\text{accNo}=x_1.\text{accNo} \]
\[ \text{with } x_1.\text{ssn}=x_1.\text{ssn} \text{ and } x_1.\text{name}=x_1.\text{name} \text{ and } x_1.\text{location}=x_1.\text{address} \]
\[ x_1.\text{cardNo}=x_1.\text{accNo} \text{ and } x_1.\text{limit}=x_1.\text{limit} \]

\[ D_2 \text{. foreach } x \text{ in } \text{SuppCards} \]
\[ \text{exists } x_2 \text{ in } \text{Fargo-Finance.Clients} \]
\[ \text{with } x_2.\text{ssn}=x_2.\text{ssn} \text{ and } x_2.\text{name}=x_2.\text{name} \text{ and } x_2.\text{address}=x_2.\text{address} \]

Target dependency:

\[ C_1 \text{. foreach } x \text{ in } \text{Fargo-Finance.Clients}, x \text{ in } x.\text{AccOwned} \]
\[ \text{exists } x_1 \text{ in } \text{Fargo-Finance.Accounts} \]
\[ \text{with } x_1.\text{accNo}=x_1.\text{accNo} \]
Outline

1 Datalog

2 Introduction to Provenance
   - Lineage
   - Why-provenance
   - How-provenance

3 Provenance Semirings

4 How-Provenance for nr-datalog

5 Summary
Provenance through annotations

Example

<table>
<thead>
<tr>
<th>Name</th>
<th>BasedIn</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>BayTours</td>
<td>SFO</td>
<td>415-1200</td>
</tr>
<tr>
<td>HarborCruz</td>
<td>SC</td>
<td>831-3000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Dest.</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BayTours</td>
<td>SFO</td>
<td>Cable</td>
</tr>
<tr>
<td>BayTours</td>
<td>SC</td>
<td>Bus</td>
</tr>
<tr>
<td>BayTours</td>
<td>SC</td>
<td>Boat</td>
</tr>
<tr>
<td>BayTours</td>
<td>MRY</td>
<td>Boat</td>
</tr>
</tbody>
</table>

\[
\pi_{\text{Dest}, \text{Phone}}(\text{Agencies}) \bowtie \left[ \begin{array}{c}
\pi_{\text{Name}, \text{Dest}}(\rho_{\text{BasedIn} \rightarrow \text{Dest}}(\text{Agencies})) \\
\cup \pi_{\text{Name}, \text{Dest}}(\text{ExternalTours})
\end{array} \right]
\]

<table>
<thead>
<tr>
<th>Dest</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFO</td>
<td>415-1200</td>
</tr>
<tr>
<td>SC</td>
<td>831-3000</td>
</tr>
<tr>
<td>SC</td>
<td>415-1200</td>
</tr>
<tr>
<td>MTY</td>
<td>415-1200</td>
</tr>
</tbody>
</table>

\[
t_1 \cdot (t_1 + t_3) \\
t_2^2 \\
t_1 \cdot (t_4 + t_5) \\
t_1 \cdot t_6
\]

We need a way to annotate relations and propagate these annotations.
**Definition**

A *$K$-relation* is a function $R$ that maps each tuple in the relation to nonzero elements of $K$, and each tuple not in the relation to a special element $0 \in K$. $R$ has finite support $\text{supp}(R) = \{ t \mid R(t) \neq 0 \}$.

Intuitively, each tuple $t$ is *annotated* with an element of $K$.

**Example**

1. $\mathbb{B}$-relations correspond to ordinary relations (zero element: FALSE)
2. $\mathbb{N}$-relations correspond to multisets or bags (zero element: 0)
3. $\mathcal{C}$-relations correspond to boolean c-tables (zero element: FALSE)
4. TupleLoc-relations (zero element: $\bot$)

<table>
<thead>
<tr>
<th>A (1)</th>
<th>A (2)</th>
<th>A (3)</th>
<th>A (4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>BayTours</td>
<td>BayTours</td>
<td>BayTours</td>
<td>BayTours</td>
</tr>
<tr>
<td>HarborCruz</td>
<td>2</td>
<td>HarborCruz</td>
<td>$\bot$</td>
</tr>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>$\times$</td>
<td>A($t_1$)</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>$\neg x$</td>
<td>A($t_2$)</td>
</tr>
</tbody>
</table>
Positive $K$-relational algebra

Definition

Let $(K, 0, 1, +, \cdot)$ be an algebraic structure with two binary operators $+$ (merge) and $\cdot$ (combine) and two distinguished elements $0$ (not in relation) and $1$ (in relation). Let $q^K(I)t$ be the annotation of $t$ in $q(I)$. The operations of the positive $K$-relational algebra are defined as follows:

**Value**

$$(\{ \langle A : a \rangle \})^K(I)t = \begin{cases} 1 & \text{if } t = \langle A : a \rangle \\ 0 & \text{otherwise} \end{cases}$$

**Relation**

$R^K(I)t = I(R)t$

**Selection**

$$(\sigma_\theta(q))^K(I)t = \begin{cases} q^K(I)t & \text{if } \theta(t) \\ 0 & \text{otherwise} \end{cases}$$

**Projection**

$$(\pi_U(q))^K(I)t = \sum_{t' \in \text{supp}(q^K(I)), t'[U]=t} q^K(I)t'$$

**Union**

$$(q_1 \cup q_2)^K(I)t = q^K_1(I)t + q^K_2(I)t$$

**Join**

$$(q_1 \bowtie q_2)^K(I)t = q^K_1(I)t[U_1] \cdot q^K_2(I)t[U_2]$$
Commutative semiring

Relational algebra over bags has the following properties:

- Union (+) is associative and commutative, and has identity $\emptyset$
- Join (·) is associative, commutative, and distributes over union
- Projection and selection commute with each other as well as with union and join

Goal: Retain these properties with positive K-relational algebra.

Definition

$(K, 0, 1, +, \cdot)$ is a commutative semiring if:

- $(K, +, 0)$ is a commutative monoid (associative, commutative, identity 0),
- $(K, \cdot, 1)$ is a commutative monoid (associative, commutative, identity 1),
- $\cdot$ distributes over $+$,
- $0 \cdot a = a \cdot 0 = 0$ for all $a \in K$. 
Common semirings

- **How-provenance**: $(\mathbb{N}[\text{TupleLoc}], 0, 1, +, \cdot)$
  - TupleLoc denotes set of all tuple locations
  - $\mathbb{N}[K] = \text{set of polynomials with coefficients in } \mathbb{N} \text{ and variables from } K$
  - $+ \text{ and } \cdot \text{ have usual definitions}$
  - Start with $R^K(I)t = (R, t)$ if $t \in I(R)$, else 0

Called *positive algebra provenance semiring*.

- **Bag semantics**: $(\mathbb{N}, 0, 1, +, \cdot)$
  - $+ \text{ and } \cdot \text{ have usual definitions}$
  - Start with $R^K(I)t = \text{multiplicity of } t \text{ in } R(I)$

- **Lineage**: $(\mathcal{P}(\text{TupleLoc}) \cup \{\bot\}, \bot, \emptyset, \cup_L, \cup_S)$
  - Lazy union $\cup_L$: $\bot \cup X = X \cup \bot = X$
  - Strict union $\cup_S$: $\bot \cup X = X \cup \bot = \bot$
  - Start with $R^K(I)t = \{(R, t)\}$ if $t \in I(R)$, else $\bot$

- **Minimal why-provenance**: $(\mathcal{P}(\mathcal{P}(\text{TupleLoc})), \emptyset, \{\emptyset\}, \cup_{\text{Min}}, \cup_{\text{Min}})$
  - Min operator computes minimal elements
    (e.g., $\text{Min}\{\{1\}, \{1, 2\}\} = \{\{1\}\}$)
  - Pairwise union: $X \cup_{\text{Min}} Y = \text{Min}\{x \cup y \mid x \in X, y \in Y\}$
  - Start with $R^K(I)t = \{(R, t)\}$ if $t \in I(R)$, else $\bot$
Common semirings (examples)

Example

Query:

\[ q(x, y) \leftarrow R(x, y), R(x, z) \]
\[ q(R) = \pi_{A,B}(R \Join_{\rho_B\rightarrow C} R) \]

<table>
<thead>
<tr>
<th>How-provenance</th>
<th>Bags</th>
<th>Lineage</th>
<th>Min. why-provenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R )</td>
<td>( R )</td>
<td>( R )</td>
<td>( R )</td>
</tr>
<tr>
<td>( \begin{array}{cc} A &amp; B \ 1 &amp; 2 \ 1 &amp; 3 \ 4 &amp; 2 \end{array} )</td>
<td>( \begin{array}{cc} A &amp; B \ 1 &amp; 2 \ 1 &amp; 3 \ 4 &amp; 2 \end{array} )</td>
<td>( \begin{array}{cc} A &amp; B \ 1 &amp; 2 \ 1 &amp; 3 \ 4 &amp; 2 \end{array} )</td>
<td>( \begin{array}{cc} A &amp; B \ 1 &amp; 2 \ 1 &amp; 3 \ 4 &amp; 2 \end{array} )</td>
</tr>
<tr>
<td>( t_1 )</td>
<td>( t_1 )</td>
<td>{ t_1 }</td>
<td>{ { t_1 } }</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( t_2 )</td>
<td>{ t_2 }</td>
<td>{ { t_2 } }</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>( t_3 )</td>
<td>{ t_3 }</td>
<td>{ { t_3 } }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( q(R) )</th>
<th>( q(R) )</th>
<th>( q(R) )</th>
<th>( q(R) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \begin{array}{cc} A &amp; B \ 1 &amp; 2 \ 1 &amp; 3 \ 4 &amp; 2 \end{array} )</td>
<td>( \begin{array}{cc} A &amp; B \ 1 &amp; 2 \ 1 &amp; 3 \ 4 &amp; 2 \end{array} )</td>
<td>( \begin{array}{cc} A &amp; B \ 1 &amp; 2 \ 1 &amp; 3 \ 4 &amp; 2 \end{array} )</td>
<td>( \begin{array}{cc} A &amp; B \ 1 &amp; 2 \ 1 &amp; 3 \ 4 &amp; 2 \end{array} )</td>
</tr>
<tr>
<td>( t_1^2 + t_1 \cdot t_2 )</td>
<td>( t_2^2 + t_1 \cdot t_2 )</td>
<td>{ t_1, t_2 }</td>
<td>{ { t_1 } }</td>
</tr>
<tr>
<td>( 10 )</td>
<td>( 15 )</td>
<td>{ t_1, t_2 }</td>
<td>{ { t_2 } }</td>
</tr>
<tr>
<td>( 4 )</td>
<td>( 2 )</td>
<td>{ t_3 }</td>
<td>{ { t_3 } }</td>
</tr>
</tbody>
</table>
Outline

1 Datalog

2 Introduction to Provenance
   - Lineage
   - Why-provenance
   - How-provenance

3 Provenance Semirings

4 How-Provenance for nr-datalog

5 Summary
Proof tree

Proof-theoretic semantics of datalog: A fact is in the result if there exists a proof for it using the rules and the database facts.

Definition

A proof tree of a fact $A$ is a labeled tree where:

- Each vertex of the tree is labeled by a fact.
- Each leaf is labeled by an EDB fact from the base data.
- The root is labeled by $A$.
- For each internal vertex, there exists an instantiation $A_1 \leftarrow A_2, \ldots, A_n$ of a rule $r$ such that the vertex is labeled $A_1$, its children are respectively labeled $A_2, \ldots, A_n$ and the edges are labeled $r$. 
Example

Proof tree (example)

\( r_1 : \) ExpensiveRoom\((r, h) \) \( \leftarrow \) Room\((r, h, p), p > \$500 \)
\( r_2 : \) ExpensiveRoom\((r, h) \) \( \leftarrow \) Room\((r, h, t), t = 'suite' \)
\( r_3 : \) ExpensiveHotelRoom\((h, r) \) \( \leftarrow \) Hotel\((h, r), \) ExpensiveRoom\((r, h) \)
\( r_4 : \) ExpensiveHotel\((h) \) \( \leftarrow \) ExpensiveHotelRoom\((h, r) \)

### Room (R)

<table>
<thead>
<tr>
<th>RoomNo</th>
<th>Type</th>
<th>HotelNo</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Suite</td>
<td>H1</td>
<td>$50</td>
</tr>
<tr>
<td>R2</td>
<td>Single</td>
<td>H1</td>
<td>$600</td>
</tr>
<tr>
<td>R3</td>
<td>Double</td>
<td>H1</td>
<td>$80</td>
</tr>
</tbody>
</table>

### Hotel (H)

<table>
<thead>
<tr>
<th>HotelNo</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>Hilton</td>
<td>SB</td>
</tr>
</tbody>
</table>

Multiple different proof trees may exist!
Lineage tree

Goal: Capture all ways of deriving an output fact.

Definition

A \textit{lineage tree} of an nr-datalog query is computed with respect to the semiring \((\text{PosBool}(\mathcal{V}), \text{FALSE}, \text{TRUE}, \lor, \land)\), where

- \(\mathcal{V}\) is a countable set of boolean variables,
- \text{PosBool}(\mathcal{V}) is the set of sets of equivalent boolean expressions involving \text{TRUE}, \text{FALSE}, variables from \(\mathcal{V}\), \lor, and \land,
- Each fact is tagged with a representative from its class in \text{PosBool}(\mathcal{V}),
- Each EDB fact is tagged with a distinct variable from \(\mathcal{V}\).

Example

\[
\text{PosBool}({t_1, t_2}) = \{ \{ \text{FALSE} \}, \{ \text{TRUE} \} \\
\{ t_1, t_1 \lor t_1, t_1 \land \text{TRUE}, \ldots \} \\
\{ t_1 \lor t_2, \ldots \}, \{ t_1 \land t_2, \ldots \} \}
\]
Lineage tree (example)

\[ \pi_{\text{HotelNo}}(\pi_{\text{HotelNo}, \text{RoomNo}}(\text{Hotel} \times \pi_{\text{RoomNo}, \text{HotelNo}}(\sigma_{\text{price} > 500 \lor \text{type} = \text{'suite'}(\text{Room}))))) \]

**Room (R)**

<table>
<thead>
<tr>
<th>RoomNo</th>
<th>Type</th>
<th>HotelNo</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Suite</td>
<td>H1</td>
<td>$50</td>
</tr>
<tr>
<td>R2</td>
<td>Single</td>
<td>H1</td>
<td>$600</td>
</tr>
<tr>
<td>R3</td>
<td>Double</td>
<td>H1</td>
<td>$80</td>
</tr>
</tbody>
</table>

**Hotel (H)**

<table>
<thead>
<tr>
<th>HotelNo</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>Hilton</td>
<td>SB</td>
</tr>
</tbody>
</table>

**ExpensiveHotels**

<table>
<thead>
<tr>
<th>HotelNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
</tr>
</tbody>
</table>

\[ t_4 \land (t_1 \lor t_2) \]

Not unique. There are many different trees, but all of them belong to the same PosBool equivalence class.
Outline

1 Datalog

2 Introduction to Provenance
   - Lineage
   - Why-provenance
   - How-provenance

3 Provenance Semirings

4 How-Provenance for nr-datalog

5 Summary
Lessons learned

- **Datalog** is a declarative language for relations
  - Based on Prolog
  - Collection of if-then rules
  - Closely related to relational algebra

- **Provenance** describes origins and history of data;
  *Annotation management* allows and propagates data annotations
  - Data warehousing, curated databases, annotated databases, update languages, *uncertain databases*, . . .

- Different types of provenance provide different amount of detail
  1. **Lineage**: *what* contributed to the output (tuples)
  2. **Why-provenance**: *why* an output tuple was produced (db instances)
  3. **How-provenance**: *how* an output tuple was produced (polynomial)

- Semirings are a natural way to study provenance

- Positive $K$-relational algebra can compute many forms of provenance

- Lineage trees are the preferred form of how-provenance for datalog (boolean formula)
Suggested reading

- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
  *Database Systems: The Complete Book*, 2nd ed. (ch. 5.3 & 5.4)
  Pearson Prentice Hall, 2009

- Serge Abiteboul, Richard Hull, Victor Vianu
  *Foundations of Databases: The Logical Level* (ch. 12)
  Addison Wesley, 1994

- James Cheney, Laura Chiticariu, Wang-Chiew Tan
  *Provenance in Databases: Why, How, and Where*
  Foundations and Trends in Databases, 1(4), 2007