Provenance in Databases

... and Links to Knowledge Compilation

Antoine Amarilli
December 17, 2019
Kocoon Workshop
Provenance management

- Common task on databases: **query evaluation**
Provenance management

- Common task on databases: query evaluation
- What if we want more than the result?
  - Where does the result come from?
  - Why was this result obtained?
  - How was the result produced?
  - What is the probability of the result?
  - How many times was the result obtained?
  - How would the result change if some data was missing?
  - What is the minimal security clearance I need to see the result?
  - How can a result be explained to the user?
Provenance management

- Common task on databases: **query evaluation**
- What if we want **more** than the result?
  - *Where* does the result come from?
  - *Why* was this result obtained?
  - *How* was the result produced?
  - What is the **probability** of the result?
  - How many **times** was the result obtained?
  - How would the result change if some data was **missing**?
  - What is the minimal **security clearance** I need to see the result?
  - How can a result be **explained** to the user?

- Provenance management: extend query evaluation with **provenance information** to answer these questions
- Provenance information often representable as a **circuit**
Goal of this talk

- Refresher on relational databases, and provenance for them (standard in database theory)
- Primer on query evaluation (MSO/automata) on words and trees (standard in database theory and logics)
- Present a notion of provenance for queries on trees (less standard, but nice connections to knowledge compilation)
- Present applications to probabilities and enumeration for relational data and trees
Goal of this talk

• Refresher on relational databases, and provenance for them (standard in database theory)
• Primer on query evaluation (MSO/automata) on words and trees (standard in database theory and logics)
• Present a notion of provenance for queries on trees (less standard, but nice connections to knowledge compilation)
• Present applications to probabilities and enumeration for relational data and trees

My co-authors for results in this talk (and some of the slides):
Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion
Relational DBMSs

- **Relational model**: express data as relations (i.e., tables)
- A standard query language: **SQL**
### Guest

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@gmail.com">john.smith@gmail.com</a></td>
</tr>
<tr>
<td>2</td>
<td>Alice Black</td>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
</tr>
<tr>
<td>3</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@ens.fr">john.smith@ens.fr</a></td>
</tr>
</tbody>
</table>

### Reservation

<table>
<thead>
<tr>
<th>id</th>
<th>guest</th>
<th>room</th>
<th>arrival</th>
<th>nights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>504</td>
<td>2019-01-01</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>107</td>
<td>2019-01-10</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>302</td>
<td>2019-01-15</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>504</td>
<td>2019-01-15</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>107</td>
<td>2019-01-30</td>
<td>1</td>
</tr>
</tbody>
</table>
Relations and databases

Formally:

• A relation schema $\mathcal{R}$ is a finite sequence of attribute names
• A database schema $\mathcal{D}$ maps each relation name to a relation schema
• A tuple over relation schema $\mathcal{R}$ maps each attribute name of $\mathcal{R}$ to a data value
• A relation instance over $\mathcal{R}$ is a finite set of tuples over $\mathcal{R}$
• A database over database schema $\mathcal{D}$ maps each relation name $R$ of $\mathcal{D}$ to a relation instance over the relation schema of $R$ in $\mathcal{D}$
The positive relational algebra

- **Algebraic language** to express queries
- Each **operator** applies to 0, 1, or 2 **subexpressions** and produces a **relation instance**
- Main operators:
  - \( R \): relation name
  - \( \rho_{a \rightarrow b} \): rename attribute \( a \) to \( b \)
  - \( \Pi_{a_1, \ldots, a_n} \): project on attributes \( a_1, \ldots, a_n \)
  - \( \sigma_{\varphi} \): select all tuples satisfying condition \( \varphi \)
  - \( \cup \): union of two relations (with same relation schema)
  - \( \times \): cross product of two relations
Relation name

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@gmail.com">john.smith@gmail.com</a></td>
</tr>
<tr>
<td>2</td>
<td>Alice Black</td>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
</tr>
<tr>
<td>3</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@ens.fr">john.smith@ens.fr</a></td>
</tr>
</tbody>
</table>

Reservation

<table>
<thead>
<tr>
<th>id</th>
<th>guest</th>
<th>room</th>
<th>arrival</th>
<th>nights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>504</td>
<td>2019-01-01</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>107</td>
<td>2019-01-10</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>302</td>
<td>2019-01-15</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>504</td>
<td>2019-01-15</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>107</td>
<td>2019-01-30</td>
<td>1</td>
</tr>
</tbody>
</table>

Expression: Guest

Result:

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@gmail.com">john.smith@gmail.com</a></td>
</tr>
<tr>
<td>2</td>
<td>Alice Black</td>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
</tr>
<tr>
<td>3</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@ens.fr">john.smith@ens.fr</a></td>
</tr>
</tbody>
</table>
### Renaming

#### Expression:

\[ \rho_{id \rightarrow \text{guest}}(\text{Guest}) \]

#### Result:

<table>
<thead>
<tr>
<th>guest</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@gmail.com">john.smith@gmail.com</a></td>
</tr>
<tr>
<td>2</td>
<td>Alice Black</td>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
</tr>
<tr>
<td>3</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@ens.fr">john.smith@ens.fr</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Guest</th>
<th>id</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@gmail.com">john.smith@gmail.com</a></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Alice Black</td>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@ens.fr">john.smith@ens.fr</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reservation</th>
<th>id</th>
<th>guest</th>
<th>room</th>
<th>arrival</th>
<th>nights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>504</td>
<td>2019-01-01</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>107</td>
<td>2019-01-10</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>302</td>
<td>2019-01-15</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>504</td>
<td>2019-01-15</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>107</td>
<td>2019-01-30</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Projection

Expression: $\Pi_{email, id}(Guest)$

Result:

<table>
<thead>
<tr>
<th>email</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:john.smith@gmail.com">john.smith@gmail.com</a></td>
<td>1</td>
</tr>
<tr>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
<td>2</td>
</tr>
<tr>
<td><a href="mailto:john.smith@ens.fr">john.smith@ens.fr</a></td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Guest</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reservation</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
The formula used in the selection can be any **Boolean combination** of comparisons of attributes to attributes or constants.
Cross product

<table>
<thead>
<tr>
<th>Guest</th>
<th>Reservation</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>John Smith</td>
</tr>
<tr>
<td>2</td>
<td>Alice Black</td>
</tr>
<tr>
<td>3</td>
<td>John Smith</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Expression: $\Pi_{id}(Guest) \times \Pi_{name}(Guest)$

Result:

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice Black</td>
</tr>
<tr>
<td>2</td>
<td>Alice Black</td>
</tr>
<tr>
<td>3</td>
<td>Alice Black</td>
</tr>
<tr>
<td>1</td>
<td>John Smith</td>
</tr>
<tr>
<td>2</td>
<td>John Smith</td>
</tr>
<tr>
<td>3</td>
<td>John Smith</td>
</tr>
</tbody>
</table>
Not a basic operator, but a **useful shorthand!**

**Expression:** \( \text{Reservation} \bowtie \rho_{id \rightarrow \text{guest}}(\text{Guest}) \)

**Result:**

<table>
<thead>
<tr>
<th>id</th>
<th>guest</th>
<th>room</th>
<th>arrival</th>
<th>nights</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>504</td>
<td>2019-01-01</td>
<td>5</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@gmail.com">john.smith@gmail.com</a></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>107</td>
<td>2019-01-10</td>
<td>3</td>
<td>Alice Black</td>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>302</td>
<td>2019-01-15</td>
<td>6</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@ens.fr">john.smith@ens.fr</a></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>504</td>
<td>2019-01-15</td>
<td>2</td>
<td>Alice Black</td>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>107</td>
<td>2019-01-30</td>
<td>1</td>
<td>Alice Black</td>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
</tr>
</tbody>
</table>

**Equivalent to:**

\[
\Pi_{id, guest, room, arrival, nights, name, email}(\sigma_{\text{temp} = \text{guest}}(\rho_{id \rightarrow \text{temp}}(\text{Guest}) \times \text{Reservation})).
\]
Expression: \[ \Pi_{\text{room}}(\sigma_{\text{guest}=2}(\text{Reservation})) \cup \Pi_{\text{room}}(\sigma_{\text{arrival}=2019-01-15}(\text{Reservation})) \]

Result:

\[ \text{room} \]

\[ 107 \]

\[ 302 \]

\[ 504 \]
Sometimes we write tuples as **ground facts** rather than tables

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@gmail.com">john.smith@gmail.com</a></td>
</tr>
<tr>
<td>2</td>
<td>Alice Black</td>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
</tr>
<tr>
<td>3</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@ens.fr">john.smith@ens.fr</a></td>
</tr>
</tbody>
</table>

Guest(1, John Smith, john.smith@gmail.com),
Guest(2, Alice Black, alice@black.name),
Guest(3, John Smith, john.smith@ens.fr)
Relational algebra vs relational calculus

Sometimes we write tuples as **ground facts** rather than tables

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@gmail.com">john.smith@gmail.com</a></td>
</tr>
<tr>
<td>2</td>
<td>Alice Black</td>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
</tr>
<tr>
<td>3</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@ens.fr">john.smith@ens.fr</a></td>
</tr>
</tbody>
</table>

Guest(1, John Smith, john.smith@gmail.com),
Guest(2, Alice Black, alice@black.name),
Guest(3, John Smith, john.smith@ens.fr)

Sometimes we write queries in **relational calculus** rather than algebra

\[ \Pi_{id}(Guest) \times \Pi_{name}(Guest) \]

\[ Q(x, y') : \exists y z x' z' \ Guest(x, y, z) \land Guest(x', y', z') \]
Relational algebra vs relational calculus

Sometimes we write tuples as **ground facts** rather than tables

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@gmail.com">john.smith@gmail.com</a></td>
</tr>
<tr>
<td>2</td>
<td>Alice Black</td>
<td><a href="mailto:alice@black.name">alice@black.name</a></td>
</tr>
<tr>
<td>3</td>
<td>John Smith</td>
<td><a href="mailto:john.smith@ens.fr">john.smith@ens.fr</a></td>
</tr>
</tbody>
</table>

```
Guest(1, John Smith, john.smith@gmail.com),
Guest(2, Alice Black, alice@black.name),
Guest(3, John Smith, john.smith@ens.fr)
```

Sometimes we write queries in **relational calculus** rather than algebra

\[ \Pi_{id}(Guest) \times \Pi_{name}(Guest) \]

\[ Q(x, y') : \exists y z x' z' \text{Guest}(x, y, z) \land \text{Guest}(x', y', z') \]

\[ \rightarrow \text{Relational algebra and calculus have the same expressive power!} \]
Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion
• **Relational data model**: data decomposed into relations, with labeled attributes...
Data model

- **Relational data model**: data decomposed into relations, with labeled attributes...

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
</tr>
</tbody>
</table>
• **Relational data model**: data decomposed into relations, with labeled attributes…

• … with an extra **provenance annotation** for each tuple (think of it as a Boolean variable)

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>x₁</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>x₂</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>x₃</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>x₄</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>x₅</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>x₆</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>x₇</td>
</tr>
</tbody>
</table>
Boolean valuations

- Database $D$ with $n$ tuples
- $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ the **Boolean variables** annotating the tuples
- **Valuation** over $\mathcal{X}$: function $\nu : \mathcal{X} \rightarrow \{\perp, \top\}$
- **Possible world** $\nu(D)$: the subset of $D$ where we keep precisely the tuples whose annotation evaluates to $\top$
Example of possible worlds

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>$x_1$</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>$x_2$</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>$x_3$</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_4$</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>$x_5$</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>$x_6$</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_7$</td>
</tr>
</tbody>
</table>

$\nu: \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{bmatrix} = \begin{bmatrix} T & T & T & T & T & T & T \end{bmatrix}$
### Example of possible worlds

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>$x_1$</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>$x_3$</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>$x_5$</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_7$</td>
</tr>
</tbody>
</table>

$\nu$: 

<table>
<thead>
<tr>
<th></th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
<th>$X_5$</th>
<th>$X_6$</th>
<th>$X_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\top$</td>
<td>$T$</td>
<td>$\bot$</td>
<td>$T$</td>
<td>$\bot$</td>
<td>$T$</td>
<td>$\bot$</td>
<td>$T$</td>
</tr>
</tbody>
</table>
Boolean provenance of query results

- **Goal:** Evaluate a positive relational algebra query \( Q \) on a database \( D \)...
Goal: Evaluate a positive relational algebra query \( Q \) on a database \( D \) ... whose tuples are annotated with \( \mathcal{X} = x_1, \ldots, x_n \)
Boolean provenance of query results

- **Goal:** Evaluate a **positive relational algebra query** $Q$
on a database $D$... whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$
- The result is a **relation instance** $R$...
Boolean provenance of query results

- **Goal:** Evaluate a positive relational algebra query $Q$ on a database $D$... whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$
- The result is a **relation instance** $R$... where each tuple is annotated with a **Boolean function** on $\mathcal{X}$

Example (What cities are in the table?)

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
</tr>
</tbody>
</table>

Claim: we can compute this provenance while evaluating the query!
Boolean provenance of query results

- **Goal:** Evaluate a positive relational algebra query $Q$ on a database $D$... whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$
- The result is a relation instance $R$... where each tuple is annotated with a Boolean function on $\mathcal{X}$
- **Semantics:** For every tuple $t$ of the result, for every valuation $\nu$ of $\mathcal{X}$, the annotation of $t$ evaluates to true on $\nu$ iff $t \in Q(\nu(D))$
Boolean provenance of query results

- **Goal:** Evaluate a positive relational algebra query $Q$ on a database $D$... whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$
- The result is a relation instance $R$... where each tuple is annotated with a Boolean function on $\mathcal{X}$
- **Semantics:** For every tuple $t$ of the result, for every valuation $\nu$ of $\mathcal{X}$, the annotation of $t$ evaluates to true on $\nu$ iff $t \in Q(\nu(D))$

**Example (What cities are in the table?)**

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>$x_1$</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>$x_2$</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>$x_3$</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_4$</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>$x_5$</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>$x_6$</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_7$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>$x_1 \lor x_2$</td>
</tr>
<tr>
<td>Paris</td>
<td>$x_3 \lor x_5 \lor x_6$</td>
</tr>
<tr>
<td>Berlin</td>
<td>$x_4 \lor x_7$</td>
</tr>
</tbody>
</table>
Boolean provenance of query results

- **Goal:** Evaluate a positive relational algebra query $Q$ on a database $D$... whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$
- The result is a relation instance $R$... where each tuple is annotated with a Boolean function on $\mathcal{X}$
- **Semantics:** For every tuple $t$ of the result, for every valuation $\nu$ of $\mathcal{X}$, the annotation of $t$ evaluates to true on $\nu$ iff $t \in Q(\nu(D))$

**Example (What cities are in the table?)**

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>$x_1$</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>$x_2$</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>$x_3$</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_4$</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>$x_5$</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>$x_6$</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_7$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>$x_1 \lor x_2$</td>
</tr>
<tr>
<td>Paris</td>
<td>$x_3 \lor x_5 \lor x_6$</td>
</tr>
<tr>
<td>Berlin</td>
<td>$x_4 \lor x_7$</td>
</tr>
</tbody>
</table>

**Claim:** we can compute this provenance while evaluating the query!
Provenance annotations of selected tuples are **unchanged**

**Example** \((\rho_{\text{name}} \rightarrow n(\sigma_{\text{city}=\text{“New York”}}(R)))\)

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>(x_1)</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>(x_2)</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>(x_3)</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>(x_4)</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>(x_5)</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>(x_6)</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>(x_7)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>(x_1)</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>(x_2)</td>
</tr>
</tbody>
</table>
Projection

Take the OR of provenance annotations of identical, merged tuples

Example ($\pi_{\text{city}}(R)$)

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>$x_1$</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>$x_2$</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>$x_3$</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_4$</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>$x_5$</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>$x_6$</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_7$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>$x_1 \lor x_2$</td>
</tr>
<tr>
<td>Paris</td>
<td>$x_3 \lor x_5 \lor x_6$</td>
</tr>
<tr>
<td>Berlin</td>
<td>$x_4 \lor x_7$</td>
</tr>
</tbody>
</table>
Take the OR of provenance annotations of identical, merged tuples

Example

$$\pi_{\text{city}}(\sigma_{\text{ends-with}(\text{position}, \text{"agent"})}(R)) \cup \pi_{\text{city}}(\sigma_{\text{position}=\text{"Analyst"}}(R))$$

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>$x_1$</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>$x_2$</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>$x_3$</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_4$</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>$x_5$</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>$x_6$</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_7$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>$x_3 \lor x_5$</td>
</tr>
<tr>
<td>Berlin</td>
<td>$x_4 \lor x_7$</td>
</tr>
</tbody>
</table>
Cross product

Take the AND of provenance annotations of combined tuples

**Example**

\[ \pi_{\text{city}}(\sigma_{\text{ends-with}(\text{position}, \text{“agent”})}(R)) \Join \pi_{\text{city}}(\sigma_{\text{position} = \text{“Analyst”}}(R)) \]

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>x₁</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>x₂</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>x₃</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>x₄</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>x₅</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>x₆</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>x₇</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>( x_3 \land x_5 )</td>
</tr>
<tr>
<td>Berlin</td>
<td>( x_4 \land x_7 )</td>
</tr>
</tbody>
</table>
How is provenance actually represented?

Provenance annotations are **Boolean functions**

- The simplest representation is **Boolean formulas**
- Formalism used in most of the provenance literature

**Example**

Is there a city with two different agents?

\[(x_1 \land x_2) \lor (x_3 \land x_6) \lor (x_3 \land x_5) \lor (x_4 \land x_7) \lor (x_5 \land x_6)\]

**Theorem (PTIME overhead)**

For any fixed **positive relational algebra** expression, given an input database, we can compute in PTIME the provenance annotation of every tuple in the result.
Other representation: Provenance circuits
[Deutch, Milo, Roy, and Tannen 2014]

- Use **Boolean circuits** to represent provenance
- Every time an operation reuses a previously computed result, link to the *previously created circuit gate*
- **Never larger** than provenance formulas
- Sometimes **more concise**: provenance circuits can be...
Other representation: Provenance circuits
[Deutch, Milo, Roy, and Tannen 2014]

- Use **Boolean circuits** to represent provenance
- Every time an operation reuses a previously computed result, link to the **previously created circuit gate**
- **Never larger** than provenance formulas
- Sometimes **more concise**: provenance circuits can be...
  - More concise by a $\log \log$ factor than provenance formulas for positive relational algebra [Amarilli, Bourhis, and Senellart 2016]
  - More concise by a $\log$ factor than **monotone** provenance formulas for positive relational algebra
  - Super-polynomially more concise for more expressive query languages [Deutch, Milo, Roy, and Tannen 2014]
Example provenance circuit

\[
\begin{align*}
X_1 & \quad \lor \quad X_2 \\
\quad & \quad \lor \quad X_3 \\
\quad & \quad \lor \quad X_5 \\
\quad & \quad \lor \quad X_6 \\
\quad & \quad \lor \quad X_4 \\
\quad & \quad \lor \quad X_7
\end{align*}
\]
What can we do with Boolean provenance?

\[(x_1 \land x_2) \lor (x_3 \land x_6) \lor (x_3 \land x_5) \lor (x_4 \land x_7) \lor (x_5 \land x_6)\]

- The provenance describes, for each result tuple, the subsets of the input database for which it appears in the query result.
What can we do with Boolean provenance?

\[(x_1 \land x_2) \lor (x_3 \land x_6) \lor (x_3 \land x_5) \lor (x_4 \land x_7) \lor (x_5 \land x_6)\]

- The provenance describes, for each result tuple, the **subsets** of the input database for which it appears in the query result
- **SAT**: test if the tuple can be an answer when we delete some input tuples (trivial here)
What can we do with Boolean provenance?

\[(x_1 \land x_2) \lor (x_3 \land x_6) \lor (x_3 \land x_5) \lor (x_4 \land x_7) \lor (x_5 \land x_6)\]

- The provenance describes, for each result tuple, the **subsets** of the input database for which it appears in the query result
- **SAT**: test if the tuple can be an answer when we delete some input tuples (trivial here)
- **#SAT**: number of sub-databases where the tuple is a result
  \[\rightarrow\] Useful for **probabilistic reasoning** (see later)
What can we do with Boolean provenance?

\[(x_1 \land x_2) \lor (x_3 \land x_6) \lor (x_3 \land x_5) \lor (x_4 \land x_7) \lor (x_5 \land x_6)\]

- The provenance describes, for each result tuple, the **subsets** of the input database for which it appears in the query result.
- **SAT**: test if the tuple can be an answer when we delete some input tuples (trivial here)
- **#SAT**: number of sub-databases where the tuple is a result
  → Useful for *probabilistic reasoning* (see later)
- **Enumerating models**: enumerating sub-databases where the tuple is a result
  → Useful to enumerate query results (see later)
Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion
Commutative semiring \((K, 0, 1, \oplus, \otimes)\)

- Set \(K\) with distinguished elements \(0, 1\)
- \(\oplus\) associative, commutative operator, with identity \(0_K\):
  - \(a \oplus (b \oplus c) = (a \oplus b) \oplus c\)
  - \(a \oplus b = b \oplus a\)
  - \(a \oplus 0 = 0 \oplus a = a\)
- \(\otimes\) associative, commutative operator, with identity \(1_K\):
  - \(a \otimes (b \otimes c) = (a \otimes b) \otimes c\)
  - \(a \otimes b = b \otimes a\)
  - \(a \otimes 1 = 1 \otimes a = a\)
- \(\otimes\) distributes over \(\oplus\):
  \[
  a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)
  \]
- \(0\) is annihilating for \(\otimes\):
  \[
  a \otimes 0 = 0 \otimes a = 0
  \]
Example semirings

- \((\mathbb{N}, 0, 1, +, \times)\): counting semiring
- \((\{\bot, \top\}, \bot, \top, \lor, \land)\): Boolean semiring
- \((\{\text{unclassified}, \text{restricted}, \text{confidential}, \text{secret}, \text{top secret}\}, \text{top secret}, \text{unclassified}, \min, \max)\): security semiring
- \((\mathbb{N} \cup \{\infty\}, \infty, 0, \min, +)\): tropical semiring
- \((\{\text{Boolean functions over } \mathcal{X}\}, \bot, \top, \lor, \land)\): semiring of Boolean functions over \(\mathcal{X}\)
- \((\mathbb{N}[\mathcal{X}], 0, 1, +, \times)\): semiring of integer-valued polynomials with variables in \(\mathcal{X}\) (also called How-semiring or universal semiring)
We fix a semiring \((K, 0, 1, \oplus, \otimes)\)

We assume provenance annotations are in \(K\)

We consider a query \(Q\) from the positive relational algebra (selection, projection, renaming, product, union)

We define a semantics for the provenance of a tuple \(t \in Q(D)\) inductively on the structure of \(Q\) just like before
Provenance annotations of selected tuples are **unchanged**

**Example** \( \rho_{\text{name} \rightarrow n}(\sigma_{\text{city} = \text{“New York”}}(R)) \)

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>( x_1 )</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>( x_2 )</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>( x_3 )</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>( x_4 )</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>( x_5 )</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>( x_6 )</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>( x_7 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>( x_1 )</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>( x_2 )</td>
</tr>
</tbody>
</table>
Projection

Provenance annotations of identical, merged, tuples are ⊕-ed

Example $(\pi_{city}(R))$
Provenance annotations of identical, merged, tuples are $\oplus$-ed

**Example**

$$\pi_{\text{city}}(\sigma_{\text{ends-with}(\text{position, \text{“agent”})}}(R)) \cup \pi_{\text{city}}(\sigma_{\text{position=\text{“Analyst”}}}(R))$$

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>$x_1$</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>$x_2$</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>$x_3$</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_4$</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>$x_5$</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>$x_6$</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_7$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>$x_3 \oplus x_5$</td>
</tr>
<tr>
<td>Berlin</td>
<td>$x_4 \oplus x_7$</td>
</tr>
</tbody>
</table>
Provenance annotations of combined tuples are $\otimes$-ed

**Example**

$$\pi_{\text{city}}(\sigma_{\text{ends-with}(\text{position,"agent")}(R)) \otimes \pi_{\text{city}}(\sigma_{\text{position="Analyst"}}(R))$$

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>$x_1$</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>$x_2$</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>$x_3$</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_4$</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>$x_5$</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>$x_6$</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_7$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>$x_3 \otimes x_5$</td>
</tr>
<tr>
<td>Berlin</td>
<td>$x_4 \otimes x_7$</td>
</tr>
</tbody>
</table>
What can we do with semiring provenance?

counting semiring: count the number of times a tuple can be derived, multiset semantics

Boolean semiring: determines if a tuple exists when a subdatabase is selected

security semiring: determines the minimum clearance level required to get a tuple as a result

tropical semiring: minimum-weight way of deriving a tuple (think shortest path in a graph)

Boolean functions: Boolean provenance, as previously defined

integer polynomials: $\mathbb{N}[X]$, universal provenance, see further
Example of security provenance

\[
\pi_{\text{city}}(\sigma_{\text{name}<\text{name2}}(\pi_{\text{name,city}}(R) \Join \rho_{\text{name}\rightarrow\text{name2}}(\pi_{\text{name,city}}(R)))))
\]

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>restricted</td>
</tr>
<tr>
<td>Paris</td>
<td>confidential</td>
</tr>
<tr>
<td>Berlin</td>
<td>secret</td>
</tr>
</tbody>
</table>
Properties [Green, Karvounarakis, and Tannen 2007]

- Semiring provenance still has PTIME data overhead
Properties [Green, Karvounarakis, and Tannen 2007]

- Semiring provenance still has PTIME data overhead
- Semiring homomorphisms commute with provenance computation: if \( K \xrightarrow{\text{hom}} K' \), then one can compute the provenance in \( K \), apply the homomorphism, and obtain the same result as when computing provenance in \( K' \)
Properties [Green, Karvounarakis, and Tannen 2007]

- Semiring provenance still has PTIME data overhead
- Semiring homomorphisms commute with provenance computation: if $K \xrightarrow{\text{hom}} K'$, then one can compute the provenance in $K$, apply the homomorphism, and obtain the same result as when computing provenance in $K'$
- The integer polynomial semiring $\mathbb{N}[X]$ is universal: there is a unique homomorphism to any other commutative semiring that respects a given valuation of the variables
Properties [Green, Karvounarakis, and Tannen 2007]

- Semiring provenance still has **PTIME** data overhead
- Semiring homomorphisms **commute** with provenance computation: if $K \xrightarrow{\text{hom}} K'$, then one can compute the provenance in $K$, apply the homomorphism, and obtain the same result as when computing provenance in $K'$
- The integer polynomial semiring $\mathbb{N}[X]$ is **universal**: there is a unique homomorphism to any other commutative semiring that respects a given valuation of the variables
- This means all computations can be performed in the universal **semiring**, and homomorphisms applied next
• Semiring provenance still has \textbf{PTIME} data overhead

• Semiring homomorphisms \textbf{commute} with provenance computation: if \( K \xrightarrow{\text{hom}} K' \), then one can compute the provenance in \( K \), apply the homomorphism, and obtain the same result as when computing provenance in \( K' \)

• The integer polynomial semiring \( \mathbb{N}[X] \) is \textbf{universal}: there is a unique homomorphism to any other commutative semiring that respects a given valuation of the variables

• This means \textbf{all computations can be performed in the universal semiring}, and homomorphisms applied next

• Two \textbf{equivalent queries} can have two \textbf{different provenance annotations} on the same database, in some semirings
Extensions

• Beyond positive relational algebra...
  • Allow relational difference: need a semiring with monus, but complicated semantics [Amer 1984; Geerts and Poggi 2010; Amsterdamer, Deutch, and Tannen 2011a; Amarilli and Monet 2016]
Extensions

• Beyond positive relational algebra...
  • Allow **relational difference**: need a semiring with **monus**, but complicated semantics [Amer 1984; Geerts and Poggi 2010; Amsterdamer, Deutch, and Tannen 2011a; Amarilli and Monet 2016]
  • Allow **aggregate queries**: extend semirings to **semimodules** [Amsterdamer, Deutch, and Tannen 2011b; Fink, Han, and Olteanu 2012]
Extensions

• Beyond positive relational algebra...
  • Allow relational difference: need a semiring with monus, but complicated semantics [Amer 1984; Geerts and Poggi 2010; Amsterdamer, Deutch, and Tannen 2011a; Amarilli and Monet 2016]
  • Allow aggregate queries: extend semirings to semimodules [Amsterdamer, Deutch, and Tannen 2011b; Fink, Han, and Olteanu 2012]
  • Allow recursive queries: representation as formal power series or cycluits [Amarilli, Bourhis, Monet, and Senellart 2017]
• Beyond positive relational algebra...
  • Allow **relational difference**: need a semiring with **monus**, but complicated semantics [Amer 1984; Geerts and Poggi 2010; Amsterdamer, Deutch, and Tannen 2011a; Amarilli and Monet 2016]
  • Allow **aggregate queries**: extend semirings to **semimodules** [Amsterdamer, Deutch, and Tannen 2011b; Fink, Han, and Olteanu 2012]
  • Allow **recursive queries**: representation as **formal power series** or **cycluits** [Amarilli, Bourhis, Monet, and Senellart 2017]

• Beyond semiring provenance...
  • **Where-provenance**: capture which output **value** comes from which input **value** [Buneman, Khanna, and Tan 2001]
Extensions

• Beyond positive relational algebra...
  • Allow relational difference: need a semiring with monus, but complicated semantics [Amer 1984; Geerts and Poggi 2010; Amsterdamer, Deutch, and Tannen 2011a; Amarilli and Monet 2016]
  • Allow aggregate queries: extend semirings to semimodules [Amsterdamer, Deutch, and Tannen 2011b; Fink, Han, and Olteanu 2012]
  • Allow recursive queries: representation as formal power series or cycluits [Amarilli, Bourhis, Monet, and Senellart 2017]

• Beyond semiring provenance...
  • Where-provenance: capture which output value comes from which input value [Buneman, Khanna, and Tan 2001]
  • Why-not provenance: capture why an output tuple was not produced, usually as a function of the query [Chapman and Jagadish 2009]
Motivation and definition

- We now move to a **different setting** for query evaluation
- We will later define **provenance** for this setting

Assume our data is a sequence of events:

- A formal model: a graph where each node has a color
- We could represent this in the relational setting:
  - One/two-ary table for the successor relation
  - One/three-ary table to list the nodes for each color

Some natural queries cannot be expressed in relational algebra!

→ "Is there a blue node after each pink node?"
Motivation and definition

- We now move to a **different setting** for query evaluation
- We will later define **provenance** for this setting

Assume our data is a sequence of events:

- Formal model: a **word** where each node has a **color**
Motivation and definition

- We now move to a **different setting** for query evaluation
- We will later define **provenance** for this setting

Assume our data is a sequence of events:

![Diagram of nodes](image)

- Formal model: a **word** where each node has a **color**
- We could represent this in the relational setting:
  - One 2-ary table for the **successor relation**
  - One 1-ary table to list the nodes for each **color**
Motivation and definition

• We now move to a different setting for query evaluation
• We will later define provenance for this setting

Assume our data is a sequence of events:

- Formal model: a word where each node has a color
- We could represent this in the relational setting:
  - One 2-ary table for the successor relation
  - One 1-ary table to list the nodes for each color
- Some natural queries cannot be expressed in relational algebra!
  → “Is there a blue node after each pink node?”
Query evaluation on words

**Database:** a word \( w \) where nodes have a color from an alphabet

Result: YES/NO indicating if the word \( w \) satisfies the query

Note that we have restricted to Boolean queries for simplicity.
Query evaluation on words

**Database**: a word $w$ where nodes have a color from an alphabet.

**Query Q**: a sentence (YES/NO question) in monadic second-order logic (MSO) (to be defined)

“Is there a blue node after each pink node?”
Database: a **word** $w$ where nodes have a color from an alphabet ○ ○ ○ ○

**Query Q:** a **sentence** (YES/NO question) in **monadic second-order logic** (MSO) *(to be defined)*

"Is there a blue node after each pink node?"

**Result:** YES/NO indicating if the word $w$ satisfies the query $Q$

→ Note that we have restricted to **Boolean queries** for simplicity
Monadic second-order logic (MSO)

- $P_\circ(x)$ means “$x$ is blue”; also $P_\circ(x)$, $P_\circ(x)$
- $x \rightarrow y$ means “$x$ is the predecessor of $y$”
Monadic second-order logic (MSO)

- \( P_\circ(x) \) means “\( x \) is blue”; also \( P_\circ(x), P_\circ(x) \)
- \( x \rightarrow y \) means “\( x \) is the predecessor of \( y \)”

- **Propositional logic:** formulas with AND \( \land \), OR \( \lor \), NOT \( \neg \)
  - \( P_\circ(x) \land P_\circ(y) \) means “Node \( x \) is pink and node \( y \) is blue”
Monadic second-order logic (MSO)

- $P(x)$ means “$x$ is blue”; also $P(x)$, $P(x)$
- $x \rightarrow y$ means “$x$ is the predecessor of $y$”

- Propositional logic: formulas with AND $\wedge$, OR $\vee$, NOT $\neg$
  - $P(x) \wedge P(y)$ means “Node $x$ is pink and node $y$ is blue”

- First-order logic: adds existential quantifier $\exists$ and universal quantifier $\forall$
  - $\exists x \forall y P(x) \wedge P(y)$ means “There is both a pink and a blue node”
Monadic second-order logic (MSO)

- $P(x)$ means “$x$ is blue”; also $P(x), P(y)$
- $x \rightarrow y$ means “$x$ is the predecessor of $y$”

- **Propositional logic**: formulas with AND $\land$, OR $\lor$, NOT $\neg$
  - $P(x) \land P(y)$ means “Node $x$ is pink and node $y$ is blue”

- **First-order logic**: adds existential quantifier $\exists$ and universal quantifier $\forall$
  - $\exists x \forall y P(x) \land P(y)$ means “There is both a pink and a blue node”

- **Monadic second-order logic (MSO)**: adds quantifiers over sets
  - $\exists S \forall x S(x)$ means “there is a set $S$ containing every element $x$”
  - Can express transitive closure $x \rightarrow^* y$, i.e., “$x$ is before $y$”
  - $\forall x P(x) \Rightarrow \exists y P(y) \land x \rightarrow^* y$
    means “There is a blue node after each pink node”
Translate the query $Q$ to a **deterministic word automaton**

**Alphabet:** $\begin{array}{c} \circ \circ \circ \end{array}$  

**$w$:** $\begin{array}{c} \circ \circ \circ \circ \circ \circ \end{array}$  

**$Q$:** $\exists x \, y \, P_{\circ}(x) \land P_{\circ}(y)$
Translate the query $Q$ to a deterministic word automaton

**Alphabet:** $\circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circle
Word automata

Translate the query $Q$ to a deterministic word automaton

Alphabet: \( \bigcirc \bigcirc \bigcirc \) \quad \text{w: } \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \quad Q: \exists x \ y \ P_\circ(x) \land P_\circ(y)

- States: \( \{ \bot, B, P, \top \} \)
- Final states: \( \{ \top \} \)
Translate the query $Q$ to a deterministic word automaton

Alphabet: $\circ \circ \circ$ \hspace{1cm} $w$: $\circ \circ \circ \circ \circ \circ$ \hspace{1cm} $Q$: $\exists x \ y \ P(x) \land P(y)$

- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: $\bot \  P \  B$
Translate the query $Q$ to a deterministic word automaton

**Alphabet:** $\{\bot, B, P, \top\}$

**Final states:** $\{\top\}$

**Initial function:** $\bot$

$Q$: $\exists x \, y \, P(x) \land P(y)$

**States:** $\{\bot, B, P, \top\}$

**Transitions:**
- $\bot \rightarrow P \rightarrow \top \rightarrow \top \rightarrow \top$
Translate the query $Q$ to a deterministic word automaton.

**Alphabet:** $w$: \[
\begin{array}{c}
\circ \circ \circ \circ \\
\circ \circ \circ \circ \circ \circ \circ
\end{array}
\]

$Q$: $\exists x \ y \ P_\circ(x) \land P_\circ(y)$

- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** $\bot \ P \ B$
- **Transitions (examples):** $\bot \ P \ P \ T \ T \ T$
Translate the query $Q$ to a **deterministic word automaton**

Alphabet: $\bigcirc \bigcirc \bigcirc \ w: \bigcirc \bigcirc \bigcirc \bigcirc \bigcirc \ P \ Q: \exists x \ y \ P(x) \land P(y)$

- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** $\bigcirc \bot \bigcirc P \bigcirc B$
- **Transitions (examples):** $\bot \bigcirc P \ P \bigcirc \top \bigcirc \top \bigcirc \top \bigcirc$
Word automata

Translate the query $Q$ to a **deterministic word automaton**

**Alphabet:** \[ w: \] 

- States: \{\bot, B, P, \top\}
- Final states: \{\top\}
- Initial function: \[ \bot \]
- Transitions (examples): \[ \bot \]
Translate the query $Q$ to a deterministic word automaton

Alphabet: \( \{\bot, B, P, \top\} \)

- States: \( \{\bot, B, P, \top\} \)
- Final states: \( \{\top\} \)
- Initial function: \( \bot \rightarrow P \rightarrow B \)
- Transitions (examples): \( \bot \rightarrow P \rightarrow \top \rightarrow \top \)
Translate the query $Q$ to a deterministic word automaton

Alphabet: $w: \emptyset \circ \circ \circ$  

$Q: \exists x \ y \ P(x) \land P(y)$

- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: $\bot \ P \ P \ B$
- Transitions (examples): $\bot \ P \ P \ T \ T \ T$
Translate the query \( Q \) to a deterministic word automaton

**Alphabet:** \( \emptyset \) \( w: \emptyset \) \( Q: \exists x y P(x) \land P(y) \)

- **States:** \( \{ \bot, B, P, \top \} \)
- **Final states:** \( \{ \top \} \)
- **Initial function:** \( \emptyset \) \( \bot \) \( P \) \( B \)
- **Transitions** (examples): \( \bot \) \( \top \) \( B \)

**Theorem (Büchi, 1960)**

\textit{MSO} and word automata and regular expressions have the same expressive power on words
Query evaluation on trees

**Database:** a tree $T$ where nodes have a color from an alphabet

Is there both a pink and a blue node?

$\exists x \ y \ P(x) \land P(y)$

**Result:** YES/NO indicating if the tree $T$ satisfies the query
Query evaluation on trees

**Database:** a tree $T$ where nodes have a color from an alphabet

**Query $Q$:** a sentence in monadic second-order logic (MSO)

- $P_\circ(x)$ means “$x$ is blue”
- $x \rightarrow y$ means “$x$ is the parent of $y$”

"Is there both a pink and a blue node?"

$\exists x \; y \; P_\circ(x) \land P_\circ(y)$
Query evaluation on trees

**Database:** a tree $T$ where nodes have a color from an alphabet 🟢🔵🟢

**Query Q:** a sentence in monadic second-order logic (MSO)
- $P_\bullet(x)$ means “$x$ is blue”
- $x \rightarrow y$ means “$x$ is the parent of $y$”

“Is there both a pink and a blue node?”

$\exists x \, y \, P_\bullet(x) \land P_\bullet(y)$

**Result:** YES/NO indicating if the tree $T$ satisfies the query $Q$
Tree alphabet:

- ○ ○ ○

**Tree automata**

- **Bottom-up deterministic tree automaton**
- "Is there both a pink and a blue node?"
- States: \{⊥, B, P, ⊤\}
- Final states: \{⊤\}
- Initial function: \(⊥ \rightarrow P \rightarrow B \rightarrow ⊤\)
- Transitions (examples):
  - \(P \rightarrow ⊤\)
  - \(B \rightarrow P \rightarrow ⊤\)

**Theorem** ([Thatcher and Wright](#))

MSO and tree automata have the same expressive power on trees.
Tree automata

Tree alphabet:

• Bottom-up deterministic tree automaton
• “Is there both a pink and a blue node?”

Theorem ([Thatcher and Wright]) MSO and tree automata have the same expressive power on trees.
Tree automata

Tree alphabet:

- Bottom-up deterministic tree automaton
- “Is there both a pink and a blue node?”
- **States:** \{⊥, B, P, ⊤\}
Tree automata

Tree alphabet:
- Bottom-up deterministic tree automaton
- “Is there both a pink and a blue node?”
- States: \( \{\bot, B, P, \top\} \)
- Final states: \( \{\top\} \)
Tree automata

Tree alphabet:

- Bottom-up deterministic tree automaton
- “Is there both a pink and a blue node?”
- States: \( \{\bot, B, P, \top\} \)
- Final states: \( \{\top\} \)
- Initial function: \( \bot \)

Theorem (Thatcher and Wright): MSO and tree automata have the same expressive power on trees.
Tree automata

Tree alphabet:

- Bottom-up deterministic tree automaton
- “Is there both a pink and a blue node?”
- States: \{⊥, B, P, ⊤\}
- Final states: \{⊤\}
- Initial function: ○ ⊥ ○ P ○ B
Tree automata

Tree alphabet: 

- Bottom-up deterministic tree automaton
- “Is there both a pink and a blue node?”
- States: \( \{ \bot, B, P, \top \} \)
- Final states: \( \{ \top \} \)
- Initial function: \( \bot \)
- Transitions (examples):

\[ \begin{align*}
P & \rightarrow \bot \rightarrow P \\
\bot & \rightarrow P \\
P & \rightarrow B \\
\bot & \rightarrow \bot 
\end{align*} \]
Tree automata

Tree alphabet: 

- Bottom-up deterministic tree automaton
- "Is there both a pink and a blue node?"
- States: \( \{ \bot, B, P, \top \} \)
- Final states: \( \{ \top \} \)
- Initial function: \( \bot P B \)
- Transitions (examples): 

\[
\begin{align*}
& P \quad \bot \quad P \quad B \\
& P \quad \bot \quad P \quad B \\
& P \quad \bot \\
& \top \\
& \bot
\end{align*}
\]
Bottom-up deterministic tree automaton

"Is there both a pink and a blue node?"

States: \(\{\bot, B, P, \top\}\)

Final states: \(\{\top\}\)

Initial function: 

Transitions (examples):
Tree automata

Tree alphabet:

- Bottom-up deterministic tree automaton
- “Is there both a pink and a blue node?”
- States: \{\bot, B, P, \top\}
- Final states: \{\top\}
- Initial function: \(\bullet\) \bot \(\bullet\) \(\bullet\) \(P\)
- Transitions (examples):

**Theorem ([Thatcher and Wright 1968])**

MSO and tree automata have the same expressive power on trees
We study data that has the shape of a **word** or **tree**
→ e.g., sequences of events, XML documents, etc.

Some queries **cannot be expressed** in relational algebra
→ e.g., “is there a blue node after each pink node?”

We restrict to **Boolean queries** (YES/NO question)

The queries can be specified:
- In a logical language (MSO)
- On words, as a **regular expression**
  → As an **automaton**
Outline

Query Evaluation on Relational Databases
Boolean Provenance on Relational Databases
Semiring Provenance on Relational Databases
Query Evaluation on Trees and Words
Boolean Provenance on Trees and Words
Applications to Probability Computation
Applications to Enumeration
Conclusion
Motivation and definition

- Goal: notion of **provenance** for queries on trees/words expressed as **automata**
- We show how to **define** Boolean provenance in this context and how to **compute** it

Remarks:
- We work with **Boolean queries** (YES/NO) so the provenance will just describe when we get the answer YES
- We restrict to **Boolean provenance** – but generalizations possible

[Amarilli, Bourhis, and Senellart /two.osf/zero.osf/one.osf/five.osf/seven.osf/one.osf]
• Goal: notion of **provenance** for queries on trees/words expressed as **automata**

• We show how to **define** Boolean provenance in this context and how to **compute** it

Remarks:

→ We work with **Boolean queries** (YES/NO) so the provenance will just describe when we get the answer YES
Motivation and definition

- Goal: notion of **provenance** for queries on trees/words expressed as **automata**
- We show how to define Boolean provenance in this context and how to **compute** it

Remarks:

→ We work with **Boolean queries** (YES/NO) so the provenance will just describe when we get the answer YES
→ We restrict to **Boolean provenance** – but generalizations possible [Amarilli, Bourhis, and Senellart 2015a]
Defining provenance: Uncertain trees

A valuation of a tree decides whether to keep or discard node labels. Valuation:

\[
\{/two.osf, /three.osf, /seven.osf \mapsto /one.osf, \ast \mapsto /zero.osf\}
\]

Q: "Is there both a pink and a blue node?"
A valuation of a tree decides whether to keep (1) or discard (0) node labels.
Defining provenance: Uncertain trees

A *valuation* of a tree decides whether to keep (1) or discard (0) node labels.

Valuation: \{2, 3, 7 \mapsto 1, \ * \mapsto 0\}
Defining provenance: Uncertain trees

A **valuation** of a tree decides whether to keep (1) or discard (0) node labels.

Valuation: \( \{2 \mapsto 1, \, \ast \mapsto 0\} \)
A **valuation** of a tree decides whether to keep (1) or discard (0) node labels.

**Valuation:** \{2, 7 \mapsto 1, * \mapsto 0\}

Q: "Is there both a pink and a blue node?"
Defining provenance: Uncertain trees

A valuation of a tree decides whether to keep (1) or discard (0) node labels

Valuation: \{2, 7 \mapsto 1, \ast \mapsto 0\}

Q: “Is there both a pink and a blue node?”
Defining provenance: Uncertain trees

A **valuation** of a tree decides whether to keep (1) or discard (0) node labels

**Valuation:** \{2, 3, 7 \mapsto 1, \; * \mapsto 0\}

**Q:** “Is there both a pink and a blue node?”

The query Q returns **YES**
A **valuation** of a tree decides whether to keep (1) or discard (0) node labels

Valuation: \{2 \mapsto 1, \星际 \mapsto 0\}

**Q:** “Is there both a pink and a blue node?”

The query **Q** returns **NO**
A **valuation** of a tree decides whether to keep (1) or discard (0) node labels.

Valuation: \( \{2, 7 \mapsto 1, \ast \mapsto 0\} \)

**Q:** “Is there both a pink and a blue node?”

The query Q returns **YES**.
Query: Is there both a pink and a blue node?
Query: Is there both a pink and a blue node?

Provenance circuit:
Example: Provenance circuit

Query: Is there both a pink and a blue node?

Provenance circuit:

Formally:

- Boolean query $Q$, uncertain tree $T$, circuit $C$
- **Variable gates** of $C$: nodes of $T$
- **Condition**: Let $\nu$ be a valuation of $T$, then $\nu(C)$ iff $\nu(T)$ satisfies $Q$
Theorem

For any bottom-up tree automaton $A$ and input tree $T$, we can build a Boolean provenance circuit of $A$ on $T$ in $O(|A| \times |T|)$.
Theorem

For any bottom-up tree automaton $A$ and input tree $T$, we can build a Boolean provenance circuit of $A$ on $T$ in $O(|A| \times |T|)$.

- **Alphabet:** $\bigcirc \bigcirc \bigcirc \bigcirc$
- **Automaton:** “Is there both a pink and a blue node?”
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**
  - $\top \rightarrow \bot \rightarrow P \rightarrow \top$
  - $\bot \rightarrow P \rightarrow \bot$
  - $P \rightarrow \bot \rightleftharpoons P \rightarrow \bot$
Theorem

For any bottom-up tree automaton $A$ and input tree $T$, we can build a Boolean provenance circuit of $A$ on $T$ in $O(|A| \times |T|)$.

- **Alphabet:**
  - $\mathbb{O}$
  - $\mathbb{P}$
  - $\mathbb{B}$
  - $\mathbb{T}$

- **Automaton:** “Is there both a pink and a blue node?”

- **States:**
  - $\{\bot, B, P, \top\}$

- **Final:** $\{\top\}$

- **Transitions:**
  - $\top \rightarrow \bot \rightarrow P \rightarrow \bot$
Theorem

For any bottom-up tree automaton \( A \) and input tree \( T \), we can build a Boolean provenance circuit of \( A \) on \( T \) in \( O(|A| \times |T|) \).

- **Alphabet:** \( \{\circ, \bullet, \square, \triangle \} \)
- **Automaton:** “Is there both a pink and a blue node?”
- **States:** \( \{\bot, B, P, \top\} \)
- **Final:** \( \{\top\} \)
- **Transitions:**
  - \( \top \rightarrow \bot \)
  - \( P \rightarrow P \)
  - \( P \rightarrow \bot \)
Theorem

For any bottom-up tree automaton $A$ and input tree $T$, we can build a Boolean provenance circuit of $A$ on $T$ in $O(|A| \times |T|)$.

- **Alphabet:** $\bigcirc \bigcirc \bigcirc \bigcirc$
- **Automaton:** “Is there both a pink and a blue node?”
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

```
\begin{array}{cccc}
\top & \bot & P & \top \\
\bot & B & P & \top \\
\bot & B & P & \top \\
\bot & B & P & \top \\
\bot & B & P & \top \\
\end{array}
```
Provenance circuits on trees [Amarilli, Bourhis, and Senellart 2015b]

Theorem

For any bottom-up tree automaton $A$ and input tree $T$, we can build a Boolean provenance circuit of $A$ on $T$ in $O(|A| \times |T|)$.

- **Alphabet:**  
  - $\bigcirc$, $\bigcirc$, $\bigcirc$, $\bigcirc$

- **Automaton:** “Is there both a pink and a blue node?”

- **States:**  
  - $\{\bot, B, P, \top\}$

- **Final:**  
  - $\{\top\}$

- **Transitions:**  
  - $\begin{array}{c} P \\ \downarrow \\ B \\ \downarrow \\ P \\ \downarrow \\ \top \\ \downarrow \\ B \\ \downarrow \\ P \\ \downarrow \\ \top \end{array}$

\[ n \quad \downarrow \quad \bot \quad B \quad P \quad \top \]

\[ n \quad \Uparrow \quad \bot \quad B \quad P \quad \top \]
Theorem

For any bottom-up tree automaton $A$ and input tree $T$, we can build a Boolean provenance circuit of $A$ on $T$ in $O(|A| \times |T|)$. 

- **Alphabet:** $\circ \circ \circ \circ$
- **Automaton:** “Is there both a pink and a blue node?”
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**
Theorem

For any bottom-up tree automaton $A$ and input tree $T$, we can build a Boolean provenance circuit of $A$ on $T$ in $O(|A| \times |T|)$.

- **Alphabet:** $\emptyset$
- **Automaton:** “Is there both a pink and a blue node?”
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

\[
\begin{array}{c}
\text{\bot} & \text{\top} & \text{P} & \text{P} \\
\lor & \lor & \lor & \lor \\
\text{\bot} & \text{B} & \text{P} & \text{T} \\
\land & \land & \land & \land \\
\text{n} & \text{n} & \text{n} & \text{n} \\
\end{array}
\]
Connections to knowledge compilation

The provenance circuits of automata on trees are...

• DNNF circuits: Negations only at the leaves, conjunctions are between disjoint subtrees.
• Structured circuits: The v-tree follows the shape of the input tree.
• \(d\)-SDNNFs when the input automaton is deterministic.
  Of width bounded by the number of states of the automaton.

\[\text{Capelli and Mengel /two.osf/zero.osf/one.osf/nine.osf}\]

→ Remark: for words, we obtain diagrams (OBDDs, etc.).

→ Ongoing work: investigating these connections in more detail.
Connections to knowledge compilation

The provenance circuits of automata on trees are...

- **DNNF** circuits:
  - Negations only at the *leaves*
  - Conjunctions are between *disjoint* subtrees
Connections to knowledge compilation

The provenance circuits of automata on trees are...

- **DNNF** circuits:
  - Negations only at the **leaves**
  - Conjunctions are between **disjoint** subtrees

- **Structured** circuits
  - The v-tree follows the **shape** of the input tree
The provenance circuits of automata on trees are...

- **DNNF** circuits:
  - Negations only at the leaves
  - Conjunctions are between disjoint subtrees
- **Structured** circuits
  - The v-tree follows the shape of the input tree
- **d-SDNNFs** when the input automaton is deterministic
Connections to knowledge compilation

The provenance circuits of automata on trees are...

- **DNNF** circuits:
  - Negations only at the leaves
  - Conjunctions are between disjoint subtrees
- **Structured** circuits
  - The v-tree follows the shape of the input tree
- **d-SDNNFs** when the input automaton is deterministic
- Of width bounded by the number of states of the automaton
  [Capelli and Mengel 2019]
Connections to knowledge compilation

The provenance circuits of automata on trees are...

- **DNNF circuits:**
  - Negations only at the leaves
  - Conjunctions are between disjoint subtrees
- **Structured circuits**
  - The v-tree follows the shape of the input tree
- **d-SDNNFs** when the input automaton is deterministic
- **Of width** bounded by the number of states of the automaton
  [Capelli and Mengel 2019]

Remark: for words, we obtain diagrams (OBDDs, etc.)
Connections to knowledge compilation

The provenance circuits of automata on trees are...

- **DNNF** circuits:
  - Negations only at the **leaves**
  - Conjunctions are between **disjoint** subtrees

- **Structured** circuits
  - The v-tree follows the **shape** of the input tree

- **d-SDNNFs** when the input automaton is **deterministic**

- Of **width** bounded by the number of **states** of the automaton
  [Capelli and Mengel 2019]

- Remark: for **words**, we obtain **diagrams** (OBDDs, etc.)

- **Ongoing work**: investigating these connections in more detail
Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion
**Probabilistic databases** [Green and Tannen 2006; Suciu, Olteanu, Ré, and Koch 2011]

- **Tuple-independent database** $D$: each tuple $t$ in $D$ is annotated with *independent* probability $Pr(t)$ of existing

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>0.5</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>0.7</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>0.3</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>0.2</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>1.0</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>0.8</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>0.2</td>
</tr>
</tbody>
</table>
Probabilistic databases [Green and Tannen 2006; Suciu, Olteanu, Ré, and Koch 2011]

- **Tuple-independent database** \( D \): each tuple \( t \) in \( D \) is annotated with independent probability \( \Pr(t) \) of existing

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>0.5</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>0.7</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>0.3</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>0.2</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>1.0</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>0.8</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>0.2</td>
</tr>
</tbody>
</table>

→ Probability of a possible world \( D' \subseteq D \):

\[
\Pr(D') = \prod_{t \in D'} \Pr(t) \times \prod_{t \notin D' \setminus D}(1 - \Pr(t'))
\]
How can we evaluate a query $Q$ over a probabilistic database?
Query evaluation on probabilistic databases (PQE)

How can we evaluate a query $Q$ over a probabilistic database?

• Probability of a tuple for a query $Q$ over $D$:

$$\Pr(t \in Q(D)) = \sum_{D' \subseteq D} \Pr(D') \Pr(t \in Q(D'))$$

• **Intuitively**: the probability of answer tuple $t$ is the probability of drawing a possible world $D' \subseteq D$ where $t$ is an answer
How can we evaluate a query \( Q \) over a probabilistic database?

- Probability of a tuple for a query \( Q \) over \( D \):
  \[
  \Pr(t \in Q(D)) = \sum_{D' \subseteq D} \Pr(D') 
  \]

- Intuitively: the probability of answer tuple \( t \) is the probability of drawing a possible world \( D' \subseteq D \) where \( t \) is an answer

**Probabilistic query evaluation (PQE) problem** for a query \( Q \): given a tuple-independent database, compute the probability of each answer

→ **Idea:** we can do this using **Boolean provenance**: the probability of \( t \) is the probability of its annotation
### Example of PQE

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>$x_1$</td>
<td>0.5</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>$x_2$</td>
<td>0.7</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>$x_3$</td>
<td>0.3</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_4$</td>
<td>0.2</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>$x_5$</td>
<td>1.0</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>$x_6$</td>
<td>0.8</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>$x_7$</td>
<td>0.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>city</th>
<th>prov</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>$x_1 \lor x_2$</td>
</tr>
<tr>
<td>Paris</td>
<td>$x_3 \lor x_5 \lor x_6$</td>
</tr>
<tr>
<td>Berlin</td>
<td>$x_4 \lor x_7$</td>
</tr>
</tbody>
</table>
## Example of PQE

<table>
<thead>
<tr>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>prov</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>(x_1)</td>
<td>0.5</td>
</tr>
<tr>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>(x_2)</td>
<td>0.7</td>
</tr>
<tr>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>(x_3)</td>
<td>0.3</td>
</tr>
<tr>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>(x_4)</td>
<td>0.2</td>
</tr>
<tr>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top secret</td>
<td>(x_5)</td>
<td>1.0</td>
</tr>
<tr>
<td>Nancy</td>
<td>HR director</td>
<td>Paris</td>
<td>restricted</td>
<td>(x_6)</td>
<td>0.8</td>
</tr>
<tr>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>(x_7)</td>
<td>0.2</td>
</tr>
</tbody>
</table>

### City Probability Calculation

<table>
<thead>
<tr>
<th>city</th>
<th>prov</th>
<th>prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>(x_1 \lor x_2)</td>
<td>(1 - (1 - 0.5) \times (1 - 0.7) = 0.85)</td>
</tr>
<tr>
<td>Paris</td>
<td>(x_3 \lor x_5 \lor x_6)</td>
<td>1.00</td>
</tr>
<tr>
<td>Berlin</td>
<td>(x_4 \lor x_7)</td>
<td>(1 - (1 - 0.2) \times (1 - 0.2) = 0.36)</td>
</tr>
</tbody>
</table>
Complexity of PQE

- In general, PQE is intractable (#P-hard)

- For select-project-join queries without self-joins:
  - Either the query is hierarchical and the Boolean provenance is always a read-once formula
  - Or the query is unsafe (#P-hard) [Dalvi and Suciu; Olteanu and Huang]

- For positive relational algebra:
  - Dichotomy between tractable (safe) and unsafe queries [Dalvi and Suciu]

- Open problem: are queries safe because of their provenance? → Intensional vs extensional conjecture
Complexity of PQE

• In general, PQE is intractable (#P-hard)
• For select-project-join queries without self-joins:
  • Either the query is hierarchical and the Boolean provenance is always a read-once formula
  • Or the query is unsafe (#P-hard) [Dalvi and Suciu 2007; Olteanu and Huang 2008]
Complexity of PQE

- In general, PQE is **intractable** (#P-hard)
- For **select-project-join** queries without **self-joins**:
  - Either the query is **hierarchical** and the Boolean provenance is always a **read-once formula**
  - Or the query is **unsafe** (#P-hard) [Dalvi and Suciu 2007; Olteanu and Huang 2008]
- For **positive relational algebra**:
  - Dichotomy between tractable (**safe**) and **unsafe** queries [Dalvi and Suciu 2012]
  - **Open problem**: are queries safe because of their provenance?
  → **Intensional vs extensional conjecture**
More about the intensional vs extensional conjecture

**Open question:** do all *safe* relational algebra queries admit provenance representations in a *tractable* circuit formalism?

- For **OBDDs**: there is a characterization of the queries with polynomial-sized OBDDs.
- For **DLDDs** (e.g., dec-DNNFs), some safe queries have no tractable provenance representation in this class.
- For **d-SDNNF**, some safe queries have no tractable provenance representation in this class.
- Good candidate: **d-DNNF** or **d-D** (allows arbitrary negations) → Note: it's open whether d-DNNFs and d-Ds are indeed different!}
- Crux of the problem: capture arithmetic operations on probabilities with a d-D circuit, specifically inclusion-exclusion.
- Latest results: [Monet](#) or chat with me at the coffee break!
More about the intensional vs extensional conjecture

Open question: do all safe relational algebra queries admit provenance representations in a tractable circuit formalism?

- For OBDDs: there is a characterization of the queries with polynomial-sized OBDDs [Jha and Suciu 2013]
More about the intensional vs extensional conjecture

Open question: do all *safe* relational algebra queries admit provenance representations in a *tractable* circuit formalism?

- For **OBDDs**: there is a characterization of the queries with polynomial-sized OBDDs [Jha and Suciu 2013]
- For **DLDDs** (e.g., dec-DNNFs), some safe queries have no tractable provenance representation in this class [Beame, Li, Roy, and Suciu 2017]

• Good candidate: **d-DNNF**, or **d-D** (allows arbitrary negations)

• Note: it’s open whether d-DNNFs and d-Ds are indeed different :)
More about the intensional vs extensional conjecture

Open question: do all safe relational algebra queries admit provenance representations in a tractable circuit formalism?

- For OBDDs: there is a characterization of the queries with polynomial-sized OBDDs [Jha and Suciu 2013]
- For DLDDs (e.g., dec-DNNFs), some safe queries have no tractable provenance representation in this class [Beame, Li, Roy, and Suciu 2017]
- For d-SDNNF, some safe queries have no tractable provenance representation in this class [Bova and Szeider 2017]
Open question: do all safe relational algebra queries admit provenance representations in a tractable circuit formalism?

- For **OBDDs**: there is a characterization of the queries with polynomial-sized OBDDs [Jha and Suciu 2013]
- For **DLDDs** (e.g., dec-DNNFs), some safe queries have no tractable provenance representation in this class [Beame, Li, Roy, and Suciu 2017]
- For **d-SDNNF**, some safe queries have no tractable provenance representation in this class [Bova and Szeider 2017]
- Good candidate: **d-DNNF**, or **d-D** (allows arbitrary negations)
  → Note: it’s open whether d-DNNFs and d-Ds are indeed different :)
Open question: do all safe relational algebra queries admit provenance representations in a tractable circuit formalism?

- For **OBDDs**: there is a characterization of the queries with polynomial-sized OBDDs [Jha and Suciu 2013]
- For **DLDDs** (e.g., dec-DNNFs), some safe queries have no tractable provenance representation in this class [Beame, Li, Roy, and Suciu 2017]
- For **d-SDNNF**, some safe queries have no tractable provenance representation in this class [Bova and Szeider 2017]
- Good candidate: **d-DNNF**, or **d-D** (allows arbitrary negations)
  → Note: it’s open whether d-DNNFs and d-Ds are indeed different :)
- **Crux of the problem**: capture arithmetic operations on probabilities with a d-D circuit, specifically **inclusion-exclusion**
More about the intensional vs extensional conjecture

Open question: do all safe relational algebra queries admit provenance representations in a tractable circuit formalism?

- For **OBDDs**: there is a characterization of the queries with polynomial-sized OBDDs [Jha and Suciu 2013]
- For **DLDDs** (e.g., dec-DNNFs), some safe queries have no tractable provenance representation in this class [Beame, Li, Roy, and Suciu 2017]
- For **d-SDNNF**, some safe queries have no tractable provenance representation in this class [Bova and Szeider 2017]
- Good candidate: **d-DNNF**, or **d-D** (allows arbitrary negations)
  → Note: it’s open whether d-DNNFs and d-Ds are indeed different :)
- **Crux of the problem**: capture arithmetic operations on probabilities with a d-D circuit, specifically **inclusion-exclusion**
- Latest results: [Monet 2020] or chat with me at the coffee break :)
Probabilistic query evaluation on trees

Query: Is there both a pink and a blue node?

Provenance circuit:
Consider a query $Q$ on a **probabilistic tree** (each node has an independent probability of keeping its color)

For queries given as **unambiguous tree automata**, we can construct a $d$-SDNNF **provenance circuit**

$\rightarrow$ PQE is **tractable** for tree automata on trees
Probabilistic query evaluation on trees

Query: Is there both a pink and a blue node?

Provenance circuit:

- Consider a query $Q$ on a probabilistic tree (each node has an independent probability of keeping its color)
- For queries given as unambiguous tree automata, we can construct a d-SDNNF provenance circuit
  - PQE is tractable for tree automata on trees
- Extends to bounded treewidth databases – and essentially only to them [Amarilli, Bourhis, and Senellart 2016]
Probabilistic query evaluation on trees

Query: Is there both a pink and a blue node?

Provenance circuit:

• Consider a query $Q$ on a **probabilistic tree** (each node has an independent probability of keeping its color)
• For queries given as **unambiguous tree automata**, we can construct a d-SDNNF **provenance circuit**
  → PQE is **tractable** for tree automata on trees
→ Extends to **bounded treewidth** databases – and essentially only to them [Amarilli, Bourhis, and Senellart 2016]
→ Relates to probability computation on bounded-treewidth **graphical models** [Amarilli, Capelli, Monet, and Senellart 2019]
Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion
**Enumerating query results**

**Idea:** Often, we do not need to compute **all results** of a query; we just need to be able to **enumerate** results quickly.
Enumerating query results

Idea: Often, we do not need to compute all results of a query, we just need to be able to enumerate results quickly.

🔍 how to find patterns
Enumerating query results

**Idea:** Often, we do not need to compute all results of a query. We just need to be able to enumerate results quickly.
**Enumerating query results**

**Idea:** Often, we do not need to compute all results of a query; we just need to be able to enumerate results quickly.
Enumerating query results

Idea: Often, we do not need to compute all results of a query, we just need to be able to enumerate results quickly.
Enumerating query results

Idea: Often, we do not need to compute all results of a query, we just need to be able to enumerate results quickly.

→ Formalization: enumeration algorithms
→ Currently a pretty important topic in database theory.
Enumeration algorithm (linear preprocessing, constant delay)

Input

Step /one.osf:
Indexing in O(input)
Indexed input

Step /two.osf:
Enumeration in O(/one.osf)

x y z
a b c
a' b c
a b' c
a' b' c

Results
State
/six.osf/four.osf//seven.osf/one.osf
Enumeration algorithm (linear preprocessing, constant delay)

Step 1:
Indexing in $O(\text{input})$
Step 1: Indexing in $O(\text{input})$

Indexed input
Enumeration algorithm (linear preprocessing, constant delay)

1. **Step 1: Indexing in O(input)**
   - Input
   - Indexed input

2. **Step 2: Enumeration in O(1)**
   - Results
   - State
Enumeration algorithm (linear preprocessing, constant delay)

Input

Step 1: Indexing in $O(\text{input})$

Indexed input

Step 2: Enumeration in $O(1)$

Results

\[
\begin{array}{ccc}
  x & y & z \\
  a & b & c \\
\end{array}
\]
Enumeration algorithm (linear preprocessing, constant delay)

Input

Step 1: Indexing in $O(\text{input})$

Indexed input

Step 2: Enumeration in $O(1)$

Results

State

0011

Input

x y z

a b c
Enumeration algorithm (linear preprocessing, constant delay)

Input

Step 1: Indexing in $O(\text{input})$

Indexed input

Step 2: Enumeration in $O(1)$

Results

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

State

0011
Enumeration algorithm (linear preprocessing, constant delay)

Input

Step 1: Indexing in $O(\text{input})$

Indexed input

Step 2: Enumeration in $O(1)$

Results

$x \ y \ z$

$\ a' \ b \ c$

State

010001
Enumeration algorithm (linear preprocessing, constant delay)

Input

Step 1: Indexing in $O(\text{input})$

Indexed input

Step 2: Enumeration in $O(1)$

State

Results

$01100111$

$x \ y \ z$

$a \ b' \ c$
Enumeration algorithm (linear preprocessing, constant delay)

Input

Step 1: Indexing in $O(\text{input})$

Indexed input

Step 2: Enumeration in $O(1)$

Results

$\downarrow$

State

$\begin{array}{ccc}
x & y & z \\
a' & b' & c \\
\end{array}$
Provenance can also represent **query answers**!
Connection to provenance

Provenance can also represent query answers!

• Study answers of non-Boolean query

\[ Q(x, y) \] on database \( D \)
Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database $D$

  $Q(x, y) : \exists z \ R(x, y) \land S(y, z)$

  $D : R(a, b), R(a', b), S(b, c)$
Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database $D$

- Add **assignment facts** $X(v), Y(v)$ to $D$ for each element $v$ (linear)
Connection to provenance

Provenance can also represent query answers!

- Study answers of non-Boolean query $Q(x, y)$ on database $D$
  
  $Q(x, y) : \exists z \ R(x, y) \land S(y, z)$
  $D : R(a, b), R(a', b), S(b, c)$

- Add assignment facts $X(v), Y(v)$ to $D$
  for each element $v$ (linear)

  $X(a), X(a'), X(b), X(c)$
  $Y(a), Y(a'), Y(b), Y(c)$
Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database $D$

- Add **assignment facts** $X(v), Y(v)$ to $D$ for each element $v$ (linear)

- Consider the **Boolean query** $Q' : X(x) \land Y(y) \land Q(x, y)$
Connection to provenance

Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** \( Q(x, y) \) on database \( D \)
- Add **assignment facts** \( X(v), Y(v) \) to \( D \) for each element \( v \) (linear)
- Consider the **Boolean query** \( Q' : X(x) \land Y(y) \land Q(x, y) \)

\[
Q(x, y) : \exists z R(x, y) \land S(y, z)
\]
\[
D : R(a, b), R(a', b), S(b, c)
\]
\[
X(a), X(a'), X(b), X(c)
\]
\[
Y(a), Y(a'), Y(b), Y(c)
\]
\[
X(x) \land Y(y) \land (\exists z R(x, y) \land S(y, z))
\]
Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database $D$
- Add **assignment facts** $X(v), Y(v)$ to $D$
  for each element $v$ (linear)
- Consider the **Boolean query** $Q' : X(x) \land Y(y) \land Q(x, y)$
- Compute the **provenance** $C'$ of $Q'$ on $D$ plus assignment facts

$Q(x, y) : \exists z \ R(x, y) \land S(y, z)$

$D : R(a, b), R(a', b), S(b, c)$

$X(a), X(a'), X(b), X(c)$

$Y(a), Y(a'), Y(b), Y(c)$

$X(x) \land Y(y) \land (\exists z \ R(x, y) \land S(y, z))$
Provenance can also represent query answers!

- Study answers of non-Boolean query $Q(x, y)$ on database $D$
  
  - Add assignment facts $X(v), Y(v)$ to $D$ for each element $v$ (linear)

- Consider the Boolean query $Q' : X(x) \land Y(y) \land Q(x, y)$
  
  - Compute the provenance $C'$ of $Q'$ on $D$ plus assignment facts
Provenance can also represent **query answers**!

- Study answers of non-Boolean query $Q(x, y)$ on database $D$
  
  $$Q(x, y) : \exists z\ R(x, y) \land S(y, z)$$
  $$D : R(a, b), R(a', b), S(b, c)$$

- Add **assignment facts** $X(v), Y(v)$ to $D$
  
  for each element $v$ (linear)
  
  $$X(a), X(a'), X(b), X(c)$$
  $$Y(a), Y(a'), Y(b), Y(c)$$

- Consider the **Boolean query** $Q' : X(x) \land Y(y) \land Q(x, y)$
  
  $$X(x) \land Y(y) \land (\exists z\ R(x, y) \land S(y, z))$$

- Compute the **provenance** $C'$ of $Q'$
  
  $$(X(a) \land R(a, b) \lor X(a') \land R(a', b)) \land Y(b) \land S(b, c)$$

- Define $C$ by replacing all variables by 1 except assignment facts
Connection to provenance

Provenance can also represent **query answers**!

- Study answers of non-Boolean query $Q(x, y)$ on database $D$
  - $Q(x, y) : \exists z \ R(x, y) \land S(y, z)$
  - $D : R(a, b), R(a', b), S(b, c)$

- Add **assignment facts** $X(v), Y(v)$ to $D$
  - for each element $v$ (linear)
  - $X(a), X(a'), X(b), X(c)$
  - $Y(a), Y(a'), Y(b), Y(c)$

- Consider the **Boolean query** $Q' : X(x) \land Y(y) \land Q(x, y)$
  - $X(x) \land Y(y) \land (\exists z \ R(x, y) \land S(y, z))$

- Compute the **provenance** $C'$ of $Q'$ on $D$ plus assignment facts
  - $(X(a) \land R(a, b) \lor X(a') \land R(a', b))$
  - $\land Y(b) \land S(b, c)$

- Define $C$ by replacing all variables by 1 except assignment facts
  - $(X(a) \lor X(a')) \land Y(b)$
Provenance can also represent **query answers**!

- Study answers of **non-Boolean query**
  \[ Q(x, y) : \exists z \ R(x, y) \land S(y, z) \]
  Database \( D \):
  \[ D : R(a, b), R(a', b), S(b, c) \]

- Add **assignment facts** \( X(v), Y(v) \) to \( D \)
  for each element \( v \) (linear)

- Consider the **Boolean query**
  \[ Q' : X(x) \land Y(y) \land (\exists z \ R(x, y) \land S(y, z)) \]

- Compute the **provenance** \( C' \) of \( Q' \)
  \[ (X(a) \land R(a, b) \lor X(a') \land R(a', b)) \land Y(b) \land S(b, c) \]

- Define \( C \) by replacing all variables by \( 1 \)
  except assignment facts

\[ (X(a) \lor X(a')) \land Y(b) \]

\( \rightarrow \) The circuit \( C \) represents the **query answers**
Connection to provenance

Provenance can also represent query answers!

- Study answers of non-Boolean query
  \[ Q(x, y) : \exists z \ R(x, y) \land S(y, z) \]
  on database \( D \)

- Add assignment facts \( X(v), Y(v) \) to \( D \)

- Consider the Boolean query
  \[ Q' : X(x) \land Y(y) \land Q(x, y) \]

- Compute the provenance \( C' \) of \( Q' \)

- Define \( C \) by replacing all variables by \( 1 \) except assignment facts

→ The circuit \( C \) represents the query answers \((a, b)\) and \((a', b)\)
• We have a **provenance circuit** representing the query answers.

```
\( \neg X(a) \lor X(a') \land Y(b) \)
```
• We have a **provenance circuit** representing the query answers

\[
\begin{align*}
\land & \quad \lor \\
X(a) & \quad X(a') & \quad Y(b)
\end{align*}
\]

• So to **enumerate query answers** we can:
  • **Compute** this provenance circuit
  • **Enumerate** its satisfying assignments

---

**Theorem** ([Amarilli, Bourhis, Jachiet, and Mengel](#))

Given a \(d\)-SDNNF circuit, we can preprocess it in linear time and then enumerate its satisfying assignments with constant delay (if the assignments have constant size).
Enumeration via provenance and knowledge compilation

- We have a **provenance circuit** representing the query answers

  \[ X(a) \lor Y(b) \land X(a') \]

- So to **enumerate query answers** we can:
  - **Compute** this provenance circuit
  - **Enumerate** its satisfying assignments

→ We want **linear preprocessing** and **constant delay**
  so we had to do our own enumeration algorithm for circuits:

**Theorem ([Amarilli, Bourhis, Jachiet, and Mengel 2017])**

*Given a *d*-SDNNF circuit, we can preprocess it in **linear time**
and then enumerate its satisfying assignments with **constant delay**
(if the assignments have constant size)*
Enumeration via knowledge compilation

Currently:

Input → Enumeration → Results

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>a</td>
<td>b'</td>
<td>c</td>
</tr>
</tbody>
</table>

Our idea:

Compilation \( \cup \times x \top \times z \) Circuit → Enumeration → Results
Enumeration via knowledge compilation

Currently:

Input

Enumeration

Results

A  B  C
a  b  c
a  b' c

Our idea:

Input

Compilation

∪

×

x

⊤

×

z

Circuit

Input

Enumeration

Results

A  B  C
a  b  c
a  b' c

/six.osf/seven.osf//seven.osf/one.osf
Enumeration via knowledge compilation

Currently:

Input → Enumeration → Results

Our idea:

Input → Compilation → Union → Cross Product → Circuit → Enumeration → Results
Enumeration via knowledge compilation

Currently:

Input → Enumeration → Results

A B C
a b c
a b' c

Results

Our idea:

Input → Compilation → Circuit

∪

×

x ⊤

×

z

Circuit

/six.osf/seven.osf//seven.osf/one.osf
Enumeration via knowledge compilation

Currently:

Input → Enumeration → Results

Input → Enumeration → Results

Input → Enumeration → Results

Our idea:

Input → Compilation → Circuit

Input → Enumeration → Results
Enumeration via knowledge compilation

Currently:

Input

Enumeration

Results

A B C

a b c
a b' c

Circuit

Results

A B C

a b c
a b' c

Our idea:

Input

Compilation

Circuit

Results

A B C

a b c
a b' c

Circuit

Results

A B C

a b c
a b' c

Circuit
Enumeration via knowledge compilation

Currently:

Input

Enumeration

Results

A B C
a b c a b’ c

Input

Enumeration

Results

A B C
a b c a b’ c

Input

Enumeration

Results

A B C
a b c a b’ c

Our idea:

Input

Compilation

Circuit

∪

×

x

⊤

×

z

A B C
a b c a b’ c

Input

Compilation

Circuit

∪

×

x

⊤

×

z

A B C
a b c a b’ c

Input

Compilation

Circuit

∪

×

x

⊤

×

z

A B C
a b c a b’ c

Circuit

Enumeration

Results

∪

×

x

⊤

×

z

A B C
a b c a b’ c

/six.osf/seven.osf//seven.osf/one.osf
• Remember that, for tree automata on trees, we can build d-SDNNF provenance representations in \textbf{linear time}
Results and extensions

• Remember that, for tree automata on trees, we can build d-SDNNF provenance representations in linear time

• With our enumeration result, this shows that we can enumerate query results with linear preprocessing and constant-delay
  → Was already known in database theory [Bagan 2006; Kazana and Segoufin 2013]
Results and extensions

- Remember that, for tree automata on trees, we can build d-SDNNF provenance representations in **linear time**
- With our enumeration result, this shows that we can enumerate query results with **linear preprocessing** and **constant-delay**
  - Was already known in database theory [Bagan 2006; Kazana and Segoufin 2013]
- When the data **changes**, we can **update** the provenance circuit efficiently [Amarilli, Bourhis, Mengel, and Niewerth 2019]
  - Refines existing database theory results
Results and extensions

- Remember that, for tree automata on trees, we can build d-SDNNF provenance representations in **linear time**
- With our enumeration result, this shows that we can enumerate query results with **linear preprocessing** and **constant-delay**
  - → **Was already known** in database theory [Bagan 2006; Kazana and Segoufin 2013]
- When the data **changes**, we can **update** the provenance circuit efficiently [Amarilli, Bourhis, Mengel, and Niewerth 2019]
  - → **Refines existing database theory results**
- We can make the enumeration **tractable** in the input query
  - → **Will be presented by Matthias** tomorrow (on words)

**Ongoing work:** provenance-based enumeration for relational algebra
Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion
How can we compute provenance in *practice*?

- **ProvSQL** module for PostgreSQL, by Pierre Senellart et al.
- Keeps track of provenance as a circuit
- [https://github.com/PierreSenellart/provsql](https://github.com/PierreSenellart/provsql)
• How can we compute provenance in practice?
  • **ProvSQL** module for PostgreSQL, by Pierre Senellart et al.
  • Keeps track of provenance as a circuit
  • [https://github.com/PierreSenellart/provsql](https://github.com/PierreSenellart/provsql)

• How can we do **probabilistic query evaluation** via provenance?
  • ProvSQL is interfaced with **c2d, d4, and dsharp**
Provenance in practice

- How can we compute provenance in **practice**?
  - **ProvSQL** module for PostgreSQL, by Pierre Senellart et al.
    - Keeps track of provenance as a circuit
    - [https://github.com/PierreSenellart/provsql](https://github.com/PierreSenellart/provsql)
- How can we do **probabilistic query evaluation** via provenance?
  - ProvSQL is interfaced with **c2d**, **d4**, and **dsharp**
- How can we do **enumeration** via provenance?
  - See [Matthias’s talk](#) tomorrow
  - **Prototype**: [https://github.com/PoDMR/enum-spanner-rs](https://github.com/PoDMR/enum-spanner-rs)
Provenance in practice

- How can we compute provenance in **practice**?
  - **ProvSQL** module for PostgreSQL, by Pierre Senellart et al.
  - Keeps track of provenance as a circuit
  - [https://github.com/PierreSenellart/provsql](https://github.com/PierreSenellart/provsql)
- How can we do **probabilistic query evaluation** via provenance?
  - ProvSQL is interfaced with **c2d**, **d4**, and **dsharp**
- How can we do **enumeration** via provenance?
  - See Matthias’s talk tomorrow
  - **Prototype**: [https://github.com/PoDMR/enum-spanner-rs](https://github.com/PoDMR/enum-spanner-rs)
- Remark: missing studies of provenance notions used in the real world, e.g., “data lineage” used by Pachyderm
Provenance in theory

• **Confession:** As a theoretical topic, provenance feels *definitional*
  → Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...

• Which directions are *less definitional*?
Provenance in theory

- **Confession:** as a theoretical topic, provenance feels **definitional**
  → Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...

- Which directions are **less definitional?**
  - Using provenance for **computational tasks**
Provenance in theory

- **Confession**: as a theoretical topic, provenance feels *definitional*
  - Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...
- Which directions are *less definitional*?
  - Using provenance for *computational tasks*
    - We have seen two examples: probabilities and enumeration
    - In both cases, provenance *competes* against other approaches
    - Sometimes, provenance provides *new insights*
Provenance in theory

- **Confession:** as a theoretical topic, provenance feels **definitional**
  - Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...

- Which directions are **less definitional**?
  - Using provenance for **computational tasks**
    - We have seen two examples: probabilities and enumeration
    - In both cases, provenance **competes** against other approaches
    - Sometimes, provenance provides **new insights**
  - Showing **bounds** on provenance representations

Thanks for your attention!
• **Confession:** as a theoretical topic, provenance feels **definitional**
  → Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...
• Which directions are **less definitional**?
  • Using provenance for **computational tasks**
    • We have seen two examples: probabilities and enumeration
    • In both cases, provenance **competes** against other approaches
    • Sometimes, provenance provides **new insights**
  • Showing **bounds** on provenance representations
    • Connects to **knowledge compilation** work on circuit classes
    • Can be easier than **computational complexity** lower bounds
Provenance in theory

- **Confession**: as a theoretical topic, provenance feels **definitional**
  - Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...

- Which directions are **less definitional**?
  - Using provenance for **computational tasks**
    - We have seen two examples: probabilities and enumeration
    - In both cases, provenance **competes** against other approaches
    - Sometimes, provenance provides **new insights**
  - Showing **bounds** on provenance representations
    - Connects to **knowledge compilation** work on circuit classes
    - Can be easier than **computational complexity** lower bounds

Thanks for your attention!


Amarilli, Antoine and Mikaël Monet (2016). *Example of a naturally ordered semiring which is not an m-semiring*. 

http://math.stackexchange.com/questions/1966858.


– (2011b). “Provenance for aggregate queries”. In: PODS.

Bagan, Guillaume (2006). “MSO Queries on Tree Decomposable Structures Are Computable with Linear Delay”. In: CSL.


Capelli, Florent and Stefan Mengel (2019). “Tractable QBF by knowledge compilation”. In: STACS.


Green, Todd, Grigoris Karvounarakis, and Val Tannen (2007). “Provenance semirings”. In: *PODS*.


Jha, Abhay Kumar and Dan Suciu (2013). “Knowledge Compilation Meets Database Theory: Compiling Queries to Decision Diagrams”. In: *TCS* 52.3.


Monet, Mikaël (2020). “Solving a Special Case of the Intensional vs Extensional Conjecture in Probabilistic Databases”. In: *PODS*.

