Provenance and Probabilities in Relational Databases: From Theory to Practice
A review of provenance by Pierre Senellart, SIGMOD17

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
Table of Contents

1. What is data provenance
   - Motivation
   - Common types of provenance

2. Semirings - a mathematical model for provenance
   - Requirements
   - Mathematical formulation
   - Model to practice
   - Universality

3. Implementing provenance in DBs
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. Probabilistic Evaluation
   - Complexity concerns
   - Approaches to tackle complexity
Table of Contents

1. What is data provenance
   - Motivation
   - Common types of provenance

2. Semirings - a mathematical model for provenance
   - Requirements
   - Mathematical formulation
   - Model to practice
   - Universality

3. Implementing provenance in DBs
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. Probabilistic Evaluation
   - Complexity concerns
   - Approaches to tackle complexity
Table of Contents

1. What is data provenance
   - Motivation
   - Common types of provenance

2. Semirings - a mathematical model for provenance
   - Requirements
   - Mathematical formulation
   - Model to practice
   - Universality

3. Implementing provenance in DBs
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. Probabilistic Evaluation
   - Complexity concerns
   - Approaches to tackle complexity

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
Motivation

The main task in data management is query evaluation. Given a DB instance I, we want a subset of the information in I that adheres to some constraints.
The main task in data management is query evaluation.
Motivation

- The main task in data management is query evaluation.
- Given a DB instance I we want a subset of the information in I that adheres to some constraints.
Table: Personel

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
</tr>
<tr>
<td>3</td>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
</tr>
<tr>
<td>4</td>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
</tr>
<tr>
<td>5</td>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top_secret</td>
</tr>
<tr>
<td>6</td>
<td>Nancy</td>
<td>HR</td>
<td>Paris</td>
<td>restricted</td>
</tr>
<tr>
<td>7</td>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
</tr>
</tbody>
</table>
Example

Here are two queries
Example

Here are two queries

- $Q_1$ asks for all cities with at least 2 employees.
  
  $\textbf{Select } P1.\textit{city}$
  
  $\textbf{From} \text{ Personal } P1 , \text{ Personal } P2$
  
  $\textbf{Where} \ P1.\textit{city} = P2.\textit{city} \ \textbf{and} \ P1.\textit{id} < P2.\textit{id}$
Here are two queries

- \( Q_1 \) asks for all cities with at least 2 employees.
  \[
  \text{Select } P1.city \\
  \text{From } \text{Personal } P1, \text{Personal } P2 \\
  \text{Where } P1.city = P2.city \text{ and } P1.id < P2.id
  \]

- \( Q_2 \) asks for all cities exactly one employee
  \[
  \text{Select Distinct } city \ \text{From} \ \text{Personel} \\
  \text{Except} \\
  \text{Select } P1.city \\
  \text{From } \text{Personal } P1, \text{Personal } P2 \\
  \text{Where } P1.city = P2.city \text{ and } P1.id < P2.id
  \]
Example

These queries can be written in Relation Algebra (RA).

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
Example

These queries can be written in Relation Algebra (RA).

- SQL

```sql
Select P1.city
From Personal P1, Personal P2
Where P1.city = P2.city and P1.id < P2.id
```
These queries can be written in Relation Algebra (RA).

- **SQL**

  ```sql
  Select P1.city
  From Personal P1, Personal P2
  Where P1.city = P2.city and P1.id < P2.id
  ```

- **Relational Algebra**

  $$\pi_{\text{city}} \left( \sigma_{\text{id} < \text{id2}} \left( \pi_{\text{id}, \text{city}}(\text{Personel}) \bowtie \rho_{\text{id} \rightarrow \text{id2}} \left( \pi_{\text{id}, \text{city}}(\text{Personel}) \right) \right) \right)$$
In this case, the answer for $Q_1(Personal)$ is

| New York
| Paris
| Berlin |
Motivation

In this case, the answer for $Q_1(\textit{Personal})$ is

\begin{center}
\begin{tabular}{c}
Ney York \\
Paris \\
Berlin
\end{tabular}
\end{center}

$Q_2(\textit{Personel})$ is the empty table

\begin{center}
\begin{tabular}{c}
-
\end{tabular}
\end{center}
However, users often want to know more about a query than just the result. We could ask:
However, users often want to know more about a query than just the result. We could ask:

- why was a result obtained
Motivation

However, users often want to know more about a query than just the result. We could ask:

- why was a result obtained
- where values in the result come from
Motivation

However, users often want to know more about a query than just the result. We could ask:

- why was a result obtained
- where values in the result come from
- how the results would change if we only had parts of the DB
However, users often want to know more about a query than just the result. We could ask:

- why was a result obtained
- where values in the result come from
- how the results would change if we only had parts of the DB
- what security clearance is needed to see each result

...
Motivation

However, users often want to know more about a query than just the result. We could ask:

- why was a result obtained
- where values in the result come from
- how the results would change if we only had parts of the DB
- what security clearance is needed to see each result
- what is the probability of a result given a Probability distribution on the data
However, users often want to know more about a query than just the result. We could ask:

- why was a result obtained
- where values in the result come from
- how the results would change if we only had parts of the DB
- what security clearance is needed to see each result
- what is the probability of a result given a Probability distribution on the data
- and more ...
The idea behind Data Provenance is
The idea behind Data Provenance is

- Keeping some additional information
The idea behind Data Provenance is

- Keeping some additional information
- Allowing us to easily answer a large number of "Meta-questions"
**Table of Contents**

1. **What is data provenance**
   - Motivation
   - Common types of provenance

2. **Semirings - a mathematical model for provenance**
   - Requirements
   - Mathematical formulation
   - Model to practice
   - Universality

3. **Implementing provenance in DBs**
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. **Probabilistic Evaluation**
   - Complexity concerns
   - Approaches to tackle complexity

---

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
Boolean provenance attaches a boolean function $\varphi$ for
Boolean provenance attaches a boolean function $\varphi$ for every tuple of the DB $t_i$. 

---

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
Boolean provenance attaches a boolean function $\varphi$ for:

- every tuple of the DB $t_i$
- every tuple $Q_t$ in an output of a query $Q$
Boolean provenance

Each tuple in the DB $D$ is given a indicator for its existence.

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>$t_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>$t_1$</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>$t_2$</td>
</tr>
<tr>
<td>3</td>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>$t_3$</td>
</tr>
<tr>
<td>4</td>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>$t_4$</td>
</tr>
<tr>
<td>5</td>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top_secret</td>
<td>$t_5$</td>
</tr>
<tr>
<td>6</td>
<td>Nancy</td>
<td>HR</td>
<td>Paris</td>
<td>restricted</td>
<td>$t_6$</td>
</tr>
<tr>
<td>7</td>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>$t_7$</td>
</tr>
</tbody>
</table>
Boolean provenance

Each tuple in the DB $D$ is given a indicator for its existance

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>$\varphi_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>$t_1$</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>$t_2$</td>
</tr>
<tr>
<td>3</td>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>$t_3$</td>
</tr>
<tr>
<td>4</td>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>$t_4$</td>
</tr>
<tr>
<td>5</td>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top_secret</td>
<td>$t_5$</td>
</tr>
<tr>
<td>6</td>
<td>Nancy</td>
<td>HR</td>
<td>Paris</td>
<td>restricted</td>
<td>$t_6$</td>
</tr>
<tr>
<td>7</td>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>$t_7$</td>
</tr>
</tbody>
</table>

In this case $\varphi_{t_i} = 1_{t_i}$ We denote $\varphi_{t_i}$ as $t_i$ for short.
Boolean provenance

each tuple $q_i$ in $Q$ is given a formula
Boolean provenance

each tuple $q_i$ in $Q$ is given a formula

$$\varphi_{q_i} : \{D' \mid D' \subseteq D\} \Rightarrow \{\top, \bot\}$$
Boolean provenance

Each tuple $q_i$ in $Q$ is given a formula

$$\varphi_{q_i} : \{D' \mid D' \subseteq D\} \Rightarrow \{\top, \bot\}$$

That only returns True if $q_i$ exists when evaluating $D'$
Boolean provenance

each tuple $q_i$ in $Q$ is given a formula

$$\varphi_{q_i} : \{D' \mid D' \subseteq D\} \Rightarrow \{\top, \bot\}$$

that only returns True if $q_i$ exists when evaluating $D'$

$$\varphi_{q_i}(D') = \top \iff q_i \in Q(D')$$
Boolean provenance

$Q_1$ becomes
Boolean provenance

$Q_1$ becomes

<table>
<thead>
<tr>
<th>Location</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>$t_1 \land t_2$</td>
</tr>
<tr>
<td>Paris</td>
<td>$(t_3 \land t_5) \lor (t_3 \land t_6) \lor (t_5 \land t_6)$</td>
</tr>
<tr>
<td>Berlin</td>
<td>$t_4 \land t_7$</td>
</tr>
</tbody>
</table>
Security provenance

Let's say our DB has different levels of information access
Security provenance

Let's say our DB has different levels of information access:
- Security provenance lets us know which query results to show to each user.
Security provenance

Let's say our DB has different levels of information access

- security provenance lets us know which query results to show to each user

In our example, the security clearance levels are

unclassified < restricted < confidential < secret < top secret
We assume each tuple in the DB has a security clearance.

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
</tr>
<tr>
<td>3</td>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
</tr>
<tr>
<td>4</td>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
</tr>
<tr>
<td>5</td>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top_secret</td>
</tr>
<tr>
<td>6</td>
<td>Nancy</td>
<td>HR</td>
<td>Paris</td>
<td>restricted</td>
</tr>
<tr>
<td>7</td>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
</tr>
</tbody>
</table>
Security provenance

with security provenance, each tuple in $Q(D)$ will have the minimal security clearance to view it.
with security provenance, each tuple in \( Q(D) \) will have the minimal security clearance to view it.

<table>
<thead>
<tr>
<th>City</th>
<th>Clearance</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>
Security provenance

Security provenance is computationally efficient
Security provenance

Security provenance is computationally efficient

- We do not need to filter the DB based on the user’s clearance and then rerun the query for all clearance levels.
Security provenance is computationally efficient

- We do not need to filter the DB based on the user’s clearance and then rerun the query for all clearance levels.
- Instead, we can compute the query once for all users and filter the results based on the provenance.
"Why" provenance

Why provenance tries to explain why a certain tuple appears in a query result.
Why provenance tries to explain why a certain tuple appears in a query result.

- Allows one to know the origin of a query tuple to further analyze its validity and relevance.
"Why" provenance

Why provenance tries to explain why a certain tuple appears in a query result.

- Allows one to know the origin of a query tuple to further analyze its validity and relevance.
- Extremely usefull in complex DBs like biological DBs where the causes of a derivation add more information to the result.
"Why" provenance

In why provenance
In why provenance

- we attach to each $q_i \in Q$ the set of combination of tuples needed for $q_i$ to exist.
"Why" provenance

In why provenance

- we attach to each $q_i \in Q$ the set of combination of tuples needed for $q_i$ to exist.
- these combinations are sometimes called \textbf{Witnesses}.
In 'why' provenance
- we attach to each $q_i \in Q$ the set of combination of tuples needed for $q_i$ to exist.
- these combinations are sometimes called **Witnesses**.

<table>
<thead>
<tr>
<th>City</th>
<th>Witnesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ney York</td>
<td>${{t_1, t_2}}$</td>
</tr>
<tr>
<td>Paris</td>
<td>${{t_3, t_5}, {t_3, t_6}, {t_5, t_6}, {t_3, t_5, t_6}}$</td>
</tr>
<tr>
<td>Berlin</td>
<td>${{t_5, t_7}}$</td>
</tr>
</tbody>
</table>
"Why" provenance

Why provenance seems similar to Boolean provenance.
"Why" provenance

Why provenance seems similar to Boolean provenance.

- In fact, it strictly generalizes it.
"Why" provenance

Why provenance seems similar to Boolean provenance.

- In fact, it strictly generalizes it.
- They seem equivalent in small examples, but Boolean provenance needs a subset of the DB to compute the answer $\varphi_{q_i}(D')$. 
"Why" provenance

Why provenance seems similar to Boolean provenance.

- In fact, it strictly generalizes it.
- They seem equivalent in small examples, but Boolean provenance needs a subset of the DB to compute the answer $\varphi_{q_i}(D')$.
- Why provenance "decomposes" the logic behind boolean provenance and gives as an understanding of the topology of the subsets of $D$ that generate $q_i$. 

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
Table of Contents

1. What is data provenance
   - Motivation
   - Common types of provenance

2. Semirings - a mathematical model for provenance
   - Requirements
   - Mathematical formulation
   - Model to practice
   - Universality

3. Implementing provenance in DBs
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. Probabilistic Evaluation
   - Complexity concerns
   - Approaches to tackle complexity

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
Table of Contents

1. What is data provenance
   - Motivation
   - Common types of provenance

2. Semirings - a mathematical model for provenance
   - Requirements
     - Mathematical formulation
     - Model to practice
     - Universality

3. Implementing provenance in DBs
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. Probabilistic Evaluation
   - Complexity concerns
   - Approaches to tackle complexity
What would we want out of a provenance model?

- Generalizes most desired types of provenance
- Allows compact representation
- Computations track query evaluations closely
- Takes advantage of similarities between queries and between provenances
- Admits to batch computations and decompositions
What would we want out of a provenance model?

- Generalizes most desired types of provenance
What would we want out of a provenance model?

- Generalizes most desired types of provenance
- Allows compact representation
What would we want out of a provenance model?

- Generalizes most desired types of provenance
- allows compact representation
- computations track query evaluations closely
What would we want out of a provenance model?

- Generalizes most desired types of provenance
- Allows compact representation
- Computations track query evaluations closely
- Takes advantage of similarities between queries and between provenances
What would we want out of a provenance model?

- Generalizes most desired types of provenance
- allows compact representation
- computations track query evaluations closely
- takes advantage of similarities between queries and between provenances
  - admits to batch computations and decompositions
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Subsections</th>
</tr>
</thead>
</table>
| 1       | What is data provenance | - Motivation  
- Common types of provenance |
| 2       | Semirings - a mathematical model for provenance | - Requirements  
- Mathematical formulation  
- Model to practice  
- Universality |
| 3       | Implementing provenance in DBs | - Relation Algebra  
- Overcoming positivity concerns  
- Limitations and extensions  
- Representations - Circuits |
| 4       | Probabilistic Evaluation | - Complexity concerns  
- Approaches to tackle complexity |
A semiring is a tuple of the form \((K, \oplus, \otimes, 0, 1)\)
A semiring is a tuple of the form \((K, \oplus, \otimes, 0, 1)\)

- \(K\) is a set
A semiring is a tuple of the form \((K, \oplus, \otimes, 0, 1)\)

- \(K\) is a set
- \(\oplus\) is an associative and commutative operator with identity 0
A semiring is a tuple of the form \((K, \oplus, \otimes, 0, 1)\)

- \(K\) is a set
- \(\oplus\) is an associative and commutative operator with identity \(0\)
  - \(k_1 \oplus 0 = 0 \oplus k_1 = k_1\)
Semiring

A semiring is a tuple of the form \((K, \oplus, \otimes, 0, 1)\)

- \(K\) is a set
- \(\oplus\) is an associative and commutative operator with identity 0
  - \(k_1 \oplus 0 = 0 \oplus k_1 = k_1\)
- \(\otimes\) is associative and commutative operator with identity 1

\(K\) is a set
\(\oplus\) is an associative and commutative operator with identity 0
- \(k_1 \oplus 0 = 0 \oplus k_1 = k_1\)
\(\otimes\) is associative and commutative operator with identity 1
A semiring is a tuple of the form \( (K, \oplus, \otimes, 0, 1) \)

- \( K \) is a set
- \( \oplus \) is an associative and commutative operator with identity \( 0 \)
  - \( k_1 \oplus 0 = 0 \oplus k_1 = k_1 \)
- \( \otimes \) is associative and commutative operator with identity \( 1 \)
  - \( k_1 \otimes 1 = 1 \otimes k_1 = k_1 \)
A semiring is a tuple of the form \((K, \oplus, \otimes, 0, 1)\)

- \(K\) is a set
- \(\oplus\) is an associative and commutative operator with identity \(0\)
  - \(k_1 \oplus 0 = 0 \oplus k_1 = k_1\)
- \(\otimes\) is associative and commutative operator with identity \(1\)
  - \(k_1 \otimes 1 = 1 \otimes k_1 = k_1\)
- \(\otimes\) distributes over \(\oplus\) and \(0\) annihilates \(\otimes\)
A semiring is a tuple of the form \((K, \oplus, \otimes, 0, 1)\)

- \(K\) is a set
- \(\oplus\) is an associative and commutative operator with identity 0
  - \(k_1 \oplus 0 = 0 \oplus k_1 = k_1\)
- \(\otimes\) is associative and commutative operator with identity 1
  - \(k_1 \otimes 1 = 1 \otimes k_1 = k_1\)
- \(\otimes\) distributes over \(\oplus\) and 0 annihilates \(\otimes\)
  - \(a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)\)
A semiring is a tuple of the form \((K, \oplus, \otimes, 0, 1)\)

- \(K\) is a set
- \(\oplus\) is an associative and commutative operator with identity \(0\)
  - \(k_1 \oplus 0 = 0 \oplus k_1 = k_1\)
- \(\otimes\) is associative and commutative operator with identity \(1\)
  - \(k_1 \otimes 1 = 1 \otimes k_1 = k_1\)
- \(\otimes\) distributes over \(\oplus\) and \(0\) annihilates \(\otimes\)
  - \(a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)\)
  - \(0 \otimes k_1 = 0\)

This is similar to a field, except without the need for inverse element
Table of Contents

1. What is data provenance
   - Motivation
   - Common types of provenance

2. Semirings - a mathematical model for provenance
   - Requirements
   - Mathematical formulation
   - Model to practice
   - Universality

3. Implementing provenance in DBs
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. Probabilistic Evaluation
   - Complexity concerns
   - Approaches to tackle complexity

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
How does this model encapsulates the provenance schemes we saw before?
The security semiring is

$\langle \text{Clearance}, \text{min}, \text{max}, \text{top\_secret}, \text{unclassified} \rangle$

where \textit{Clearance} is the security partial order we saw before.
The security semiring is

\[(Clearance, \text{min}, \text{max}, \text{top\_secret}, \text{unclassified})\]

where \text{Clearance} is the security partial order we saw before.

\[\text{restricted } \otimes \text{secret} = \text{max}\{\text{restricted, secret}\} = \text{secret}\]
security semiring

The security semiring is

\((\text{Clearance}, \min, \max, \text{top}\_\text{secret}, \text{unclassified})\)

where \(\text{Clearance}\) is the security partial order we saw before.

- restricted \(\otimes\) secret = \(\max\{\text{restricted, secret}\}\) = secret
- This semiring makes more sense when compared to the boolean case.
The security semiring is

\[(\text{Clearance}, \text{min}, \text{max}, \text{top}_\text{secret}, \text{unclassified})\]

where \text{Clearance} is the security partial order we saw before.

- restricted \otimes \text{secret} = \text{max}\{\text{restricted, secret}\} = \text{secret}
- This semiring makes more sense when compared to the boolean case.
  - \text{max} \sim \wedge, \text{min} \sim \vee
security semiring

The security semiring is

\((\text{Clearance}, \min, \max, \text{top}_\text{secret}, \text{unclassified})\)

where \(\text{Clearance}\) is the security partial order we saw before.

- restricted \(\otimes\) secret = \(\max\{\text{restricted}, \text{secret}\}\) = secret
- This semiring makes more sense when compared to the boolean case.
  - \(\max \sim \land\), \(\min \sim \lor\)
  - if a \(q_1\) required both \(t_1\) and \(t_2\) then it needs the maximal clearance of the two to be derived
security semiring

The security semiring is

\((\text{Clearance}, \text{min}, \text{max}, \text{top-secret}, \text{unclassified})\)

where \text{Clearance} is the security partial order we saw before.

- restricted \(\otimes\) secret = max\{restricted, secret\} = secret
- This semiring makes more sense when compared to the boolean case.
  - \text{max} \sim \wedge, \text{min} \sim \vee
  - if a \(q_1\) required both \(t_1\) and \(t_2\) then it needs the maximal clearance of the two to be derived
- note that this extends to any full ordering of elements.
The security semiring is

\[(\text{Clearance, min, max, top\_secret, unclassified})\]

where \textit{Clearance} is the security partial order we saw before.

- restricted $\otimes$ secret $= \max\{\text{restricted, secret}\} = \text{secret}$
- This semiring makes more sense when compared to the boolean case.
  - $\max \sim \wedge$, $\min \sim \vee$
  - if a $q_1$ required both $t_1$ and $t_2$ then it needs the maximal clearance of the two to be derived
- note that this extends to any full ordering of elements.
- can also be extended to lattices (partial orders)
positive Boolean function semiring

The positive Boolean function semiring

\[(F, \lor, \land, \bot, \top)\]

where \(F\) is the family of positive boolean functions.
The positive Boolean function semiring

\[(F, \vee, \wedge, \bot, \top)\]

where \(F\) is the family of positive boolean functions.

- \(f_1 \oplus f_2\) translates to \(f_1 \vee f_2\) etc...
The positive Boolean function semiring

\[(F, \lor, \land, \bot, \top)\]

where \(F\) is the family of positive boolean functions.

- \(f_1 \oplus f_2\) translates to \(f_1 \lor f_2\) etc...
- like boolean algebra
positive Boolean function semiring

The positive Boolean function semiring

$$(F, \lor, \land, \perp, \top)$$

where $F$ is the family of positive boolean functions.

- $f_1 \oplus f_2$ translates to $f_1 \lor f_2$ etc...
- like boolean algebra
- this gives us the boolean provenance we wanted before (up to negation).
The positive Boolean function semiring

\[(F, \lor, \land, \bot, \top)\]

where \(F\) is the family of positive boolean functions.

- \(f_1 \oplus f_2\) translates to \(f_1 \lor f_2\) etc...
- like boolean algebra
- this gives us the boolean provenance we wanted before (up to negation).
- more on negation later
New operation - Deep Union

Now things are going to get more tricky. We need a new operation, Deep union.
Now things are going to get more tricky. We need a new operation, Deep union. Let

\[ X = \{1, \ldots, 10\} \]
\[ A = \{\{1, 2\}, \{3\}\} \]
\[ B = \{\{3\}, \{4, 5\}\} \]
Now things are going to get more tricky. We need a new operation, Deep union. Let

\[ X = \{1, ..., 10\} \]
\[ A = \{\{1, 2\}, \{3\}\} \]
\[ B = \{\{3\}, \{4, 5\}\} \]

We get

\[ A \uplus B = \{\{1, 2, 3\}, \{3\}, \{1, 2, 4, 5\}, \{3, 4, 5\}\} \]
A more formal definition
A more formal definition

**Definition (DeepUnion)**

Given two sets of subsets in the same world \( A, B \in P(P(X)) \), the Deep Union is defined to be

\[
A \cup B = \{ a \cup b \mid a \in A, b \in B \}
\]
A more formal definition

**Definition (DeepUnion)**

Given two sets of subsets in the same world $A, B \in P(P(X))$, the Deep Union is defined to be

$$A \cup B = \{a \cup b \mid a \in A, b \in B\}$$

Basically all the pairwise unions of sets in $A$ and $B$
The why semiring is defined as \((P(P(T)), \cup, \uplus, \emptyset, \{\emptyset\})\) Think in terms witnesses.
why - semiring

The why semiring is defined as \((P(P(T)), \cup, \sqcup, \emptyset, \{\emptyset\})\) Think in terms witnesses.

- We have \(A = \{\{1, 2\}, \{3\}\}, B = \{\{3\}, \{4, 5\}\}\) sets of necessary combination of witnesses for events \(\alpha\) and \(\beta\) respectively.
why - semiring

The why semiring is defined as \((P(P(T)), \cup, \bigcup, \emptyset, \{\emptyset\})\) Think in terms witnesses.

- We have \(A = \{\{1, 2\}, \{3\}\}, B = \{\{3\}, \{4, 5\}\}\) sets of necessary combination of witnesses for events \(\alpha\) and \(\beta\) respectively
- if \(\gamma\) is an event that needs witnesses of \(\alpha\ or \beta\) then any \(a_i, b_i\) is good.
The why semiring is defined as \((P(P(T)), \cup, \uplus, \emptyset, \{\emptyset}\)) Think in terms witnesses.

- We have \(A = \{\{1, 2\}, \{3\}\}, B = \{\{3\}, \{4, 5\}\}\) sets of necessary combination of witnesses for events \(\alpha\) and \(\beta\) respectively.
- If \(\gamma\) is an event that needs witnesses of \(\alpha\) or \(\beta\) then any \(a_i, b_i\) is good.
- Therefore, the sets of witnesses combinations we need is \(A \cup B\).
why - semiring

The why semiring is defined as \((P(P(T)), \cup, \cup, \emptyset, \{\emptyset\})\) Think in terms witnesses.

- We have \(A = \{\{1, 2\}, \{3\}\}, B = \{\{3\}, \{4, 5\}\}\) sets of necessary combination of witnesses for events \(\alpha\) and \(\beta\) respectively.
- If \(\gamma\) is an event that needs witnesses of \(\alpha\) or \(\beta\) then any \(a_i, b_i\) is good.
- Therefore, the sets of witnesses combinations we need is \(A \cup B\).
- If \(\gamma\) is an event that needs witnesses of \(\alpha\) and \(\beta\), then any set that has a witness set \(a \in A\) and has a witness set \(b \in B\) suffices.
why - semiring

The why semiring is defined as \((P(P(T)), \cup, \cup, \emptyset, \{\emptyset\})\) Think in terms witnesses.

- We have \(A = \{\{1, 2\}, \{3\}\}, B = \{\{3\}, \{4, 5\}\}\) sets of necessary combination of witnesses for events \(\alpha\) and \(\beta\) respectively
- if \(\gamma\) is an event that needs witnesses of \(\alpha\) or \(\beta\) then any \(a_i, b_i\) is good.
- therefore, the sets of witnesses combinations we need is \(A \cup B\)
- if \(\gamma\) is an event that needs witnesses of \(\alpha\) and \(\beta\), then any set that has a witness set \(a \in A\) and has a witness set \(b \in B\) suffices.
- therefore, we need the deep union of \(A\) and \(B\)
Why - semiring

But do we really need all subsets of witnesses? A $\mathcal{D} \mathcal{B} = \{ f_1; f_2; f_3; f_3; f_1; f_2; f_4; f_5 \}$

We need only minimal sets from $\mathcal{D} \mathcal{B}$. $f_3; f_4; f_5$ is redundant.

Proposed solution: $\mathcal{D} \mathcal{B} = \text{minset}(\mathcal{D} \mathcal{B})$

However, this operation does not distribute with $\mathcal{A} \mathcal{D} = [\mathcal{A} \mathcal{B}]$

This problem can be solved to avoid unnecessary bloating in why provenance.
But do we really need all subsets of witnesses?
But do we really need all subsets of witnesses?

\[ A \cup B = \{\{1, 2, 3\}, \{3\}, \{1, 2, 4, 5\}, \{3, 4, 5\}\} \]
But do we really need all subsets of witnesses?

\[ A \cup B = \{\{1, 2, 3\}, \{3\}, \{1, 2, 4, 5\}, \{3, 4, 5\}\} \]

We need only minimal sets from \( A \cup B \)
But do we really need all subsets of witnesses?
- \( A \cup B = \{\{1, 2, 3\}, \{3\}, \{1, 2, 4, 5\}, \{3, 4, 5\}\} \)
- We need only minimal sets from \( A \cup B \)
- \( \{3, 4, 5\} \) is redundant.
But do we really need all subsets of witnesses?

\[ A \cup B = \{ \{1, 2, 3\}, \{3\}, \{1, 2, 4, 5\}, \{3, 4, 5\} \} \]

We need only minimal sets from \( A \cup B \)

\{3, 4, 5\} is redundant.

Proposed solution: \( \otimes = \text{minset}(A \cup B) \)
But do we really need all subsets of witnesses?

\[ A \uplus B = \{\{1, 2, 3\}, \{3\}, \{1, 2, 4, 5\}, \{3, 4, 5\}\} \]

We need only minimal sets from \( A \uplus B \)

\{3, 4, 5\} is redundant.

Proposed solution: \( \otimes = \text{minset}(A \uplus B) \)

However, this operation does not distribute with \( \oplus = \cup \)
But do we really need all subsets of witnesses?

\[ A \cup B = \{\{1, 2, 3\}, \{3\}, \{1, 2, 4, 5\}, \{3, 4, 5\}\} \]

We need only minimal sets from \( A \cup B \)

\( \{3, 4, 5\} \) is redundant.

Proposed solution : \( \otimes = \text{minset}(A \cup B) \)

however, this operation does not distribute with \( \oplus = \cup \)

this problem can be solved to avoid unnecessary bloating in why provenance
So far:
So far:

- we found a mathematical model that generalises a lot of desired provenances.
So far:

- we found a mathematical model that generalises a lot of desired provenances.
- However, keeping a different semiring for each tuple in the original DB and in query results (and computing them for each intermediate result) is too much.
So far:

- we found a mathematical model that generalises a lot of desired provenances.
- However, keeping a different semiring for each tuple in the original DB and in query results (and computing them for each intermediate result) is too much.
- Luckily, we have a universal semiring that can be morphed into any semiring we desire.
# Table of Contents

1. **What is data provenance**
   - Motivation
   - Common types of provenance

2. **Semirings - a mathematical model for provenance**
   - Requirements
   - Mathematical formulation
   - Model to practice
   - Universality

3. **Implementing provenance in DBs**
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. **Probabilistic Evaluation**
   - Complexity concerns
   - Approaches to tackle complexity

---

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
The universal semiring

Introducing the int polynomial semiring

$$(\mathbb{N}[X], +, \times, 0, 1)$$

Note that it is a set of polynomials with $|X|$ variables.

$X = x_1, x_2, \ldots$

**Theorem**

$\mathbb{N}[X]$ is universal w.r.t any semiring $K$ This means that there a semiring homomorphism $h: \mathbb{N}[X] \rightarrow K$ for any $K$
examples of universality

Let's see the homomorphism between the universal semiring and the semirings we know.
Examples of universality

Let's see the homomorphism between the universal semiring and the semirings we know. Remember that a homomorphism needs to only be defined on the generating set of $\mathbb{N}[X]$. 
Examples of universality

The boolean algebra semiring

\[ h : \mathbb{N}[X] \rightarrow \{ \top, \bot \} \]
Examples of universality

The boolean algebra semiring

\[ h : \mathbb{N}[X] \rightarrow \{ \top, \bot \} \]

- \( h(1) = h(0) = \top \)
Examples of universality

The boolean algebra semiring

\[ h : \mathbb{N}[X] \rightarrow \{\top, \bot\} \]

- \( h(1) = h(0) = \top \)
- \( h(x_i) = \bot \)
Examples of universality

The boolean algebra semiring

\[ h : \mathbb{N}[X] \rightarrow \{ \top, \bot \} \]

- \[ h(1) = h(0) = \top \]
- \[ h(x_i) = \bot \]
- from these we get that \( h(p) = \top \) iff \( p \) has a constant monom
Example

So

\[ h((1 + x)(x + y) + 2(3x + z)) = x^2 + xy + 7x + y + 2z = \perp \]

since it has no constant terms
The positive boolean function semirings.

\[ h : \mathbb{N}[X] \rightarrow \{ f : X \rightarrow \{\top, \bot\} \} \]
Examples of universality

The positive boolean function semirings.

\[ h : \mathbb{N}[X] \rightarrow \{ f : X \rightarrow \{\top, \bot\} \} \]

- \( h(1) = h(n) = \top \)
Examples of universality

The positive boolean function semirings.

\[ h : \mathbb{N}[X] \rightarrow \{ f : X \rightarrow \{\top, \bot\} \} \]

- \( h(1) = h(n) = \top \)
- \( h(0) = \bot \)
Examples of universality

The positive boolean function semirings.

\[ h : \mathbb{N}[X] \rightarrow \{ f : X \rightarrow \{ \top, \bot \} \} \]

- \( h(1) = h(n) = \top \)
- \( h(0) = \bot \)
- \( h(x_i) = x_i \)
Example

So

\[ h((x^3 + 5y)t + z + 2) = (x \lor (\top \land y) \land t \lor z \lor \bot) = (x \lor y) \land t \lor z \]

Just like in boolean algebra
The universal semiring

The idea behind the proof will help us see the limitations of semirings.
The idea behind the proof will help as see the limitations of semirings.

**Proof - well the idea.**
The universal semiring

The idea behind the proof will help us see the limitations of semirings.

Proof - well the idea.

- We have a distinct element in \( \mathbb{N}[X] \) for every distinct series of operations \(+, \times\) on generating elements, up to distribution.
The universal semiring

The idea behind the proof will help as see the limitations of semirings.

**Proof - well the idea.**

- We have a distinct element in $\mathbb{N}[X]$ for every distinct series of operations $+$, $\times$ on generating elements, up to distribution.
- We have unlimited generating variables
The universal semiring

The idea behind the proof will help as see the limitations of semirings.

Proof - well the idea.

- We have a distinct element in \( \mathbb{N}[X] \) for every distinct series of operations \(+, \times\) on generating elements, up to distribution.
- We have unlimited generating variables
- equality \( w.r.t \) to distribution can be solved by limiting ourselves to standard form polynomials
Utilizing universality

Now, we can use universality to out advantage.
Now, we can use universality to out advantage.

- For each \( q_i \in Q \)
Now, we can use universality to out advantage.

- For each $q_i \in Q$
  - calculate its provenance w.r.t $\mathbb{N}[X]$, $p_{q_i}$ once.
Utilizing universality

Now, we can use universality to our advantage.

For each $q_i \in Q$

- calculate its provenance w.r.t $\mathbb{N}[X]$, $p_{q_i}$ once.
- whenever we need a specific provenance, use $h(p_{q_i})$ for the relevant $h$
Now, we can use universality to out advantage.

- For each $q_i \in Q$
  - calculate its provenance w.r.t $\mathbb{N}[X]$, $p_{q_i}$ once.
  - whenever we need a specific provenance, use $h(p_{q_i})$ for the relevant $h$
  - Since polynomial factorization and decompositions are studied heavily, they are a good tool to use computationally.
Table of Contents

1. What is data provenance
   - Motivation
   - Common types of provenance

2. Semirings - a mathematical model for provenance
   - Requirements
   - Mathematical formulation
   - Model to practice
   - Universality

3. Implementing provenance in DBs
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. Probabilistic Evaluation
   - Complexity concerns
   - Approaches to tackle complexity
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>What is data provenance</td>
</tr>
<tr>
<td></td>
<td>Motivation</td>
</tr>
<tr>
<td></td>
<td>Common types of provenance</td>
</tr>
<tr>
<td>2</td>
<td>Semirings - a mathematical model for provenance</td>
</tr>
<tr>
<td></td>
<td>Requirements</td>
</tr>
<tr>
<td></td>
<td>Mathematical formulation</td>
</tr>
<tr>
<td></td>
<td>Model to practice</td>
</tr>
<tr>
<td></td>
<td>Universality</td>
</tr>
<tr>
<td>3</td>
<td>Implementing provenance in DBs</td>
</tr>
<tr>
<td></td>
<td>Relation Algebra</td>
</tr>
<tr>
<td></td>
<td>Overcoming positivity concerns</td>
</tr>
<tr>
<td></td>
<td>Limitations and extensions</td>
</tr>
<tr>
<td></td>
<td>Representations - Circuits</td>
</tr>
<tr>
<td>4</td>
<td>Probabilistic Evaluation</td>
</tr>
<tr>
<td></td>
<td>Complexity concerns</td>
</tr>
<tr>
<td></td>
<td>Approaches to tackle complexity</td>
</tr>
</tbody>
</table>
But how do we compute the provenance of the query?
But how do we compute the provenance of the query?

- Relational Algebra!
But how do we compute the provenance of the query?

- Relational Algebra!
- All SQL queries can be decomposed into relational algebra formulas.
But how do we compute the provenance of the query?

- Relational Algebra!
- All SQL queries can be decomposed into relational algebra formulas.
- So we can define semiring operations on SQL operations
RA on provenance tokens

Relation algebra operation on a DB with provenance annotation in $(K, \oplus, \otimes, 0, 1)$
RA on provenance tokens

Relation algebra operation on a DB with provenance annotation in $(K, \oplus, \otimes, 0, 1)$

- selection and projection do not affect provenance
RA on provenance tokens

Relation algebra operation on a DB with provenance annotation in \((K, \oplus, \otimes, 0, 1)\)

- selection and projection do not affect provenance
- the provenance of a unioned tuples is \(\oplus\)-ed
RA on provenance tokens

Relation algebra operation on a DB with provenance annotation in $(K, \oplus, \otimes, 0, 1)$

- selection and projection do not affect provenance
- the provenance of a unioned tuples is $\oplus$-ed
- the provenance of a tuples combined by cartesian product is $\otimes$-ed
Relation algebra operation on a DB with provenance annotation in \((K, \oplus, \otimes, 0, 1)\)

- selection and projection do not affect provenance
- the provenance of a unioned tuples is \(\oplus\)-ed
- the provenance of a tuples combined by cartesian product is \(\otimes\)-ed
- projection does not affect provenance unless identical tuples undergo *duplicate elimination* In that case, we \(\oplus\) the results. This is because duplicate elimination is identical to union.
RA on provenance tokens

Relation algebra operation on a DB with provenance annotation in 
$(K, \oplus, \otimes, 0, 1)$

- selection and projection do not affect provenance
- the provenance of a unioned tuples is $\oplus$-ed
- the provenance of a tuples combined by cartesian product is $\otimes$-ed
- projection does not affect provenance unless identical tuples undergo duplicate elimination. In that case, we $\oplus$ the results. This is because duplicate elimination is identical to union.
- Unfortunately, semiring operation cannot express negation.
RA on provenance tokens

Relation algebra operation on a DB with provenance annotation in $(K, \oplus, \otimes, 0, 1)$

- selection and projection do not affect provenance
- the provenance of a unioned tuples is $\oplus$-ed
- the provenance of a tuples combined by cartesian product is $\otimes$-ed
- projection does not affect provenance unless identical tuples undergo *duplicate elimination* In that case, we $\oplus$ the results. This is because duplicate elimination is identical to union.
- Unfortunately, semiring operation cannot express negation.
- This means that the semiring model holds only for the positive fraction of relational Algebra.
RA on provenance tokens

Relation algebra operation on a DB with provenance annotation in $(K, \oplus, \otimes, 0, 1)$

- selection and projection do not affect provenance
- the provenance of a unioned tuples is $\oplus$-ed
- the provenance of a tuples combined by cartesian product is $\otimes$-ed
- projection does not affect provenance unless identical tuples undergo *duplicate elimination* In that case, we $\oplus$ the results. This is because duplicate elimination is identical to union.
- Unfortunately, semiring operation cannot express negation.
- This means that the semiring model holds only for the positive fraction of relational Algebra.
- We will see a solution for this later on.
Example

Lets us look at \((Clearance, min, max, top\_secret, unclassified)\). This means that each tuple has a security clearance.
Example

Let us look at \((\text{Clearance}, \text{min}, \text{max}, \text{top\_secret}, \text{unclassified})\). This means that each tuple has a security clearance.

- Selection and projection do not change the security clearance of the tuples.
Example

Let us look at \((\text{Clearance}, \text{min}, \text{max}, \text{top\_secret}, \text{unclassified})\) . This means that each tuple has a security clearance.

- selection and projection do not change the security clearance of the tuples.
- if a tuple undergoes union with itself, the tuple can be gotten by the minimal security clearance of the original tuples, so we take \(\oplus = \text{min}\)
Example

Let us look at \((\text{Clearance}, \text{min}, \text{max}, \text{top\_secret}, \text{unclassified})\). This means that each tuple has a security clearance.

- selection and projection do not change the security clearance of the tuples.
- if a tuple undergoes union with itself, the tuple can be gotten by the minimal security clearance of the original tuples, so we take \(\oplus = \text{min}\)
- if a tuple is received from the cartesian product of an unclassified and top-secret tuples, it has top secret information in it. therefore we take \(\otimes = \text{max}\) of the clearances.
Example

Let's us look at \((\text{Clearance}, \min, \max, \text{top_secret}, \text{unclassified})\). This means that each tuple has a security clearance.

- selection and projection do not change the security clearance of the tuples.
- if a tuple undergoes union with itself, the tuple can be gotten by the minimal security clearance of the original tuples, so we take \(\oplus = \min\)
- if a tuple is received from the cartesian product of an unclassified and top-secret tuples, it has top secret information in it. therefore we take \(\otimes = \max\) of the clearances.
- projection (without duplicate elimination) means taking a subset of the information of the tuple, hence that information retains its clearance level.
The intuition behind semirings

The reasons why semirings work so well for relational databases is
The intuition behind semirings

The reasons why semirings work so well for relational databases is

- $\oplus$ and $\otimes$ operations map to aggregation and merging of data.
The reasons why semirings work so well for relational databases is

- $\oplus$ and $\otimes$ operations map to aggregation and merging of data.
- These form the backbone of RA, up to negation.
The intuition behind semirings

The reasons why semirings work so well for relational databases is:

- \( \oplus \) and \( \otimes \) operations map to aggregation and merging of data.
- These form the backbone of RA, up to negation.
- we shall handle the problem of negation NOW
# Table of Contents

1. **What is data provenance**  
   - Motivation  
   - Common types of provenance

2. **Semirings - a mathematical model for provenance**  
   - Requirements  
   - Mathematical formulation  
   - Model to practice  
   - Universality

3. **Implementing provenance in DBs**  
   - Relation Algebra  
   - Overcoming positivity concerns  
   - Limitations and extensions  
   - Representations - Circuits

4. **Probabilistic Evaluation**  
   - Complexity concerns  
   - Approaches to tackle complexity

---

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
m-semirings
m-semirings

Semirings cannot capture negative operations.
m-semirings

- Semirings cannot capture negative operations.
- However, we can extend them to do so.
Semirings cannot capture negative operations.

However, we can extend them to do so.

the negative extensions of semirings are called semirings with monus (minus)
m-semirings

- Semirings cannot capture negative operations.
- However, we can extend them to do so.
- the negative extensions of semirings are called semirings with monus (minus)
- m-semirings for short
m-semirings

- Semirings cannot capture negative operations.
- However, we can extend them to do so.
- The negative extensions of semirings are called semirings with monus (minus)
- m-semirings for short
- Most semirings can be extended (but not all of them!)
m-semirings

First the intuition, using information.
First the intuition, using information.

**Definition**

Definition: m-semirings A semiring attached with an information separation operator such that
m-semirings

First the intuition, using information.

**Definition**

Definition: m-semirings A semiring attached with an information separation operator such that

- you can separate information exactly, and add and subtract iteratively.

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
First the intuition, using information.

**Definition**

Definition: m-semirings A semiring attached with an information separation operator such that

- you can separate information exactly, and add and subtract iteratively.
- you can subtract information step by step, or all at once
m-semirings

First the intuition, using information.

**Definition**

Definition: m-semirings A semiring attached with an information separation operator such that

- you can separate information exactly, and add and subtract iteratively.
- you can subtract information step by step, or all at once
- you can’t have zero information
Now the formal definition

Definition: m-semirings

A semiring attached with a monus operator \( \ominus \) such that

\[
\begin{align*}
(a \ominus (b \ominus a)) &= b \\
(a \ominus (b \ominus c)) &= a \ominus (b \ominus c)
\end{align*}
\]

\[
\begin{align*}
(a \ominus a) &= 0 \\
(0 \ominus a) &= 0
\end{align*}
\]
m-semirings

Now the formal definition

**Definition**

Definition: m-semirings A semiring attached with a monus operator $\ominus$ such that

- $(a \ominus (b \ominus a)) = b \ominus (a \ominus b) = a \ominus (b \ominus c) = a \ominus (b \ominus (b \ominus c))$
- $a \ominus a = 0 \ominus a = 0$
m-semirings

Now the formal definition

**Definition**

Definition: m-semirings A semiring attached with a monus operator $\ominus$ such that

- $a \oplus (b \ominus a) = b \oplus (a \ominus b)$
Now the formal definition

**Definition**

Definition: m-semirings A semiring attached with a monus operator ⊖ such that

1. \( a \oplus (b \ominus a) = b \oplus (a \ominus b) \)
2. \( (a \ominus b) \ominus c = a \ominus (b \oplus c) \)
m-semirings

Now the formal definition

**Definition**

Definition: m-semirings A semiring attached with a monus operator $\ominus$ such that

- $a \oplus (b \ominus a) = b \oplus (a \ominus b)$
- $(a \ominus b) \ominus c = a \ominus (b \oplus c)$
- $a \ominus a = 0 \ominus a = 0$
We can extend \((F, \lor, \land, \perp, \top)\) from positive boolean to all boolean functions.
We can extend \((F, \lor, \land, \perp, \top)\) from positive boolean to all boolean functions.

- we define \(\ominus = \land(\ , \neg)\)
We can extend \((F, \lor, \land, \perp, \top)\) from positive boolean to all boolean functions.

- we define \(\ominus = \land(\land, \neg)\)
- \(a \ominus b\) becomes \(a \land \neg b\).
Extending to all boolean functions

We can extend \((F, \lor, \land, \bot, \top)\) from positive boolean to all boolean functions.

- we define \(\ominus = \land(\ , \neg \ )\)
- \(a \ominus b\) becomes \(a \land \neg b\).
- This captures RA’s set differences for boolean functions.
We can extend \((F, \lor, \land, \bot, \top)\) from positive boolean to all boolean functions.

- we define \(\ominus = \land(\, , \neg\)\)
- \(a \ominus b\) becomes \(a \land \neg b\).
- This captures RA’s set differences for boolean functions.

If we want \(R1 \setminus R2\) to be generated by \(Q\) from \(D'\), we need the formulas of \(R1\) to hold and the formulas of \(R2\) not to hold.
extending why provenance

We can extend the why provenance \( (P(P(T)), \cup, \emptyset, \emptyset, \{\emptyset\}) \) using deep set difference.
We can extend the why provenance \((P(P(T)), \cup, \uplus, \emptyset, \{\emptyset\})\) using deep set difference

- Lets say we want to witness event \(\alpha\) but not \(\beta\)
extending why provenance

We can extend the why provenance \((P(P(T)), \cup, \uplus, \emptyset, \{\emptyset\})\) using deep set difference

- Lets say we want to witness event \(\alpha\) but not \(\beta\)
- We need to remove every witness combination of \(\beta\) from every witness combination of \(\alpha\)
Universality?

<table>
<thead>
<tr>
<th>What is data provenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semirings - a mathematical model for provenance</td>
</tr>
<tr>
<td>Implementing provenance in DBs</td>
</tr>
<tr>
<td>Probabilistic Evaluation</td>
</tr>
<tr>
<td>Relation Algebra</td>
</tr>
<tr>
<td>Overcoming positivity concerns</td>
</tr>
<tr>
<td>Limitations and extensions</td>
</tr>
<tr>
<td>Representations - Circuits</td>
</tr>
</tbody>
</table>

There is a universal m-semiring. However, it does not have an elegant representation. We will later see how it can still be compactly represented.
Universality?

- There is a universal m-semiring
Universality?

- There is a universal m-semiring
- However, it does not have an elegant representation.
Universality?

- There is a universal m-semiring
- However, it does not have an elegant representation.
- We will later see how it can still be compactly represented.
The Universal m-semiring

The universal m-semiring is the free m-semiring, this means it consists of all the words generated by the context free grammar

\[ S \rightarrow T \]

\[ T \rightarrow S \]

\[ x \]

\[ i \]

\[ j \]

\[ j \ominus \]

\[ T \rightarrow S \]

\[ x \]

\[ i \]

\[ j \]

\[ i \]

\[ j \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]

\[ \]
The Universal m-semiring

- The universal m-semiring is the free m-semiring
The Universal m-semiring

• The universal m-semiring is the **free** m-semiring
• this means it consists of all the words generated by the context free grammar

\[
\begin{align*}
S & \rightarrow T \mid TOPT \\
OP & \rightarrow \oplus \mid \otimes \mid \ominus \\
T & \rightarrow S \mid x_i
\end{align*}
\]

under equivalence relation defined by the axioms of semirings
The Universal m-semiring

- The universal m-semiring is the **free** m-semiring
- this means it consists of all the words generated by the context free grammar

\[
S \rightarrow T \mid TOPT \\
OP \rightarrow \oplus \mid \otimes \mid \ominus \\
T \rightarrow S \mid x_i
\]

under equivalence relation defined by the axioms of semirings

- computing this equivalence relation is not trivial
The Universal m-semiring

- The universal m-semiring is the **free** m-semiring
- this means it consists of all the words generated by the context free grammar

\[
\begin{align*}
S & \rightarrow T | TOPT \\
OP & \rightarrow \oplus | \otimes | \ominus \\
T & \rightarrow S | x_i
\end{align*}
\]

under equivalence relation defined by the axioms of semirings

- computing this equivalence relation is not trivial
- therefore it is not immediately possible to see equality between elements in the free-semirings
Table of Contents

1. What is data provenance
   - Motivation
   - Common types of provenance

2. Semirings - a mathematical model for provenance
   - Requirements
   - Mathematical formulation
   - Model to practice
   - Universality

3. Implementing provenance in DBs
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. Probabilistic Evaluation
   - Complexity concerns
   - Approaches to tackle complexity
<table>
<thead>
<tr>
<th>What is data provenance</th>
<th>Relation Algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semirings - a mathematical model for provenance</td>
<td>Overcoming positivity concerns</td>
</tr>
<tr>
<td>Implementing provenance in DBs</td>
<td>Limitations and extensions</td>
</tr>
<tr>
<td>Probabilistic Evaluation</td>
<td>Representations - Circuits</td>
</tr>
</tbody>
</table>

Where provenance - what can't we ask?

There are some provenance schemes that cannot be captured by semirings. A well-known example is

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
There are some provenance schemes that cannot be captured by semirings.
Where provenance - what can’t we ask?

There are some provenance schemes that cannot be captured by semirings.

A well known example is Where provenance
Where provenance

Given \( q_i \) we want a map between each value in the tuple and the value it originated from in the original DB. Especially useful for DBs with complex interdependencies like biological databases. Useful when the data stored is not completely processed or formalized. Where the context of the position of a value still retains some information that is not fully contained in the RDB scheme.
Where provenance

Given \( q_i \in Q(D) \) we want a map between each \textbf{value} in the tuple and the value it originated from in the original DB.
Where provenance

- Given $q_i \in Q(D)$ we want a map between each value in the tuple and the value it originated from in the original DB.
- Especially useful for DBs with complex interdependencies like biological databases.
Given $q_i \in Q(D)$ we want a map between each value in the tuple and the value it originated from in the original DB.

Especially useful for DBs with complex interdependencies like biological databases.

Useful when the data stored is not completely processed or formalized. Where the context of the position of a value still retains some information that is not fully contained in the RDB scheme.
Where provenance

- Given $q_i \in Q(D)$ we want a map between each **value** in the tuple and the value it originated from in the original DB.
- Especially useful for DBs with complex interdependencies like biological databases.
- Useful when the data stored is not completely processed or formalized. Where the context of the position of a value still retains some information that is not fully contained in the RDB scheme.
- Formulated by a bipartite graph with edges between values in $DB$ to values in $Q(DB)$ that were derived from them.
Why is where provenance not realizable by semirings?

Since we are talking about individual values, we are affected by projection and renaming. Joins work differently on where provenance logically. Values computed by join must point to an intersection of both values that were joined, rather than the addition of them. These differences stem from the fact that where provenance sees information in the DB beyond the values in the tuples. Despite this setback, support for where provenance is not complicated (though beyond the scope of this talk).
Why is where provenance not realizable by semirings?

- Since we are talking about individual values, we are affected by projection and renaming.
Why is where provenance not realizable by semirings?

- Since we are talking about individual values, we are affected by projection and renaming.
- Joins work differently on where provenance logically. Values computed by join must point to an intersection of both values that where joined, rather then the addition of them.
Why is where provenance not realizable by semirings?

- Since we are talking about individual values, we are affected by projection and renaming.
- Joins work differently on where provenance logically. Values computed by join must point to an intersection of both values that were joined, rather than the addition of them.
- These differences stem from the fact that where provenance sees information in the DB beyond the values in the tuples.
Why is where provenance not realizable by semirings?

- Since we are talking about individual values, we are affected by projection and renaming.
- Joins work differently on where provenance logically. Values computed by join must point to an intersection of both values that were joined, rather than the addition of them.
- These differences stem from the fact that where provenance sees information in the DB beyond the values in the tuples.
- Despite this setback, support for where provenance is not complicated (though beyond the scope of this talk)
Recursive queries
The provenance type might not be the only player at fault.
Recursive queries

- The provenance type might not be the only player at fault.
- It can also be the query type.
Recursive queries

The key idea behind this incompatibility is that recursive queries require formalisms which can express cycles. You cannot write something like $p(X) = (3 + 5(p(X)))$ and end up with a finite polynomial. There are extensions of semirings that support semirings.
Recursive queries

- Recursive queries provenance is not expressable by semirings
Recursive queries

- Recursive queries provenance is not expressable by semirings
- The key idea behind this incompatibility is that Recursive queries requires formalisms which can express cycles
Recursive queries

- Recursive queries provenance is not expressable by semirings
- The key idea behind this incompatibility is that Recursive queries requires formalisms which can express cycles
- you can’t write something like \( p(X) = (3 + 5(p(X))) \) and end up with a finite polynomial
Recursive queries

- Recursive queries provenance is not expressable by semirings
- The key idea behind this incompatibility is that Recursive queries requires formalisms which can express cycles
- you can’t write something like \( p(X) = (3 + 5(p(X))) \) and end up with a finite polynomial
- There are extensions of semirings that support semirings
Cyclical extensions of semirings

- Continuous semirings are mathematical creatures that support cycles.
- They have a universal element $N$, the integer power series on $X$.
- Just with more than 1 variable, they support cycles.

However, infinite power series are not computationally tractable.
Cyclical extensions of semirings

- \(\omega\)-continuous semirings are mathematical creatures that support cycles.
Cyclical extensions of semirings

- ω-continuous semirings are mathematical creatures that support cycles.
- They have a universal element \( \mathbb{N}[[X]] \) the integer power series on \( X \).
Cyclical extensions of semirings

- $\omega$-continuous semirings are mathematical creatures that support cycles.
- They have a universal element $\mathbb{N}[[X]]$ the integer power series on $X$
- basically elements of the form $\sum_{i=-\infty}^{\infty} a_i X^i$ just with more than 1 variable
Cyclical extensions of semirings

- \( \omega \)-continuous semirings are mathematical creatures that support cycles.
- They have a universal element \( \mathbb{N}[[X]] \) the integer power series on \( X \)
- basically elements of the form \( \sum_{i=-\infty}^{\infty} a_i x^i \) just with more than 1 variable
- They support cycles

\[
\left( \sum_{i=-\infty}^{\infty} 1x^i \right) \times \left( \sum_{i=-\infty}^{\infty} (-1)^i x^i \right)x^2 = \sum_{i=-\infty}^{\infty} 1x^i = \sum_{i=-\infty}^{\infty} (-1x)^i
\]
Cyclical extensions of semirings

- \( \omega \)-continuous semirings are mathematical creatures that support cycles.
- They have a universal element \( \mathbb{N}[[X]] \) the integer power series on \( X \)
- basically elements of the form \( \sum_{i=-\infty}^{\infty} a_i x^i \) just with more than 1 variable
- They support cycles

\[
\left( \sum_{i=-\infty}^{\infty} 1 x^i \right) \times \left( \sum_{i=-\infty}^{\infty} (-1)^i x^i \right) \times 2 = \sum_{i=-\infty}^{\infty} 1 x^i = \sum_{i=-\infty}^{\infty} (-1x)^i
\]

- however, infinite power series are not computationally tractable
Cyclical extensions of semirings

A set of cyclical extensions that are more feasible are called $k$-closed semirings. In essence, they are extensions of cyclical objects such as $\mathbb{Z}_p$ into ringlike cycles. This solution is workable and has been used in the past to compute recursive queries such as shortest graph path. Since $k$ depends on parameters of the query and the DB, this is still not a "perfect" extension. Some generalization ability is lost.
A set of cyclical extensions that are more feasible are called $k$-closed semirings
A set of cyclical extensions that are more feasible are called $k$-closed semirings.

In essence they are extensions of cyclical objects such as $\mathbb{Z}_p$ into ringlike cycles.
A set of cyclical extensions that are more feasible are called \( k \)-closed semirings.

In essence they are extensions of cyclical objects such as \( \mathbb{Z}_p \) into ringlike cycles.

This solution is workable and has been used in the past to compute recursive queries such as shortest graph path.
Cyclical extensions of semirings

- A set of cyclical extensions that are more feasible are called *k*-closed semirings.
- In essence they are extensions of cyclical objects such as $\mathbb{Z}_p$ into ringlike cycles.
- This solution is workable and has been used in the past to compute recursive queries such as shortest graph path.
- Since *k* depends on parameters of the query and DB this is still not a "perfect" extension.
Cyclical extensions of semirings

- A set of cyclical extensions that are more feasible are called $k$-closed semirings.
- In essence they are extensions of cyclical objects such as $\mathbb{Z}_p$ into ringlike cycles.
- This solution is workable and has been used in the past to compute recursive queries such as shortest graph path.
- Since $k$ depends on parameters of the query and DB this is still not a "perfect" extension.
- Some generalization ability is lost.
Table of Contents

1. What is data provenance
   - Motivation
   - Common types of provenance

2. Semirings - a mathematical model for provenance
   - Requirements
   - Mathematical formulation
   - Model to practice
   - Universality

3. Implementing provenance in DBs
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. Probabilistic Evaluation
   - Complexity concerns
   - Approaches to tackle complexity
What is data provenance
Semirings - a mathematical model for provenance
Implementing provenance in DBs
Probabilistic Evaluation
Relation Algebra
Overcoming positivity concerns
Limitations and extensions
Representations - Circuits

recap
leaving where provenance and recursive queries aside, we shall focus on how to represent provenances
leaving where provenance and recursive queries aside, we shall focus on how to represent provenances

In the current scope, provenances can be boolean function or semirings or $m$- semiring
Our requirements of our model for provenance were:
Our requirements of our model for provenance were:

- Generalizes most desired types of provenance
Reminder

Our requirements of our model for provenance were:

- Generalizes most desired types of provenance
- Allows compact representation
Our requirements of our model for provenance were:

- Generalizes most desired types of provenance
- Allows compact representation
- Computations track query evaluations closely
Our requirements of our model for provenance were:

- Generalizes most desired types of provenance
- Allows compact representation
- Computations track query evaluations closely
- Takes advantage of similarities between queries and between provenances
Our requirements of our model for provenance were:

- Generalizes most desired types of provenance
- Allows compact representation
- Computations track query evaluations closely
- Takes advantage of similarities between queries and between provenances
Reminder

Our requirements of our model for provenance were:

- Generalizes most desired types of provenance
- Allows compact representation
- Computations track query evaluations closely
- Takes advantage of similarities between queries and between provenances
Recap

The security semiring can be represented as elementary types such as ints. However, boolean functions or elements in $\mathbb{N}[X]$ are more complex. Naively, we can store them as formula strings. However, this is suboptimal since there are many common factors that will be reused between queries and between tuples of the same query. Circuits can remedy this.
The security semiring can be represented as elementary types such as ints.
Recap

- The security semiring can be represented as elementary types such as ints.
- But boolean functions or elements in $\mathbb{N}[X]$ are more complex.
The security semiring can be represented as elementary types such as ints.

But boolean functions or elements in $\mathbb{N}[X]$ are more complex.

Naively, we can store them as formula strings.
The security semiring can be represented as elementary types such as ints.

But boolean functions or elements in $\mathbb{N}[X]$ are more complex.

Naively, we can store them as formula strings.

However, this is suboptimal since there are many common factors that will be reused between queries and between tuples of the same query.
Recap

- The security semiring can be represented as elementary types such as ints.
- But boolean functions or elements in $\mathbb{N}[X]$ are more complex.
- Naively, we can store them as formula strings.
- However, this is suboptimal since there are many common factors that will be reused between queries and between tuples of the same query.
- Circuits can remedy this.
### Example

**Table: Personel**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>position</th>
<th>city</th>
<th>classification</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Director</td>
<td>New York</td>
<td>unclassified</td>
<td>t₁</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>Janitor</td>
<td>New York</td>
<td>restricted</td>
<td>t₂</td>
</tr>
<tr>
<td>3</td>
<td>Dave</td>
<td>Analyst</td>
<td>Paris</td>
<td>confidential</td>
<td>t₃</td>
</tr>
<tr>
<td>4</td>
<td>Ellen</td>
<td>Field agent</td>
<td>Berlin</td>
<td>secret</td>
<td>t₄</td>
</tr>
<tr>
<td>5</td>
<td>Magdalen</td>
<td>Double agent</td>
<td>Paris</td>
<td>top_secret</td>
<td>t₅</td>
</tr>
<tr>
<td>6</td>
<td>Nancy</td>
<td>HR</td>
<td>Paris</td>
<td>restricted</td>
<td>t₆</td>
</tr>
<tr>
<td>7</td>
<td>Susan</td>
<td>Analyst</td>
<td>Berlin</td>
<td>secret</td>
<td>t₇</td>
</tr>
</tbody>
</table>
Example

**Table:** $Q_1$ with annotations

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ney York</td>
<td>$g_1$</td>
</tr>
<tr>
<td>Paris</td>
<td>$g_2$</td>
</tr>
<tr>
<td>Berlin</td>
<td>$g_3$</td>
</tr>
</tbody>
</table>

**Table:** $Q_2$ with annotations

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ney York</td>
<td>$g_4$</td>
</tr>
<tr>
<td>Paris</td>
<td>$g_5$</td>
</tr>
<tr>
<td>Berlin</td>
<td>$g_6$</td>
</tr>
</tbody>
</table>
Example

We can build a free $m$-semiring form computation graph that shares nodes between query items
Example

When trying to compute a specific provenance on a specific tuple $g$, we used the relevant homomorphism $h$ on $g$’s subtree.
Example

When trying to compute a specific provenance on a specific tuple $g$, we used the relevant homomorphism $h$ on $g$’s subtree.
When trying to compute a specific provenance on a specific tuple $g$, we used the relevant homomorphism $h$ on $g$’s subtree.

Note that we just ignored the negation operations. Since they have no meaning in this provenance.
Circuits

Circuits are basically computation graphs. Combining them with what we have learned so far, we get the following approach:

- Construct inductively a provenance circuit over input tuples for every RA operation performed on a query.
- Reuse nodes that have already been computed.
- Construct the circuit in the free $m$-semiring so that all provenances we want are computed in 1 circuit. Each additional provenance needs to only save the implementation of the homomorphism from the free-$m$-semiring.

This methodology is utilized in the ProvSQL engine.
Circuits

- Circuits are basically computation graphs.
Circuits

- Circuits are basically computation graphs.
- Combining them with what we have learned so far, we get the following approach:
Circuits

- Circuits are basically computation graphs.
- Combining them with what we have learned so far, we get the following approach:
  - Construct inductively a provenance circuit over input tuples for every RA operation performed on a query.
Circuits

- Circuits are basically computation graphs.
- Combining them with what we have learned so far, we get the following approach:
  - Construct inductively a provenance circuit over input tuples for every RA operation performed on a query
  - Reuse nodes that have already been computed
Circuits

- Circuits are basically computation graphs.
- Combining them with what we have learned so far, we get the following approach:
  - Construct inductively a provenance circuit over input tuples for every RA operation performed on a query
  - reuse nodes that have already been computed
  - construct the circuit in the free $m$-semiring so that all provenances we want are computed in 1 circuit.
Circuits

- Circuits are basically computation graphs.
- Combining them with what we have learned so far, we get the following approach:
  - Construct inductively a provenance circuit over input tuples for every RA operation performed on a query
  - reuse nodes that have already been computed
  - construct the circuit in the free $m$-semiring so that all provenances we want are computed in 1 circuit.
  - each additional provenance needs to only save the implementation of the homomorphism from the free-$m$ semiring
Circuits

- Circuits are basically computation graphs.
- Combining them with what we have learned so far, we get the following approach:
  - Construct inductively a provenance circuit over input tuples for every RA operation performed on a query
  - Reuse nodes that have already been computed
  - Construct the circuit in the free $m$-semiring so that all provenances we want are computed in 1 circuit.
  - Each additional provenance needs to only save the implementation of the homomorphism from the free-$m$ semiring
  - This methodology is utilized in the ProvSQL engine.
Probabilistic DBs

Probabilistic evaluation on DBs is the problem of querying a DB in which not all information is certain. Therefore, the query result cannot always return certain answers. Instead, we can return results and their probability of existing.
Probabilistic evaluation on DBs is the problem of querying a DB in which not all information is certain.
Probabilistic DBs

- Probabilistic evaluation on DBs is the problem of querying a DB in which not all information is certain.
- therefore, the query result cannot always return certain answers.
Probabilistic evaluation on DBs is the problem of querying a DB in which not all information is certain. Therefore, the query result cannot always return certain answers. Instead, we can return results and their probability of existing.
When addressing the problem of evaluating probabilities, we will use the model of tuple-independent databases.
tuple-independant model

When addressing the problem of evaluating probabilities, we will use the model of tuple-independant databases.

- each tuple \( i \) is assigned an independent probability of existing \( t_i \)
When addressing the problem of evaluating probabilities, we will use the model of tuple-independent databases.

- Each tuple $i$ is assigned an independent probability of existing $t_i$.
- Note that this information is expressible by boolean provenance.
Example - boolean circuit
Table of Contents

1. What is data provenance
   - Motivation
   - Common types of provenance

2. Semirings - a mathematical model for provenance
   - Requirements
   - Mathematical formulation
   - Model to practice
   - Universality

3. Implementing provenance in DBs
   - Relation Algebra
   - Overcoming positivity concerns
   - Limitations and extensions
   - Representations - Circuits

4. Probabilistic Evaluation
   - Complexity concerns
   - Approaches to tackle complexity

Presented by Shaked Or - Knowledge seminar SPR18 (236804)
What is data provenance

Semirings - a mathematical model for provenance

Implementing provenance in DBs

Probabilistic Evaluation

Complexity concerns

Approaches to tackle complexity

probability is hard

Presented by Shaked Or - Knowledge seminar SPR18 (236804)

Probabilistic Evaluation in DBs

Provenance and Probabilities in Relational Databases: From Theory to Practice
probability is hard

- probabilistic evaluation on relational DB is \( \#P \)-hard.
probabilistic evaluation on relational DB is \#P-hard.

provenance will not reduce the hardness of this problem.
probability is hard

• probabilistic evaluation on relational DB is $\#P$-hard.
• provenance will not reduce the hardness of this problem.
• However, using boolean provenance, we can separate the task of producing the boolean functions from the task of evaluating probabilities on them.
probability is hard

- Probabilistic evaluation on relational DB is $\#P$-hard.
- Provenance will not reduce the hardness of this problem.
- However, using boolean provenance, we can separate the task of producing the boolean functions from the task of evaluating probabilities on them.
  - the former is in $P$-time
probability is hard

- probabilistic evaluation on relational DB is \#P-hard.
- provenance will not reduce the hardness of this problem.
- However, using boolean provenance, we can separate the task of producing the boolean functions from the task of evaluating probabilities on them
  - the former is in \( P \)-time
  - the latter is \#P-hard and is the real cause of the hardness of prob DB evaluation
So how can we use the tools seen so far to perform probabilistic evaluation?
| 1 | What is data provenance  |
|   | Motivation               |
|   | Common types of provenance |

| 2 | Semirings - a mathematical model for provenance  |
|   | Requirements               |
|   | Mathematical formulation   |
|   | Model to practice          |
|   | Universality               |

| 3 | Implementing provenance in DBs  |
|   | Relation Algebra            |
|   | Overcoming positivity concerns |
|   | Limitations and extensions   |
|   | Representations - Circuits  |

| 4 | Probabilistic Evaluation  |
|   | Complexity concerns       |
|   | Approaches to tackle complexity |
Naive approach - Brute force

We take \( p_q \), and run through all possible valuations of its variables.
Naive approach - Brute force

We take $p_{qi}$ and run through all possible valuations of its variables.

- Pretty bad, rarely feasible.
Naive approach - Brute force

We take $p_{qi}$ and run through all possible valuations of its variables.

- Pretty bad, rarely feasible.
- but much better than the naive approach without provenance
Naive approach - Brute force

We take $p_{q_i}$ and run through all possible valuations of its variables.

- Pretty bad, rarely feasible.
- but much better than the naive approach without provenance
  - Enumerating all possible DBs and running $Q$ on all of them.
We take $p_{q_i}$ and run through all possible valuations of its variables.

- Pretty bad, rarely feasible.
- but much better than the naive approach without provenance
  - Enumerating all possible DBs and running $Q$ on all of them.
- we still take advantage of the sharing across the circuit
Naive approach - Brute force

We take $p_{q_i}$ and run through all possible valuations of its variables.

- Pretty bad, rarely feasible.
- but much better than the naive approach without provenance
  - Enumerating all possible DBs and running $Q$ on all of them.
- we still take advantage of the sharing across the circuit
- easily enables trivial pruning of computation trees
We take $p_{qi}$ and run through all possible valuations of its variables.

- Pretty bad, rarely feasible.
- but much better than the naive approach without provenance
  - Enumerating all possible DBs and running $Q$ on all of them.
- we still take advantage of the sharing across the circuit
- easily enables trivial pruning of computation trees
  - due to formula structure
Naive approach - Brute force

We take $p_{qi}$ and run through all possible valuations of its variables.

- Pretty bad, rarely feasible.
- but much better than the naive approach without provenance
  - Enumerating all possible DBs and running $Q$ on all of them.
- we still take advantage of the sharing across the circuit
- easily enables trivial pruning of computation trees
  - due to formula structure
  - due to approximation thresholds
Approximation techniques

We can use approximation techniques
Approximation techniques

We can use approximation techniques

- The most simple is Monte-Carlo sampling, feasible but slow
Approximation techniques

We can use approximation techniques

- The most simple is Monte-Carlo sampling, feasible but slow
- There are increasingly refined approximation techniques
Monte Carlo

In a nut shell. Uniform sampling converges to measure. Here we calculate $\pi$ using monte carlo.
Exploiting structure
Exploiting structure

- General techniques are very important.
Exploiting structure

- General techniques are very important.
- But sometimes, real world needs do not always map to the more general case
**Exploiting structure**

- General techniques are very important.
- But sometimes, real world needs do not always map to the more general case
- We can dynamically exploit the structure of our specific problem to allow the use of more specific tools
Exploiting structure

- General techniques are very important.
- But sometimes, real world needs do not always map to the more general case
- We can dynamically exploit the structure of our specific problem to allow the use of more specific tools
  - Exploiting the query structure
Exploiting structure

- General techniques are very important.
- But sometimes, real world needs do not always map to the more general case.
- We can dynamically exploit the structure of our specific problem to allow the use of more specific tools:
  - Exploiting the query structure
  - Exploiting the DB structure
Exploiting query structure
Exploiting query structure

- Some types of boolean functions admit to efficient probabilistic evaluations.
Exploiting query structure

- Some types of boolean functions admit to efficient probabilistic evaluations.
- A well known type is *inversion-free* boolean functions. for which tractable ordered binary diagrams can be obtained.
Exploiting query structure

- Some types of boolean functions admit to efficient probabilistic evaluations.
- A well known type is *inversion-free* boolean functions. for which tractable ordered binary diagrams can be obtained.
- A more general type is d-DNNF.
Exploiting query structure

- Some types of boolean functions admit to efficient probabilistic evaluations.
- A well known type is *inversion-free* boolean functions, for which tractable ordered binary diagrams can be obtained.
- A more general type is d-DNNF.
- Both admit to linear-time probabilistic evaluation.
Binary decision trees are trees whose leaves represent boolean variable selection.

Naive BDDs are exponential in the number of variables.
inversion free UCQs

Some boolean functions have variable orderings that result in linear size BDDs
inversion free UCQs

Some boolean functions have variable orderings that result in linear size BDDs
inversion free UCQs

Some boolean functions have variable orderings that result in linear size BDDs

Bad OBDD of $f = x_1x_2 + x_3x_4 .... x_{2n-1}x_{2n}$
inversion free UCQs
inversion free UCQs

Good OBDD of \( f = x_1x_2 + x_3x_4 \ldots x_{2n-1}x_{2n} \)
A boolean function $f$ is in deterministic decomposable negation normal form (d-DDNF) if
A boolean function $f$ is in deterministic decomposable negation normal form (d-DDNF) if

$$V_{ars}(f_1) \cap V_{ars}(f_2) = \emptyset \text{ for every } f_1 \land f_2 \text{ in } f$$
A boolean function \( f \) is in deterministic decomposable negation normal form (d-DDNF) if

- \( \text{Vars}(f_1) \cap \text{Vars}(f_2) = \emptyset \) for every \( f_1 \land f_2 \) in \( f \)
- \( f_1 \) and \( f_2 \) are incompatible for every \( f_1 \lor f_2 \) in \( f \)
Example of d-DDNF

For example, $Q_2$ in the original example admits a d-DDNF form
Exploiting DB structure

Alternatively, we can exploit the structure of the DB.
Exploiting DB structure

Alternatively, we can exploit the structure of the DB.

- if the treewidth of the DB is bounded, we can create bounded treewidth provenance circuits
Exploiting DB structure

Alternatively, we can exploit the structure of the DB.

- If the treewidth of the DB is bounded, we can create bounded treewidth provenance circuits.
- Boolean evaluation on bounded treewidth provenance if tractable (for constant tree-width).
Knowledge compilation

If non of these approaches are feasible, we can always result to general *Knowledge compilation*
Knowledge compilation

If none of these approaches are feasible, we can always resort to general *knowledge compilation*

- Knowledge compilation are techniques that transform boolean functions of one form into a more tractable form.
If non of these approaches are feasible, we can always result to general *Knowledge compilation*

- knowledge compilation are techniques that transform boolean functions of one form into a more tractable form.
- A usefull technique it to take some $f$
Knowledge compilation

If none of these approaches are feasible, we can always resort to general *knowledge compilation*

- knowledge compilation are techniques that transform boolean functions of one form into a more tractable form.
- A useful technique is to take some \( f \)
  - convert to \( cnf \) (linear time)
If none of these approaches are feasible, we can always result to general *Knowledge compilation*.

- Knowledge compilation are techniques that transform boolean functions of one form into a more tractable form.
- A useful technique is to take some $f$:
  - Convert to $cnf$ (linear time)
  - Try to convert to d-DNNF using knowledge compilation
Combined approaches

One can always combine these techniques to try and handle hard queries
Combined approaches

One can always combine these techniques to try and handle hard queries

- exploit both data structure and query structure
Combined approaches

One can always combine these techniques to try and handle hard queries:

- exploit both data structure and query structure
- decompose the $f$ into combinations of functions, some of which are tractable
Provenance is very useful in real-world applications on relational DBs. Most provenance can be generalized through semirings and m-semirings. m-Semiring actions correspond to RA, making provenance easy to implement and integrate with any RDB engine. Universal m-semirings can remove the need for semiring per provenance storage. Circuits can compactly represent multiple query provenances. Provenance circuits reduce the difficulty of tackling probabilistic evaluations.
Provenance is very useful in real world applications on relational DBs.
Summary

- Provenance is very useful in real world applications on relational DBs
- Most provenance can be generalised through semirings and m-semirings
Provenance is very useful in real world applications on relational DBs

Most provenance can be generalised through semirings and m-semirings

m-Semiring actions correspond to RA, making provenance easy to implement and integrate with any RDB engine
Summary

- Provenance is very useful in real world applications on relational DBs.
- Most provenance can be generalised through semirings and m-semirings.
- m-Semiring actions correspond to RA, making provenance easy to implement and integrate with any RDB engine.
- Universal m-semirings can remove the need for semiring per provenance storage.
Summary

- Provenance is very useful in real-world applications on relational DBs.
- Most provenance can be generalized through semirings and m-semirings.
- m-Semiring actions correspond to RA, making provenance easy to implement and integrate with any RDB engine.
- Universal m-semirings can remove the need for semiring per provenance storage.
- Circuits can compactly represent multiple query provenances.
Summary

- Provenance is very useful in real-world applications on relational DBs.
- Most provenance can be generalized through semirings and m-semirings.
- $m$-Semiring actions correspond to RA, making provenance easy to implement and integrate with any RDB engine.
- Universal m-semirings can remove the need for semiring per provenance storage.
- Circuits can compactly represent multiple query provenances.
- Provenance circuits reduce the difficulty of tackling probabilistic evaluations.
Select Question From Audience?