Factorized Databases

fdbresearch.github.io

Ahmet Kara (University of Zurich)

Probabilistic Circuits and Logic Workshop
Simons Institute, October 18, 2023
Factorized Representations of Query Results: Size Bounds and Readability.
Dan Olteanu and Jakub Závodný.

Size Bounds for Factorized Representations of Query Results.
Dan Olteanu, and Jakub Závodný.
What are Factorized Representations About?

Two fundamental observations:

- The listing representation of query answers entails redundancy
- This can be avoided by a succinct and lossless factorized representation
What are Factorized Representations About?

Two fundamental observations:

- The **listing representation** of query answers entails redundancy
- This can be avoided by a succinct and lossless **factorized representation**

Effective tools for managing factorized representations:

- **Representation systems** for factorized query answers and provenance
- Computation of factorized query answers in **worst-case optimal time**
- **Constant-delay enumeration** of the tuples represented by factorization
## Ordering Pizzas

<table>
<thead>
<tr>
<th>Orders</th>
<th>Pizza</th>
<th>Ingredients</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
<td>day</td>
<td>pizza</td>
</tr>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
<td>Basilea</td>
</tr>
<tr>
<td>Haozhe</td>
<td>Friday</td>
<td>Hawaii</td>
</tr>
<tr>
<td>Johannes</td>
<td>Friday</td>
<td>Hawaii</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Ordering Pizzas

<table>
<thead>
<tr>
<th>Orders</th>
<th>Pizza</th>
<th>Ingredients</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
<td>day</td>
<td>pizza</td>
</tr>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
</tr>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
</tr>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
</tr>
<tr>
<td>Haozhe</td>
<td>Friday</td>
<td>Hawaii</td>
</tr>
<tr>
<td>Johannes</td>
<td>Friday</td>
<td>Hawaii</td>
</tr>
</tbody>
</table>

Natural join of the above relations:

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>pizza</th>
<th>ingredient</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
<td>garlic</td>
<td>6</td>
</tr>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
<td>mozza</td>
<td>8</td>
</tr>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
<td>tomato</td>
<td>4</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
<td>Basilea</td>
<td>garlic</td>
<td>6</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
<td>Basilea</td>
<td>mozza</td>
<td>8</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
<td>Basilea</td>
<td>tomato</td>
<td>4</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Basileas & Hawaiis in Relational Algebra

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>pizza</th>
<th>ingredient</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
<td>garlic</td>
<td>6</td>
</tr>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
<td>mozza</td>
<td>8</td>
</tr>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
<td>tomato</td>
<td>4</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
<td>Basilea</td>
<td>garlic</td>
<td>6</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
<td>Basilea</td>
<td>mozza</td>
<td>8</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
<td>Basilea</td>
<td>tomato</td>
<td>4</td>
</tr>
</tbody>
</table>

... ... ... ... ...

An algebraic encoding uses product (\(\times\)), union (\(\cup\)), and values:

\[
Dan \times Thursday \times Basilea \times garlic \times 6 \cup \\
Dan \times Thursday \times Basilea \times mozza \times 8 \cup \\
Dan \times Thursday \times Basilea \times tomato \times 4 \cup \\
Dan \times Friday \times Basilea \times garlic \times 6 \cup \\
Dan \times Friday \times Basilea \times mozza \times 8 \cup \\
Dan \times Friday \times Basilea \times tomato \times 4 \cup ...
\]
Factorized Join

Variable order
There are several algebraically equivalent factorized joins defined by distributivity of product over union and their commutativity.

Variable order

Instantiation of the variable order over the input database
There are several algebraically equivalent factorized joins defined by distributivity of product over union and their commutativity.
Observation:

- **price only depends** on ingredient and not on pizza

- .. so the same price for an ingredient **regardless** of the pizza.
... Now with Further Compression

Observation:

- price only \textit{depends} on ingredient and not on pizza
- \ldots so the same price for an ingredient \textit{regardless} of the pizza.

Idea: \textbf{Cache} price for a specific ingredient and avoid repetition!
... Now with Further Compression

Observation:

- price only depends on ingredient and not on pizza
- .. so the same price for an ingredient regardless of the pizza.

Idea: Cache price for a specific ingredient and avoid repetition!
Factorized Representations from a Knowledge Compilation Perspective
Factorized representations are

- deterministic
- decomposable
- smooth
- multi-valued
- ordered
From a Knowledge Compilation Perspective

Factorized representations are

- deterministic
  
  *all child trees of a union node are distinct*

- decomposable

- smooth

- multi-valued

- ordered
From a Knowledge Compilation Perspective

Factorized representations are

- deterministic
  
  all child trees of a union node are distinct

- decomposable

  all child trees of a product node are over disjoint variable sets

- smooth

- multi-valued

- ordered
From a Knowledge Compilation Perspective

Factorized representations are

- **deterministic**
  
  *all child trees of a union node are distinct*

- **decomposable**
  
  *all child trees of a product node are over disjoint variable sets*

- **smooth**
  
  *all child trees of a union node are over the same variable set*

- **multi-valued**

- **ordered**
Factorized representations are

- deterministic
  
  \textit{all child trees of a union node are distinct}

- decomposable
  
  \textit{all child trees of a product node are over disjoint variable sets}

- smooth
  
  \textit{all child trees of a union node are over the same variable set}

- multi-valued
  
  \textit{variables may have non-binary domains}

- ordered
From a Knowledge Compilation Perspective

Factorized representations are

- deterministic
  
  *all child trees of a union node are distinct*

- decomposable
  
  *all child trees of a product node are over disjoint variable sets*

- smooth
  
  *all child trees of a union node are over the same variable set*

- multi-valued
  
  *variables may have non-binary domains*

- ordered
  
  *all child trees of a union node are over the same variable order*
Operations on Factorized Representations
Operations on factorized representations in the compressed domain

- **join**
- **selection**
- **projection**
- **constant-delay enumeration**
- **aggregates** (count, sum-product, group-by)
- **updates**
Operations on factorized representations in the compressed domain

- **join**
  
  size of the output depends on the structure of the result (more on this later)

- **selection**

- **projection**

- **constant-delay enumeration**

- **aggregates** (count, sum-product, group-by)

- **updates**
Operations on factorized representations in the compressed domain

- **join**
  size of the output depends on the structure of the result (more on this later)

- **selection**
  linear time if selection variables on top of all other variables

- **projection**

- **constant-delay enumeration**

- **aggregates** (count, sum-product, group-by)

- **updates**
Operations on factorized representations in the compressed domain

- **join**
  
  size of the output depends on the structure of the result (more on this later)

- **selection**
  
  linear time if selection variables on top of all other variables

- **projection**
  
  linear time if projection variables on top of all other variables

- **constant-delay enumeration**

- **aggregates** (count, sum-product, group-by)

- **updates**
Operations on factorized representations in the compressed domain

- **join**
  
  size of the output depends on the structure of the result (more on this later)

- **selection**
  
  linear time if selection variables on top of all other variables

- **projection**
  
  linear time if projection variables on top of all other variables

- **constant-delay enumeration**
  
  also *order-by* if enumeration order is compatible with variable order

- **aggregates** (count, sum-product, group-by)

- **updates**
Operations on factorized representations in the compressed domain

- **join**
  size of the output depends on the structure of the result (more on this later)

- **selection**
  linear time if selection variables on top of all other variables

- **projection**
  linear time if projection variables on top of all other variables

- **constant-delay enumeration**
  also order-by if enumeration order is compatible with variable order

- **aggregates** (count, sum-product, group-by)
  group-by: linear time if group-by variables on top of all other variables

- **updates**
Operations on factorized representations in the compressed domain

- **join**
  
  size of the output depends on the structure of the result (more on this later)

- **selection**
  
  linear time if selection variables on top of all other variables

- **projection**
  
  linear time if projection variables on top of all other variables

- **constant-delay enumeration**
  
  also order-by if enumeration order is compatible with variable order

- **aggregates** (count, sum-product, group-by)
  
  group-by: linear time if group-by variables on top of all other variables

- **updates**
  
  update time depends on the *dynamic width* of the query
Compression Gains Brought by Factorization
Factorization versus Gzip for our Join Query

- **Tabular**: Lists one tuple per row in CSV text format
- **Gzip** (compression level 6): Outputs binary format
- **Factorization**: In text format (each digit takes one character)

![Graph showing compression ratio vs. database scale for different formats](image)
Factorization versus Gzip for our Join Query

- **Tabular**: Lists one tuple per row in CSV text format
- **Gzip** (compression level 6): Outputs binary format
- **Factorization**: In text format (each digit takes one character)

Take-away messages:

- Gzip does not identify distant repetitions
- Factorizations can be arbitrarily more succinct than gzipped relations
- Gzipping factorizations improves the compression by 3x
Real-world dataset used for commercial analytics in the retail domain

- Inventory (84M tuples), Census (1K), Location (1K),
  Sales (1.5M), Clearance (368K), Promotions (183K)

- All joins are key – foreign key

Compression factors by factorizing the natural joins of these relations:

- **26.61x** for the natural join of Inventory, Census, Location

- **159.59x** for the natural join of Inventory, Sales, Clearance, Promotions
Size Bounds for Factorized Representations

[Olteanu and Závodný, 2011-2015]

Given any conjunctive query \( Q \) and database \( D \), the result \( Q(D) \) has a \textbf{factorized} representation \textbf{with caching} of size \( \mathcal{O}(|D|^{s^\uparrow(Q)}) \)
Given any conjunctive query $Q$ and database $D$, the result $Q(D)$ has a factorized representation with caching of size $O(|D|^{s^\uparrow(Q)})$.

- For full conjunctive queries, this bound is asymptotically tight:
  - There exist arbitrarily large databases $D$ such that all factorized representations following variable orders have size $\Omega(|D|^{s^\uparrow(Q)})$.
Size Bounds for Factorized Representations

[Olteanu and Závodný, 2011-2015]

Given any conjunctive query $Q$ and database $D$, the result $Q(D)$ has a factorized representation with caching of size $O(|D|^{s^\uparrow(Q)})$

- For full conjunctive queries, this bound is asymptotically tight:
  - There exist arbitrarily large databases $D$ such that all factorized representations following variable orders have size $\Omega(|D|^{s^\uparrow(Q)})$

- The listing representation can have size $\Omega(|D|^{\rho^*(Q)})$, where the gap between $s^\uparrow(Q)$ and $\rho^*(Q)$ can be up to $|Q| - 1$
Size Bounds for Factorized Representations

[Olteanu and Závodný, 2011-2015]

Given any conjunctive query $Q$ and database $D$, the result $Q(D)$ has a factorized representation with caching of size $O(|D|^{s^\uparrow(Q)})$.

- For full conjunctive queries, this bound is asymptotically tight:
  - There exist arbitrarily large databases $D$ such that all factorized representations following variable orders have size $\Omega(|D|^{s^\uparrow(Q)})$.

- The listing representation can have size $\Omega(|D|^{\rho^*(Q)})$, where the gap between $s^\uparrow(Q)$ and $\rho^*(Q)$ can be up to $|Q| - 1$.

- For full conjunctive queries, factorized representations can be computed worst-case optimally (up to a log $|D|$ factor).
Factorization Width $s^\uparrow$

For any conjunctive query $Q$:

$$s^\uparrow(Q) = \min_{\text{variable orders } \omega \text{ for } Q} s^\uparrow(\omega)$$
For any conjunctive query $Q$:

$$s^\uparrow(Q) = \min_{\text{variable orders } \omega \text{ for } Q} s^\uparrow(\omega)$$

- For any hypertree decomposition $\mathcal{T}$, let $\text{fhtw}(\mathcal{T})$ be the fractional hypertree width of $\mathcal{T}$
Factorization Width $s^\uparrow$

For any conjunctive query $Q$:

$$s^\uparrow(Q) = \min_{\text{variable orders } \omega \text{ for } Q} s^\uparrow(\omega)$$

- For any hypertree decomposition $\mathcal{T}$, let $\text{fhtw}(\mathcal{T})$ be the fractional hypertree width of $\mathcal{T}$

Variable order $\omega$ for $Q$

\[ s^\uparrow(\omega) \]

translates to

Free-connex hypertree decomposition $\mathcal{T}$ for $Q$

\[ \text{fhtw}(\mathcal{T}) \]
Factorization Width $s^{\uparrow}$

For any conjunctive query $Q$:

$$s^{\uparrow}(Q) = \min_{\text{variable orders } \omega \text{ for } Q} s^{\uparrow}(\omega)$$

- For any hypertree decomposition $\mathcal{T}$, let $\text{fhtw}(\mathcal{T})$ be the fractional hypertree width of $\mathcal{T}$

variable order $\omega$
for $Q$

$s^{\uparrow}(\omega)$

translates to

free-connex hypertree decomposition $\mathcal{T}$ for $Q$

$fhtw(\mathcal{T})$

free-connex hypertree decomposition $\mathcal{T}$ for $Q$

$fhtw(\mathcal{T})$

translates to

variable order $\omega$
for $Q$

$s^{\uparrow}(\omega)$

$\geq$
For any conjunctive query $Q$:

$$s^\uparrow(Q) = \min_{\text{variable orders } \omega \text{ for } Q} s^\uparrow(\omega)$$

- For any hypertree decomposition $\mathcal{T}$, let $fhtw(\mathcal{T})$ be the fractional hypertree width of $\mathcal{T}$

$$s^\uparrow(\omega) \text{ translates to } \begin{cases} \text{variable order } \omega & \text{for } Q \\ s^\uparrow(\omega) \end{cases} \quad \text{translates to} \quad \begin{cases} \text{free-connex hypertree decomposition } \mathcal{T} & \text{for } Q \\ fhtw(\mathcal{T}) \end{cases}$$

$$fhtw(\mathcal{T}) \text{ translates to } \begin{cases} \text{free-connex hypertree decomposition } \mathcal{T} & \text{for } Q \\ fhtw(\mathcal{T}) \end{cases} \quad \geq \quad \begin{cases} \text{variable order } \omega & \text{for } Q \\ s^\uparrow(\omega) \end{cases}$$

$$\implies s^\uparrow(Q) = fhtw(Q), \text{ where } fhtw(Q) \text{ is the generalization of the fractional hypertree width from Boolean to conjunctive queries}$$
Where are Factorized Databases Used?
Where are Factorized Databases Used?

Research and development in **database systems and database theory**

- Graph data representation and processing
- Static and dynamic query evaluation
- Query provenance management
- Factorized aggregates
- Factorized machine learning
Use Case:
Probabilistic Databases
### Probabilistic Databases

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>pizza</th>
<th>o.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
<td>$o_1$</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
<td>Basilea</td>
<td>$o_2$</td>
</tr>
<tr>
<td>Haozhe</td>
<td>Friday</td>
<td>Hawaii</td>
<td>$o_3$</td>
</tr>
<tr>
<td>Johannes</td>
<td>Friday</td>
<td>Hawaii</td>
<td>$o_4$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pizza</th>
<th>ingredient</th>
<th>p.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basilea</td>
<td>garlic</td>
<td>$p_1$</td>
</tr>
<tr>
<td>Basilea</td>
<td>tomato</td>
<td>$p_2$</td>
</tr>
<tr>
<td>Basilea</td>
<td>mozza</td>
<td>$p_3$</td>
</tr>
<tr>
<td>Hawaii</td>
<td>tomato</td>
<td>$p_4$</td>
</tr>
<tr>
<td>Hawaii</td>
<td>mozza</td>
<td>$p_5$</td>
</tr>
<tr>
<td>Hawaii</td>
<td>pineapple</td>
<td>$p_6$</td>
</tr>
</tbody>
</table>

- Each tuple is associated with a Boolean random variable
- The random variables are independent
Querying Probabilistic Databases

Orders

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>pizza</th>
<th>o.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
<td>o₁</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
<td>Basilea</td>
<td>o₂</td>
</tr>
<tr>
<td>Haozhe</td>
<td>Friday</td>
<td>Hawaii</td>
<td>o₃</td>
</tr>
<tr>
<td>Johannes</td>
<td>Friday</td>
<td>Hawaii</td>
<td>o₄</td>
</tr>
</tbody>
</table>

Pizza

<table>
<thead>
<tr>
<th>pizza</th>
<th>ingredient</th>
<th>p.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basilea</td>
<td>garlic</td>
<td>p₁</td>
</tr>
<tr>
<td>Basilea</td>
<td>tomato</td>
<td>p₂</td>
</tr>
<tr>
<td>Basilea</td>
<td>mozza</td>
<td>p₃</td>
</tr>
<tr>
<td>Hawaii</td>
<td>tomato</td>
<td>p₄</td>
</tr>
<tr>
<td>Hawaii</td>
<td>mozza</td>
<td>p₅</td>
</tr>
<tr>
<td>Hawaii</td>
<td>pineapple</td>
<td>p₆</td>
</tr>
</tbody>
</table>

Query: “Is the natural join of Orders and Pizza non-empty?”

\[ Q = \bigvee_{c,d,p,i} \text{Orders}(c,d,p) \land \text{Pizza}(p,i) \]
Querying Probabilistic Databases

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>pizza</th>
<th>o.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
<td>o₁</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
<td>Basilea</td>
<td>o₂</td>
</tr>
<tr>
<td>Haozhe</td>
<td>Friday</td>
<td>Hawaii</td>
<td>o₃</td>
</tr>
<tr>
<td>Johannes</td>
<td>Friday</td>
<td>Hawaii</td>
<td>o₄</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pizza</th>
<th>ingredient</th>
<th>p.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basilea</td>
<td>garlic</td>
<td>p₁</td>
</tr>
<tr>
<td>Basilea</td>
<td>tomato</td>
<td>p₂</td>
</tr>
<tr>
<td>Basilea</td>
<td>mozza</td>
<td>p₃</td>
</tr>
<tr>
<td>Hawaii</td>
<td>tomato</td>
<td>p₄</td>
</tr>
<tr>
<td>Hawaii</td>
<td>mozza</td>
<td>p₅</td>
</tr>
<tr>
<td>Hawaii</td>
<td>pineapple</td>
<td>p₆</td>
</tr>
</tbody>
</table>

Query: “Is the natural join of Orders and Pizza non-empty?”

\[
Q = \bigvee_{c,d,p,i} \text{Orders}(c, d, p) \land \text{Pizza}(p, i)
\]

The query now returns the empty tuple mapped to a probability
## Querying Probabilistic Databases

**Orders**

<table>
<thead>
<tr>
<th>customer</th>
<th>day</th>
<th>pizza</th>
<th>o.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>Thursday</td>
<td>Basilea</td>
<td>o₁</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
<td>Basilea</td>
<td>o₂</td>
</tr>
<tr>
<td>Haozhe</td>
<td>Friday</td>
<td>Hawaii</td>
<td>o₃</td>
</tr>
<tr>
<td>Johannes</td>
<td>Friday</td>
<td>Hawaii</td>
<td>o₄</td>
</tr>
</tbody>
</table>

**Pizza**

<table>
<thead>
<tr>
<th>pizza</th>
<th>ingredient</th>
<th>p.v</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basilea</td>
<td>garlic</td>
<td>p₁</td>
</tr>
<tr>
<td>Basilea</td>
<td>tomato</td>
<td>p₂</td>
</tr>
<tr>
<td>Basilea</td>
<td>mozza</td>
<td>p₃</td>
</tr>
<tr>
<td>Hawaii</td>
<td>tomato</td>
<td>p₄</td>
</tr>
<tr>
<td>Hawaii</td>
<td>mozza</td>
<td>p₅</td>
</tr>
<tr>
<td>Hawaii</td>
<td>pineapple</td>
<td>p₆</td>
</tr>
</tbody>
</table>

**Query:** “Is the natural join of Orders and Pizza non-empty?”

\[
Q = \bigvee_{c,d,p,i} \text{Orders}(c, d, p) \land \text{Pizza}(p, i)
\]

Query \( Q \) is **hierarchical**

- For any two variables, either their atom sets are disjoint or one is contained in the other.
### Querying Probabilistic Databases

<table>
<thead>
<tr>
<th>Orders</th>
<th>Pizza</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
<td>day</td>
</tr>
<tr>
<td>Dan</td>
<td>Thursday</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
</tr>
<tr>
<td>Haozhe</td>
<td>Friday</td>
</tr>
<tr>
<td>Johannes</td>
<td>Friday</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Query:** "Is the natural join of Orders and Pizza non-empty?"

\[
Q = \bigvee_{c,d,p,i} \text{Orders}(c,d,p) \land \text{Pizza}(p,i)
\]

Query \(Q\) is **hierarchical**

- For any two variables, either their atom sets are disjoint or one is contained in the other.

\[\Rightarrow\] Probability of \(Q\) can be computed in time linear in the database size

[Dalvi and Suciu, 2004]
Query Provenance

<table>
<thead>
<tr>
<th>Orders</th>
<th>Pizza</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer</td>
<td>day</td>
</tr>
<tr>
<td>Dan</td>
<td>Thursday</td>
</tr>
<tr>
<td>Dan</td>
<td>Friday</td>
</tr>
<tr>
<td>Haozhe</td>
<td>Friday</td>
</tr>
<tr>
<td>Johannes</td>
<td>Friday</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ Q = \bigvee_{c,d,p,i} \text{Orders}(c, d, p) \land \text{Pizza}(p, i) \]

The provenance of \( Q \):

\[ (o₁ \land p₁) \lor (o₁ \land p₂) \lor (o₁ \land p₃) \lor (o₂ \land p₁) \lor (o₂ \land p₂) \lor (o₂ \land p₃) \lor (o₃ \land p₄) \lor (o₃ \land p₅) \lor (o₃ \land p₆) \lor (o₄ \land p₄) \lor (o₄ \land p₅) \lor (o₄ \land p₆) \]
The provenance of $Q$ has some **structure**

\[
\begin{align*}
(\bigodot_1 \land \bigcirc_1) \lor (\bigodot_1 \land \bigcirc_2) \lor (\bigodot_1 \land \bigcirc_3) \\
(\bigodot_2 \land \bigcirc_1) \lor (\bigodot_2 \land \bigcirc_2) \lor (\bigodot_2 \land \bigcirc_3) \\
(\bigodot_3 \land \bigcirc_4) \lor (\bigodot_3 \land \bigcirc_5) \lor (\bigodot_3 \land \bigcirc_6) \\
(\bigodot_4 \land \bigcirc_4) \lor (\bigodot_4 \land \bigcirc_5) \lor (\bigodot_4 \land \bigcirc_6)
\end{align*}
\]
The provenance of $Q$ has some structure

$$\begin{align*}
(o_1 \land p_1) \lor (o_1 \land p_2) \lor (o_1 \land p_3) \lor \\
(o_2 \land p_1) \lor (o_2 \land p_2) \lor (o_2 \land p_3) \lor \\
(o_3 \land p_4) \lor (o_3 \land p_5) \lor (o_3 \land p_6) \lor \\
(o_4 \land p_4) \lor (o_4 \land p_5) \lor (o_4 \land p_6)
\end{align*}$$
The provenance of $Q$ has some structure

$$(o_1 \land p_1) \lor (o_1 \land p_2) \lor (o_1 \land p_3) \lor (o_2 \land p_1) \lor (o_2 \land p_2) \lor (o_2 \land p_3) \lor (o_3 \land p_4) \lor (o_3 \land p_5) \lor (o_3 \land p_6) \lor (o_4 \land p_4) \lor (o_4 \land p_5) \lor (o_4 \land p_6)$$
The provenance of Q has some structure

\[(o_1 \land p_1) \lor (o_1 \land p_2) \lor (o_1 \land p_3) \lor (o_2 \land p_1) \lor (o_2 \land p_2) \lor (o_2 \land p_3) \lor (o_3 \land p_4) \lor (o_3 \land p_5) \lor (o_3 \land p_6) \lor (o_4 \land p_4) \lor (o_4 \land p_5) \lor (o_4 \land p_6)\]
The provenance of $Q$ has some structure

$$(o_1 \land p_1) \lor (o_1 \land p_2) \lor (o_1 \land p_3) \lor$$

$$(o_2 \land p_1) \lor (o_2 \land p_2) \lor (o_2 \land p_3) \lor$$

$$(o_3 \land p_4) \lor (o_3 \land p_5) \lor (o_3 \land p_6) \lor$$

$$(o_4 \land p_4) \lor (o_4 \land p_5) \lor (o_4 \land p_6)$$
The provenance of $Q$ has some structure

\[
(\ell_1 \land p_1) \lor (\ell_1 \land p_2) \lor (\ell_1 \land p_3) \lor \\
(\ell_2 \land p_1) \lor (\ell_2 \land p_2) \lor (\ell_2 \land p_3) \lor \\
(\ell_3 \land p_4) \lor (\ell_3 \land p_5) \lor (\ell_3 \land p_6) \lor \\
(\ell_4 \land p_4) \lor (\ell_4 \land p_5) \lor (\ell_4 \land p_6)
\]
The provenance of $Q$ has some structure

$$( o_1 \land p_1 ) \lor ( o_1 \land p_2 ) \lor ( o_1 \land p_3 ) \lor$$

$$( o_2 \land p_1 ) \lor ( o_2 \land p_2 ) \lor ( o_2 \land p_3 ) \lor$$

$$( o_3 \land p_4 ) \lor ( o_3 \land p_5 ) \lor ( o_3 \land p_6 ) \lor$$

$$( o_4 \land p_4 ) \lor ( o_4 \land p_5 ) \lor ( o_4 \land p_6 )$$

The provenance can be factorized:

\[
\left[ o_1 \land [p_1 \lor p_2 \lor p_3] \right] \lor \left[ o_2 \land [p_1 \lor p_2 \lor p_3] \right] \lor \left[ o_3 \land [p_4 \lor p_5 \lor p_6] \right] \lor \left[ o_4 \land [p_4 \lor p_5 \lor p_6] \right]
\]
The provenance of $Q$ has some structure

$$( o_1 \land p_1 ) \lor ( o_1 \land p_2 ) \lor ( o_1 \land p_3 ) \lor$$
$$( o_2 \land p_1 ) \lor ( o_2 \land p_2 ) \lor ( o_2 \land p_3 ) \lor$$
$$( o_3 \land p_4 ) \lor ( o_3 \land p_5 ) \lor ( o_3 \land p_6 ) \lor$$
$$( o_4 \land p_4 ) \lor ( o_4 \land p_5 ) \lor ( o_4 \land p_6 )$$

The provenance can be factorized:

$$[ o_1 \land [ p_1 \lor p_2 \lor p_3 ] ] \lor [ o_2 \land [ p_1 \lor p_2 \lor p_3 ] ] \lor$$
$$[ o_3 \land [ p_4 \lor p_5 \lor p_6 ] ] \lor [ o_4 \land [ p_4 \lor p_5 \lor p_6 ] ]$$

$$\equiv [ [ o_1 \lor o_2 ] \land [ p_1 \lor p_2 \lor p_3 ] ] \lor [ [ o_3 \lor o_4 ] \land [ p_4 \lor p_5 \lor p_6 ] ]$$

This is read-once factorization: every variable appears at most once
- We can compute the factorized provenance directly from the input relations

\[ ([o_1 \lor o_2] \land [p_1 \lor p_2 \lor p_3]) \lor ([o_3 \lor o_4] \land [p_4 \lor p_5 \lor p_6]) \]

Variable order extended by random variables
We can compute the factorized provenance directly from the input relations:

\[
\left[ o_1 \lor o_2 \right] \land \left[ p_1 \lor p_2 \lor p_3 \right] \lor \left[ o_3 \lor o_4 \right] \land \left[ p_4 \lor p_5 \lor p_6 \right]
\]
We can compute the factorized provenance directly from the input relations

\[
\left[ o_1 \lor o_2 \right] \land \left[ p_1 \lor p_2 \lor p_3 \right] \lor \left[ o_3 \lor o_4 \right] \land \left[ p_4 \lor p_5 \lor p_6 \right]
\]

Variable order extended by random variables

- Keep Boolean nodes and provenance variables
We can compute the factorized provenance directly from the input relations:

\[
\left[ (o_1 \lor o_2) \land (p_1 \lor p_2 \lor p_3) \right] \lor \left[ (o_3 \lor o_4) \land (p_4 \lor p_5 \lor p_6) \right]
\]

- Variable order extended by random variables
- Factorization following the variable order
- Keep Boolean nodes and provenance variables
How to compute the probability that the provenance evaluates to true?

\[
\begin{align*}
P_1 &= \prod_{i=1}^{2} (1 - P(o_i)) \\
P_2 &= \prod_{i=1}^{3} (1 - P(p_i)) \\
P_3 &= \prod_{i=3}^{4} (1 - P(o_i)) \\
P_4 &= \prod_{i=4}^{6} (1 - P(p_i)) \\
P_5 &= P_1 \cdot P_2 \\
P_6 &= P_3 \cdot P_4 \\
P_7 &= (1 - P_5) \cdot (1 - P_6)
\end{align*}
\]
How to compute the probability that the provenance evaluates to true?

- Turn $\lor$ into $\oplus$ and $\land$ into $\otimes$
How to compute the probability that the provenance evaluates to true?

- Turn \( \lor \) into \( \oplus \) and \( \land \) into \( \otimes \)
- Compute probabilities of sub-expressions bottom-up
Linear-Time Probability Computation

How to compute the probability that the provenance evaluates to true?

\[ P_1 = 1 - \prod_{i=1}^{2} (1 - P(o_i)) \]

- Turn \( \lor \) into \( \oplus \) and \( \land \) into \( \otimes \)
- Compute probabilities of sub-expressions bottom-up
How to compute the probability that the provenance evaluates to true?

\[
P_1 = 1 - \prod_{i=1}^{2} (1 - P(o_i))
\]

\[
P_2 = 1 - \prod_{i=1}^{3} (1 - P(p_i))
\]

- Turn \(\lor\) into \(\oplus\) and \(\land\) into \(\otimes\)

- Compute probabilities of sub-expressions bottom-up
Linear-Time Probability Computation

How to compute the probability that the provenance evaluates to true?

$P_5 = P_1 \cdot P_2$

$P_1 = 1 - \prod_{i=1}^{2}(1 - P(o_i))$

$P_2 = 1 - \prod_{i=1}^{3}(1 - P(p_i))$

- Turn $\lor$ into $\oplus$ and $\land$ into $\otimes$

- Compute probabilities of sub-expressions bottom-up
How to compute the probability that the provenance evaluates to true?

- **Turn** $\lor$ **into** $\oplus$ **and** $\land$ **into** $\otimes$

- **Compute probabilities of sub-expressions bottom-up**
How to compute the probability that the provenance evaluates to true?

- Turn $\lor$ into $\oplus$ and $\land$ into $\otimes$

- Compute probabilities of sub-expressions bottom-up
How to compute the probability that the provenance evaluates to true?

\[ P_7 = 1 - ((1 - P_5) \cdot (1 - P_6)) \]

- Turn $\lor$ into $\oplus$ and $\land$ into $\otimes$
- Compute probabilities of sub-expressions bottom-up
Use Case: Aggregates
COUNT(*) computed in one pass over the factorisation:

- values $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto \ast$. 
COUNT(*) computed in one pass over the factorisation:

- values $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto \ast$. 
Factorised Aggregate Computation (2/2)

SUM(price) GROUP BY pizza computed in one pass over the factorisation:

- All values except for pizza & price $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto \ast$. 
SUM(price) GROUP BY pizza computed in one pass over the factorisation:

- All values except for pizza & price $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$. 
Sum-Product Ring Abstraction

⇓

Sharing Aggregate Computation
Ring for computing \( \text{SUM}(1) \), \( \text{SUM}(\text{price}) \), \( \text{SUM}(\text{price}) \) GROUP BY pizza:

- Elements = triples, one per aggregate
- Sum (+) and product (*) now defined over triples
  They enable shared computation across the aggregates
Ring for computing $\text{SUM}(1)$, $\text{SUM}(\text{price})$, $\text{SUM}(\text{price})$ GROUP BY pizza:

- Elements $=$ triples, one per aggregate
- Sum ($+$) and product ($*$) now defined over triples
  They enable shared computation across the aggregates
Ring generalisation for the entire covariance matrix

Ring \((\mathcal{R}, +, \ast, 0, 1)\) over triples of aggregates \((c, s, Q) \in \mathcal{R}\):

\[
\begin{pmatrix}
\text{SUM(1)} & \text{SUM}(x_i) & \text{SUM}(x_i \ast x_j)
\end{pmatrix}
\]

\[
(c_1, s_1, Q_1) + (c_2, s_2, Q_2) = (c_1 + c_2, s_1 + s_2, Q_1 + Q_2)
\]

\[
(c_1, s_1, Q_1) \ast (c_2, s_2, Q_2) = (c_1 \cdot c_2, c_2 \cdot s_1 + c_1 \cdot s_2, c_2 \cdot Q_1 + c_1 \cdot Q_2 + s_1 s_2^T + s_2 s_1^T)
\]

\[
0 = (0, 0_{n \times 1}, 0_{n \times n})
\]

\[
1 = (1, 0_{n \times 1}, 0_{n \times n})
\]

- **SUM(1)** reused for all **SUM**(\(x_i\)) and **SUM**(\(x_i \ast x_j\))
- **SUM**(\(x_i\)) reused for all **SUM**(\(x_i \ast x_j\))
Thank you!