Factorized Databases

fdbresearch.github.io

Ahmet Kara (University of Zurich)

Probabilistic Circuits and Logic Workshop
Simons Institute, October 18, 2023

DaST Data·(Systems+Theory)
DEPARTMENT OF INFORMATICS, UNIVERSITY OF ZURICH

Foundations of Factorized Databases

Factorized Representations of Query Results: Size Bounds and Readability.
Dan Olteanu and Jakub Závodný.
In 15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012. ACM. 2012.

Size Bounds for Factorized Representations of Query Results.
Dan Olteanu, and Jakub Závodný.
In ACM Trans. in Database Syst. 40 (1), 2:1-2:44. 2015.

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

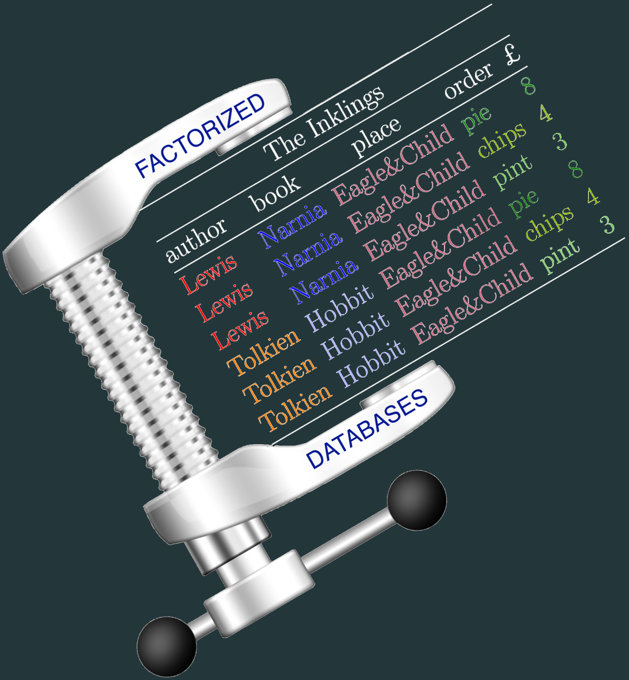
What are Factorized Representations About?

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


FACTORIZED
The Inklings
author book place order £
Lewis Narnia Eagle&Child pie 8
Lewis Narnia Eagle&Child chips 4
Lewis Narnia Eagle&Child pint 3
Tolkien Hobbit Eagle&Child pie 8
Tolkien Hobbit Eagle&Child chips 4
Tolkien Hobbit Eagle&Child pint 3
DATABASES

Ordering Pizzas

Orders

customer	day	pizza
Dan	Thursday	Basilea
Dan	Friday	Basilea
Haozhe	Friday	Hawaii
Johannes	Friday	Hawaii

Pizza

pizza	ingredient
Basilea	garlic
Basilea	tomato
Basilea	mozza
Hawaii	tomato
Hawaii	mozza
Hawaii	pineapple

Ingredients

ingredient	price
garlic	6
tomato	4
mozza	8
pineapple	4

Ordering Pizzas

Orders

customer	day	pizza
Dan	Thursday	Basilea
Dan	Friday	Basilea
Haozhe	Friday	Hawaii
Johannes	Friday	Hawaii

Pizza

pizza	ingredient
Basilea	garlic
Basilea	tomato
Basilea	mozza
Hawaii	tomato
Hawaii	mozza
Hawaii	pineapple

Ingredients

ingredient	price
garlic	6
tomato	4
mozza	8
pineapple	4

Natural join of the above relations:

customer	day	pizza	ingredient	price
Dan	Thursday	Basilea	garlic	6
Dan	Thursday	Basilea	mozza	8
Dan	Thursday	Basilea	tomato	4
Dan	Friday	Basilea	garlic	6
Dan	Friday	Basilea	mozza	8
Dan	Friday	Basilea	tomato	4
...	...	...	...	...

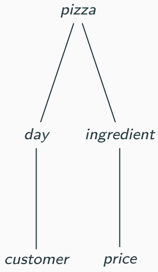
Basileas & Hawaiis in Relational Algebra

customer	day	pizza	ingredient	price
Dan	Thursday	Basilea	garlic	6
Dan	Thursday	Basilea	mozza	8
Dan	Thursday	Basilea	tomato	4
Dan	Friday	Basilea	garlic	6
Dan	Friday	Basilea	mozza	8
Dan	Friday	Basilea	tomato	4
...	...	...	...	...

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

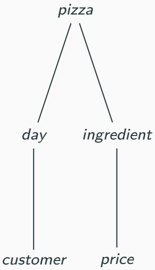
$$
\begin{array}{lllllllllr}
Dan & \times & Thursday & \times & Basilea & \times & \text{garlic} & \times & 6 & \cup \\
Dan & \times & Thursday & \times & Basilea & \times & \text{mozza} & \times & 8 & \cup \\
Dan & \times & Thursday & \times & Basilea & \times & \text{tomato} & \times & 4 & \cup \\
Dan & \times & Friday & \times & Basilea & \times & \text{garlic} & \times & 6 & \cup \\
Dan & \times & Friday & \times & Basilea & \times & \text{mozza} & \times & 8 & \cup \\
Dan & \times & Friday & \times & Basilea & \times & \text{tomato} & \times & 4 & \cup \ldots
\end{array}
$$

Factorized Join

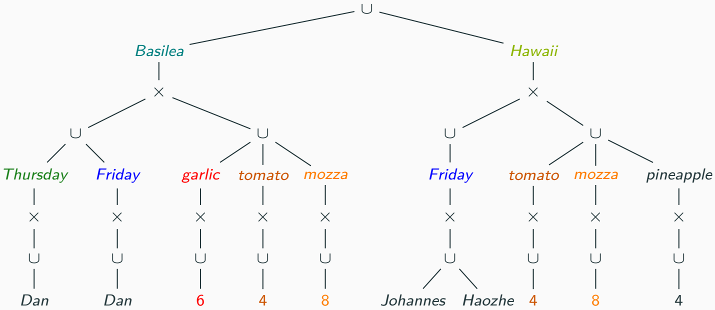


Variable order

Factorized Join

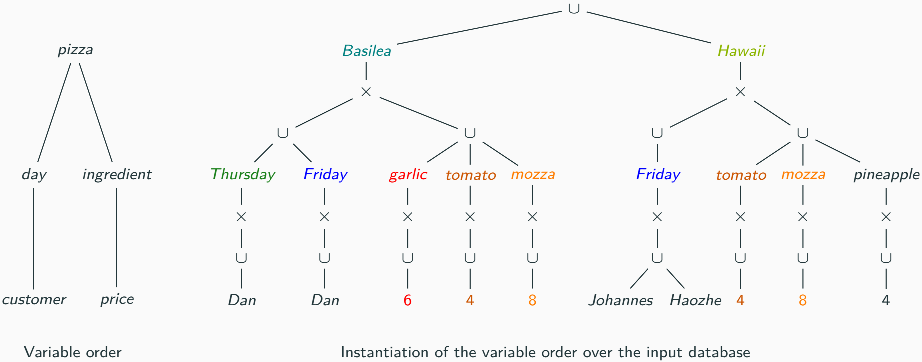


Variable order



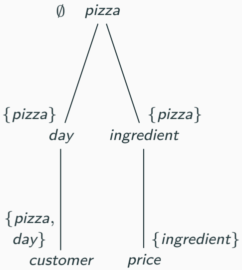
Instantiation of the variable order over the input database

Factorized Join



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

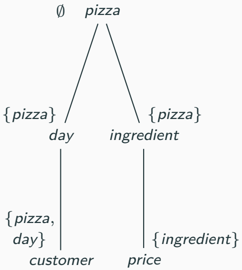
... Now with Further Compression



Observation:

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

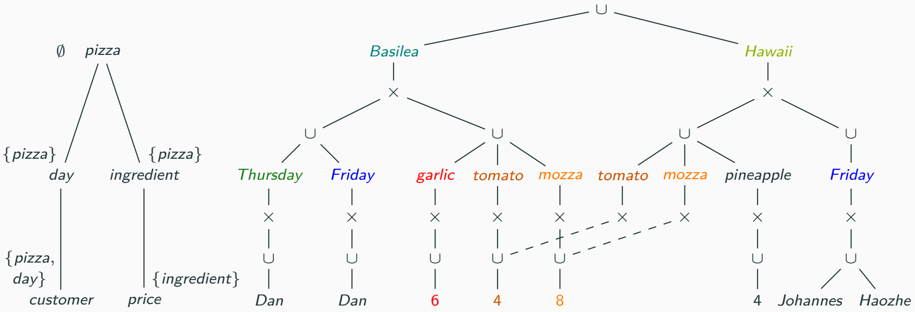
... Now with Further Compression



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!



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

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

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

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

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

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

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

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

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

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

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)

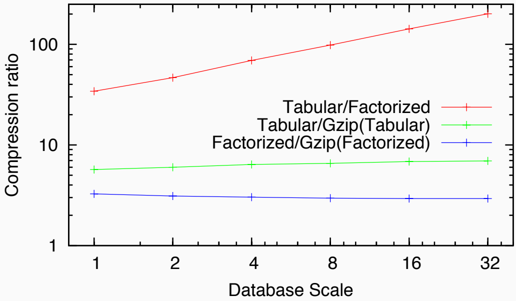
  group-by: linear time if group-by variables on top of all other variables

- updates

  update time depends on the dynamic width of the query

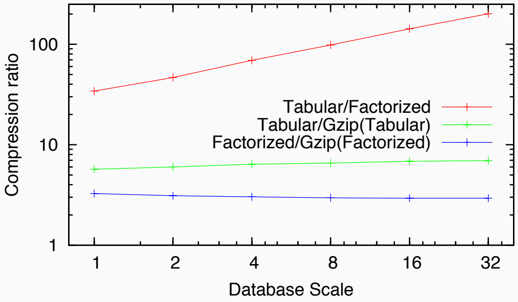
Compression Gains Brought by Factorization

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)

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

Compression Gains in Practice

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 $\boldsymbol{D}$, the result $Q(\boldsymbol{D})$ has a factorized representation with caching of size $\mathcal{O}(|\boldsymbol{D}|^{s^{\uparrow}(Q)})$

Size Bounds for Factorized Representations

[Olteanu and Závodný, 2011-2015]

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

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

Size Bounds for Factorized Representations

[Olteanu and Závodný, 2011-2015]

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

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

- The listing representation can have size $\Omega(|\boldsymbol{D}|^{\rho^*(Q)})$, 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 $\boldsymbol{D}$, the result $Q(\boldsymbol{D})$ has a factorized representation with caching of size $\mathcal{O}(|\boldsymbol{D}|^{s^{\uparrow}(Q)})$

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

- The listing representation can have size $\Omega(|\boldsymbol{D}|^{\rho^*(Q)})$, 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 $\log |\boldsymbol{D}|$ factor)

Factorization Width $s^{\uparrow}$

For any conjunctive query Q:

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

Factorization Width $s^{\uparrow}$

For any conjunctive query Q:

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

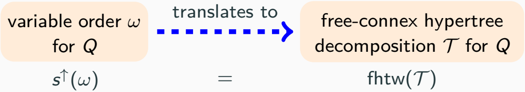
- For any hypertree decomposition $\mathcal{T}$, let $\text{fhtw}(\mathcal{T})$ be the fractional hypertree width of $\mathcal{T}$

Factorization Width $s^{\uparrow}$

For any conjunctive query Q:

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

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

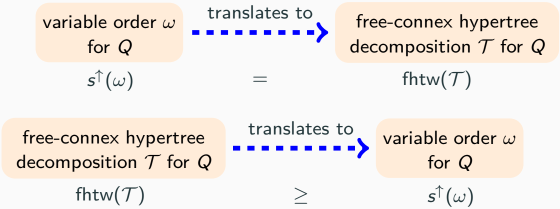


Factorization Width $s^{\uparrow}$

For any conjunctive query Q:

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

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

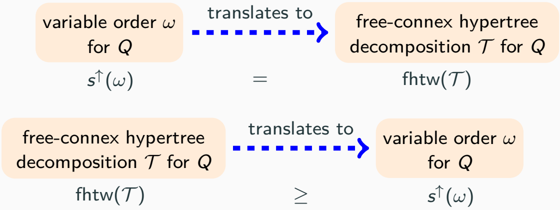


Factorization Width $s^{\uparrow}$

For any conjunctive query Q:

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

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



$\Longrightarrow s^{\uparrow}(Q) = \text{fhtw}(Q)$, where $\text{fhtw}(Q)$ 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

Orders

customer	day	pizza	o.v
Dan	Thursday	Basilea	o_1
Dan	Friday	Basilea	o_2
Haozhe	Friday	Hawaii	o_3
Johannes	Friday	Hawaii	o_4

Pizza

pizza	ingredient	$p.v$
Basilea	garlic	p_1
Basilea	tomato	p_2
Basilea	mozza	p_3
Hawaii	tomato	p_4
Hawaii	mozza	p_5
Hawaii	pineapple	p_6

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

Orders

customer	day	pizza	o.v
Dan	Thursday	Basilea	o_1
Dan	Friday	Basilea	o_2
Haozhe	Friday	Hawaii	o_3
Johannes	Friday	Hawaii	o_4

Pizza

pizza	ingredient	$p.v$
Basilea	garlic	p_1
Basilea	tomato	p_2
Basilea	mozza	p_3
Hawaii	tomato	p_4
Hawaii	mozza	p_5
Hawaii	pineapple	p_6

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

$$Q = \bigvee_{c,d,p,i} \text{Orders}(c, d, p) \wedge \text{Pizza}(p, i)$$

Querying Probabilistic Databases

Orders

customer	day	pizza	o.v
Dan	Thursday	Basilea	o_1
Dan	Friday	Basilea	o_2
Haozhe	Friday	Hawaii	o_3
Johannes	Friday	Hawaii	o_4

Pizza

pizza	ingredient	$p.v$
Basilea	garlic	p_1
Basilea	tomato	p_2
Basilea	mozza	p_3
Hawaii	tomato	p_4
Hawaii	mozza	p_5
Hawaii	pineapple	p_6

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

$$Q = \bigvee_{c,d,p,i} \text{Orders}(c, d, p) \wedge \text{Pizza}(p, i)$$

The query now returns the empty tuple mapped to a probability

Querying Probabilistic Databases

Orders

customer	day	pizza	o.v
Dan	Thursday	Basilea	o_1
Dan	Friday	Basilea	o_2
Haozhe	Friday	Hawaii	o_3
Johannes	Friday	Hawaii	o_4

Pizza

pizza	ingredient	$p.v$
Basilea	garlic	p_1
Basilea	tomato	p_2
Basilea	mozza	p_3
Hawaii	tomato	p_4
Hawaii	mozza	p_5
Hawaii	pineapple	p_6

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

$$Q = \bigvee_{c,d,p,i} \text{Orders}(c, d, p) \wedge \text{Pizza}(p, i)$$

Query Q is hierarchical

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

Querying Probabilistic Databases

Orders

customer	day	pizza	o.v
Dan	Thursday	Basilea	o_1
Dan	Friday	Basilea	o_2
Haozhe	Friday	Hawaii	o_3
Johannes	Friday	Hawaii	o_4

Pizza

pizza	ingredient	$p.v$
Basilea	garlic	p_1
Basilea	tomato	p_2
Basilea	mozza	p_3
Hawaii	tomato	p_4
Hawaii	mozza	p_5
Hawaii	pineapple	p_6

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

$$Q = \bigvee_{c,d,p,i} \text{Orders}(c, d, p) \wedge \text{Pizza}(p, i)$$

Query Q is hierarchical

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

$\Longrightarrow$ Probability of Q can be computed in time linear in the database size

[Dalvi and Suciu, 2004]

Query Provenance

Orders

customer	day	pizza	o.v
Dan	Thursday	Basilea	o_1
Dan	Friday	Basilea	o_2
Haozhe	Friday	Hawaii	o_3
Johannes	Friday	Hawaii	o_4

Pizza

pizza	ingredient	$p.v$
Basilea	garlic	p_1
Basilea	tomato	p_2
Basilea	mozza	p_3
Hawaii	tomato	p_4
Hawaii	mozza	p_5
Hawaii	pineapple	p_6

$$Q = \bigvee_{c,d,p,i} \text{Orders}(c, d, p) \wedge \text{Pizza}(p, i)$$

The provenance of Q:

$$(o_1 \wedge p_1) \vee (o_1 \wedge p_2) \vee (o_1 \wedge p_3) \vee$$
$$(o_2 \wedge p_1) \vee (o_2 \wedge p_2) \vee (o_2 \wedge p_3) \vee$$
$$(o_3 \wedge p_4) \vee (o_3 \wedge p_5) \vee (o_3 \wedge p_6) \vee$$
$$(o_4 \wedge p_4) \vee (o_4 \wedge p_5) \vee (o_4 \wedge p_6)$$

Observing Structure in Query Provenance

- The provenance of Q has some structure

$$
\begin{aligned}
&(o_1 \wedge p_1) \vee (o_1 \wedge p_2) \vee (o_1 \wedge p_3) \vee \\
&(o_2 \wedge p_1) \vee (o_2 \wedge p_2) \vee (o_2 \wedge p_3) \vee \\
&(o_3 \wedge p_4) \vee (o_3 \wedge p_5) \vee (o_3 \wedge p_6) \vee \\
&(o_4 \wedge p_4) \vee (o_4 \wedge p_5) \vee (o_4 \wedge p_6)
\end{aligned}
$$

Observing Structure in Query Provenance

- The provenance of Q has some structure

$$
\begin{aligned}
&(o_1 \wedge p_1) \vee (o_1 \wedge p_2) \vee (o_1 \wedge p_3) \vee \\
&(o_2 \wedge p_1) \vee (o_2 \wedge p_2) \vee (o_2 \wedge p_3) \vee \\
&(o_3 \wedge p_4) \vee (o_3 \wedge p_5) \vee (o_3 \wedge p_6) \vee \\
&(o_4 \wedge p_4) \vee (o_4 \wedge p_5) \vee (o_4 \wedge p_6)
\end{aligned}
$$

Observing Structure in Query Provenance

- The provenance of Q has some structure

$$
\begin{aligned}
&(o_1 \wedge p_1) \vee (o_1 \wedge p_2) \vee (o_1 \wedge p_3) \vee \\
&(o_2 \wedge p_1) \vee (o_2 \wedge p_2) \vee (o_2 \wedge p_3) \vee \\
&(o_3 \wedge p_4) \vee (o_3 \wedge p_5) \vee (o_3 \wedge p_6) \vee \\
&(o_4 \wedge p_4) \vee (o_4 \wedge p_5) \vee (o_4 \wedge p_6)
\end{aligned}
$$

Observing Structure in Query Provenance

- The provenance of Q has some structure

$$
\begin{aligned}
&(o_1 \wedge p_1) \vee (o_1 \wedge p_2) \vee (o_1 \wedge p_3) \vee \\
&(o_2 \wedge p_1) \vee (o_2 \wedge p_2) \vee (o_2 \wedge p_3) \vee \\
&(o_3 \wedge p_4) \vee (o_3 \wedge p_5) \vee (o_3 \wedge p_6) \vee \\
&(o_4 \wedge p_4) \vee (o_4 \wedge p_5) \vee (o_4 \wedge p_6)
\end{aligned}
$$

Observing Structure in Query Provenance

- The provenance of Q has some structure

$$
\begin{aligned}
&(o_1 \wedge p_1) \vee (o_1 \wedge p_2) \vee (o_1 \wedge p_3) \vee \\
&(o_2 \wedge p_1) \vee (o_2 \wedge p_2) \vee (o_2 \wedge p_3) \vee \\
&(o_3 \wedge p_4) \vee (o_3 \wedge p_5) \vee (o_3 \wedge p_6) \vee \\
&(o_4 \wedge p_4) \vee (o_4 \wedge p_5) \vee (o_4 \wedge p_6)
\end{aligned}
$$

Observing Structure in Query Provenance

- The provenance of Q has some structure

$$
\begin{aligned}
&(o_1 \wedge p_1) \vee (o_1 \wedge p_2) \vee (o_1 \wedge p_3) \vee \\
&(o_2 \wedge p_1) \vee (o_2 \wedge p_2) \vee (o_2 \wedge p_3) \vee \\
&(o_3 \wedge p_4) \vee (o_3 \wedge p_5) \vee (o_3 \wedge p_6) \vee \\
&(o_4 \wedge p_4) \vee (o_4 \wedge p_5) \vee (o_4 \wedge p_6)
\end{aligned}
$$

Observing Structure in Query Provenance

- The provenance of Q has some structure

$$
\begin{aligned}
&(o_1 \land p_1) \lor (o_1 \land p_2) \lor (o_1 \land p_3) \lor \\
&(o_2 \land p_1) \lor (o_2 \land p_2) \lor (o_2 \land p_3) \lor \\
&(o_3 \land p_4) \lor (o_3 \land p_5) \lor (o_3 \land p_6) \lor \\
&(o_4 \land p_4) \lor (o_4 \land p_5) \lor (o_4 \land p_6)
\end{aligned}
$$

- The provenance can be factorized:

$$
\begin{aligned}
&\Big[o_1 \land [p_1 \lor p_2 \lor p_3]\Big] \lor \Big[o_2 \land [p_1 \lor p_2 \lor p_3]\Big] \lor \\
&\Big[o_3 \land [p_4 \lor p_5 \lor p_6]\Big] \lor \Big[o_4 \land [p_4 \lor p_5 \lor p_6]\Big]
\end{aligned}
$$

Observing Structure in Query Provenance

- The provenance of Q has some structure

$$(o_1 \wedge p_1) \vee (o_1 \wedge p_2) \vee (o_1 \wedge p_3) \vee$$
$$(o_2 \wedge p_1) \vee (o_2 \wedge p_2) \vee (o_2 \wedge p_3) \vee$$
$$(o_3 \wedge p_4) \vee (o_3 \wedge p_5) \vee (o_3 \wedge p_6) \vee$$
$$(o_4 \wedge p_4) \vee (o_4 \wedge p_5) \vee (o_4 \wedge p_6)$$

- The provenance can be factorized:

$$\Big[o_1 \wedge [p_1 \vee p_2 \vee p_3]\Big] \vee \Big[o_2 \wedge [p_1 \vee p_2 \vee p_3]\Big] \vee$$
$$\Big[o_3 \wedge [p_4 \vee p_5 \vee p_6]\Big] \vee \Big[o_4 \wedge [p_4 \vee p_5 \vee p_6]\Big]$$

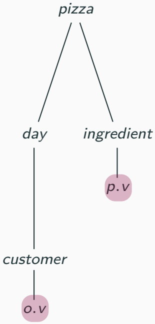
$$\equiv \Big[[o_1 \vee o_2] \wedge [p_1 \vee p_2 \vee p_3]\Big] \vee \Big[[o_3 \vee o_4] \wedge [p_4 \vee p_5 \vee p_6]\Big]$$

- This is read-once factorization: every variable appears at most once

Computing Factorized Provenance from Input Relations

- We can compute the factorized provenance directly from the input relations

$$\Big[[o_1 \vee o_2] \wedge [p_1 \vee p_2 \vee p_3]\Big] \vee \Big[[o_3 \vee o_4] \wedge [p_4 \vee p_5 \vee p_6]\Big]$$

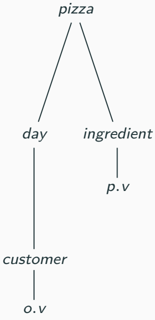


Variable order extended by random variables

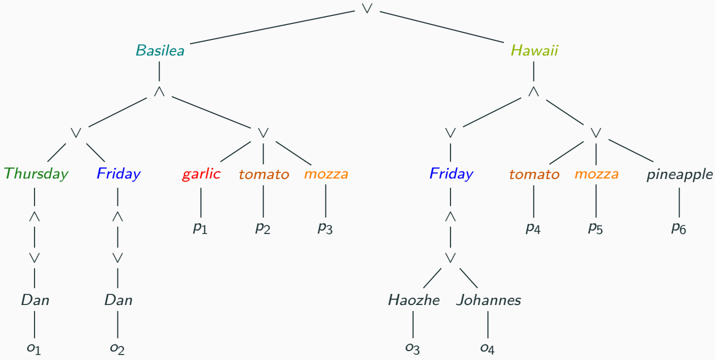
Computing Factorized Provenance from Input Relations

- We can compute the factorized provenance directly from the input relations

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



Variable order extended by random variables

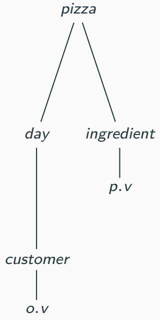


Factorization following the variable order

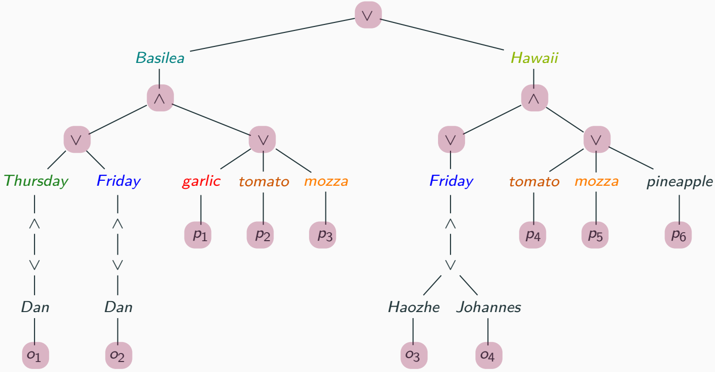
Computing Factorized Provenance from Input Relations

- We can compute the factorized provenance directly from the input relations

$$\Big[[o_1 \vee o_2] \wedge [p_1 \vee p_2 \vee p_3]\Big] \vee \Big[[o_3 \vee o_4] \wedge [p_4 \vee p_5 \vee p_6]\Big]$$



Variable order extended by random variables



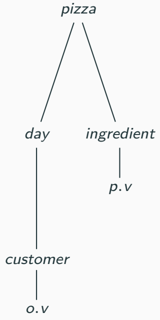
Factorization following the variable order

- Keep Boolean nodes and provenance variables

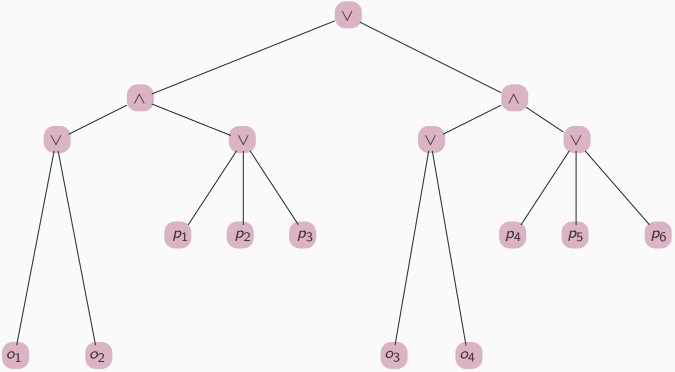
Computing Factorized Provenance from Input Relations

- We can compute the factorized provenance directly from the input relations

$$\Big[[o_1 \vee o_2] \wedge [p_1 \vee p_2 \vee p_3]\Big] \vee \Big[[o_3 \vee o_4] \wedge [p_4 \vee p_5 \vee p_6]\Big]$$



Variable order extended by random variables

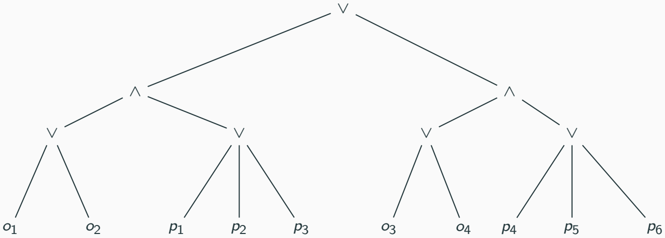


Factorization following the variable order

- Keep Boolean nodes and provenance variables

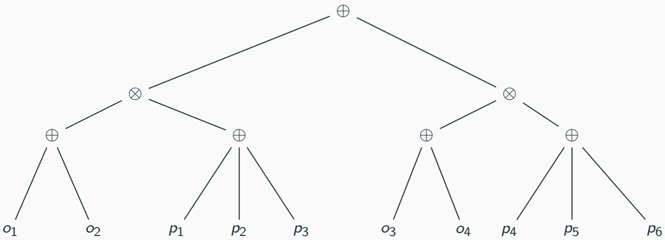
Linear-Time Probability Computation

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



Linear-Time Probability Computation

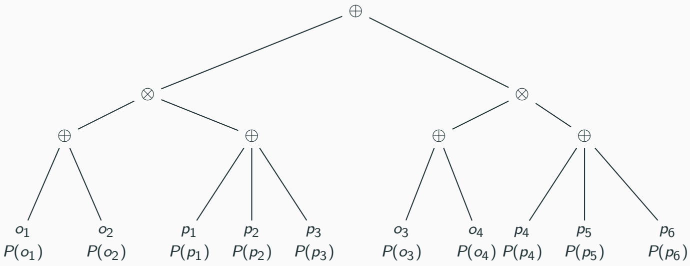
How to compute the probability that the provenance evaluates to true?



- Turn $\vee$ into $\oplus$ and $\wedge$ into $\otimes$

Linear-Time Probability Computation

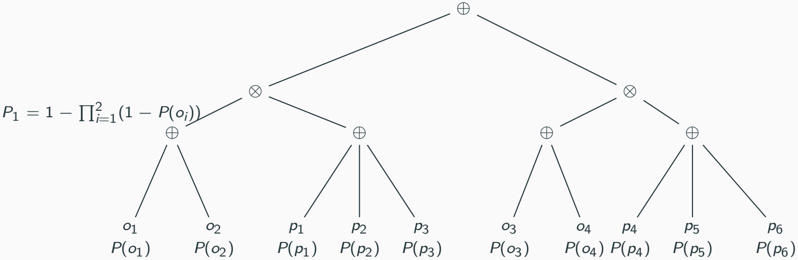
How to compute the probability that the provenance evaluates to true?



- Turn $\vee$ into $\oplus$ and $\wedge$ into $\otimes$
- Compute probabilities of sub-expressions bottom-up

Linear-Time Probability Computation

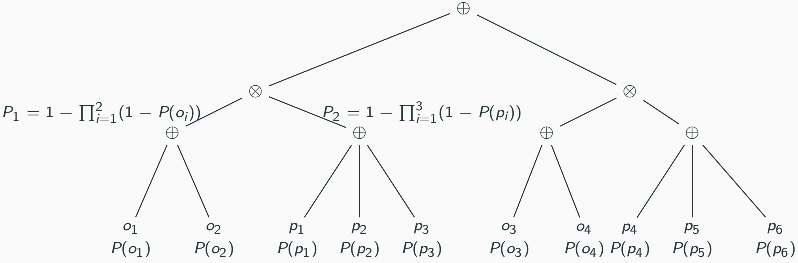
How to compute the probability that the provenance evaluates to true?



- Turn $\vee$ into $\oplus$ and $\wedge$ into $\otimes$
- Compute probabilities of sub-expressions bottom-up

Linear-Time Probability Computation

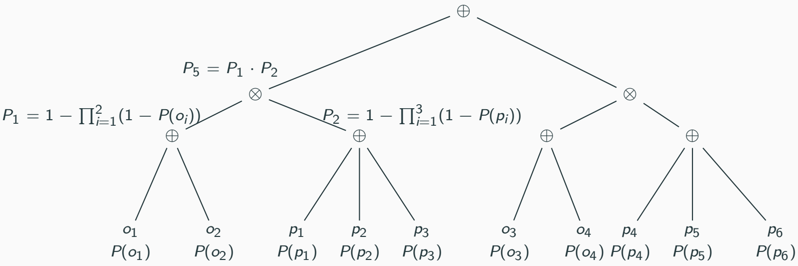
How to compute the probability that the provenance evaluates to true?



- Turn $\vee$ into $\oplus$ and $\wedge$ into $\otimes$
- Compute probabilities of sub-expressions bottom-up

Linear-Time Probability Computation

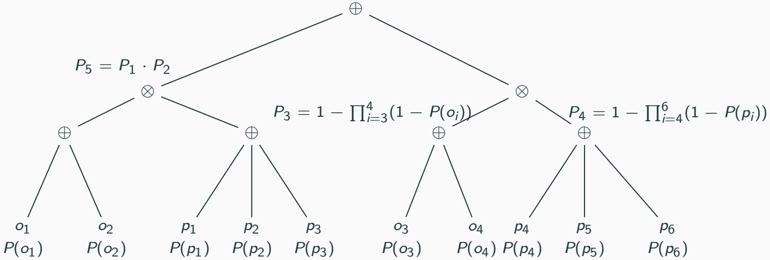
How to compute the probability that the provenance evaluates to true?



- Turn $\vee$ into $\oplus$ and $\wedge$ into $\otimes$
- Compute probabilities of sub-expressions bottom-up

Linear-Time Probability Computation

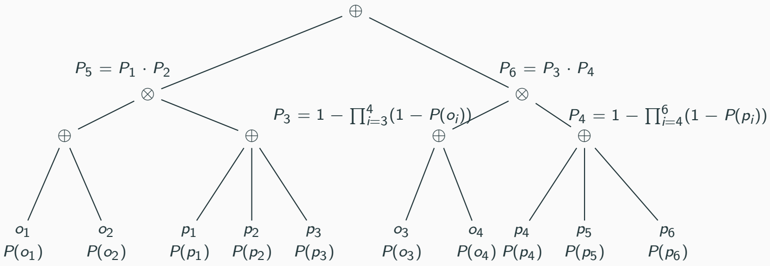
How to compute the probability that the provenance evaluates to true?



- Turn $\vee$ into $\oplus$ and $\wedge$ into $\otimes$
- Compute probabilities of sub-expressions bottom-up

Linear-Time Probability Computation

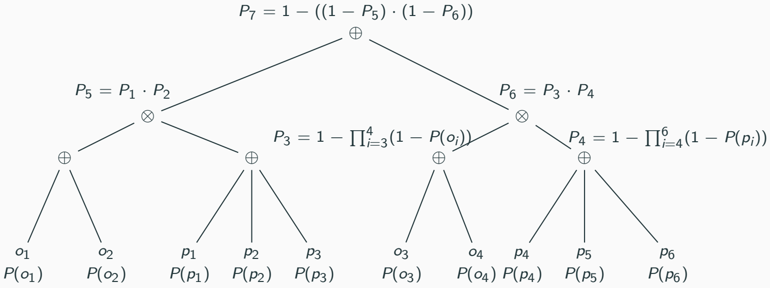
How to compute the probability that the provenance evaluates to true?



- Turn $\vee$ into $\oplus$ and $\wedge$ into $\otimes$
- Compute probabilities of sub-expressions bottom-up

Linear-Time Probability Computation

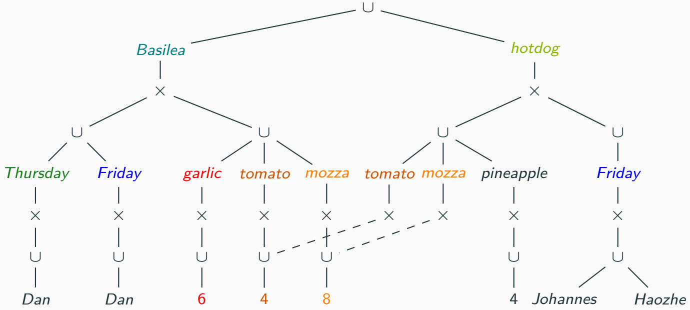
How to compute the probability that the provenance evaluates to true?



- Turn $\vee$ into $\oplus$ and $\wedge$ into $\otimes$
- Compute probabilities of sub-expressions bottom-up

Use Case: Aggregates

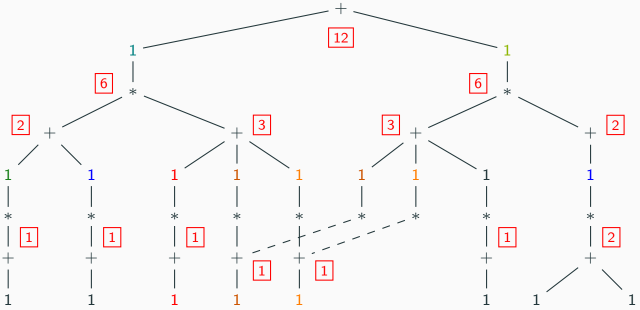
Factorised Aggregate Computation (1/2)



COUNT(*) computed in one pass over the factorisation:

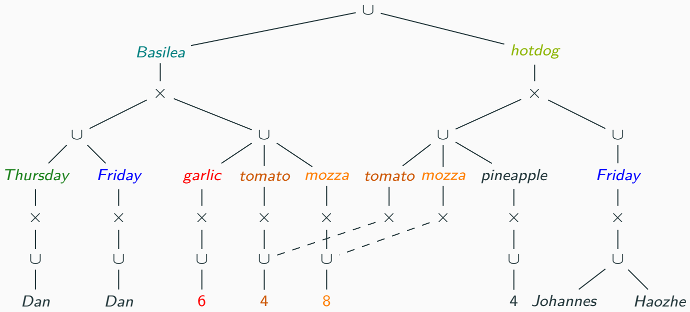
- values $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$.

Factorised Aggregate Computation (1/2)



COUNT(*) computed in one pass over the factorisation:

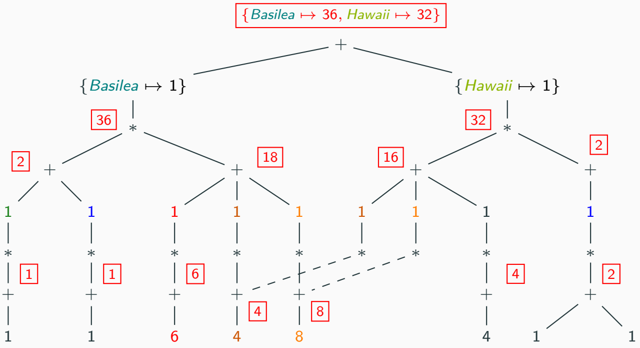
- values $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$.



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

- All values except for `pizza` & `price` $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$.

Factorising the Computation of Aggregates (2/2)



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

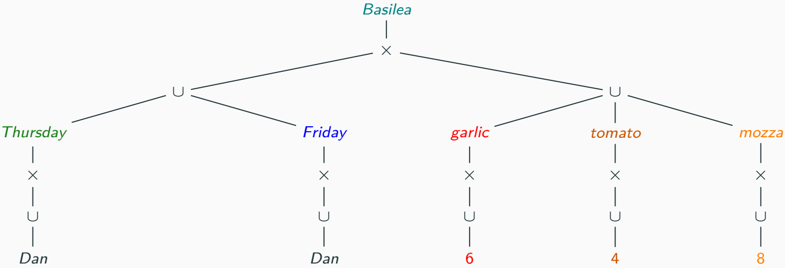
- All values except for pizza & price $\mapsto 1$,
- $\cup \mapsto +$, $\times \mapsto *$.

Sum-Product Ring Abstraction

$\Downarrow$

Sharing Aggregate Computation

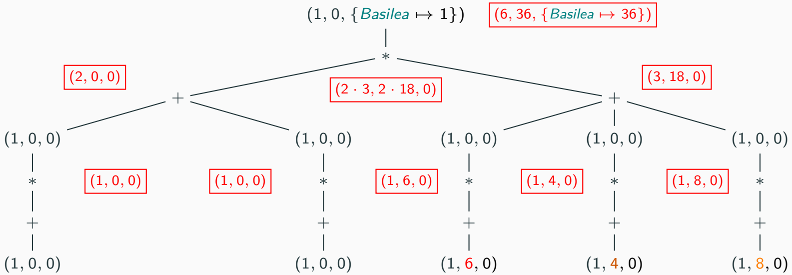
Shared Computation of Several Aggregates (1/2)



Ring for computing `SUM(1)`, `SUM(price)`, `SUM(price) GROUP BY pizza`:

- Elements = triples, one per aggregate
- Sum (+) and product (*) now defined over triples
  They enable shared computation across the aggregates

Shared Computation of Several Aggregates (2/2)

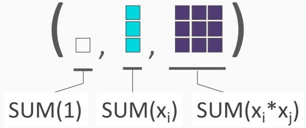


Ring for computing `SUM(1), SUM(price), SUM(price) GROUP BY pizza`:

- 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 $(\mathcal{R}, +, *, \mathbf{0}, \mathbf{1})$ over triples of aggregates $(c, \mathbf{s}, \mathbf{Q}) \in \mathcal{R}$:



$$(c_1, \mathbf{s}_1, \mathbf{Q}_1) + (c_2, \mathbf{s}_2, \mathbf{Q}_2) = (c_1 + c_2, \mathbf{s}_1 + \mathbf{s}_2, \mathbf{Q}_1 + \mathbf{Q}_2)$$

$$(c_1, \mathbf{s}_1, \mathbf{Q}_1) * (c_2, \mathbf{s}_2, \mathbf{Q}_2) = (c_1 \cdot c_2, c_2 \cdot \mathbf{s}_1 + c_1 \cdot \mathbf{s}_2, c_2 \cdot \mathbf{Q}_1 + c_1 \cdot \mathbf{Q}_2 + \mathbf{s}_1 \mathbf{s}_2^T + \mathbf{s}_2 \mathbf{s}_1^T)$$

$$\mathbf{0} = (0, \mathbf{0}_{n \times 1}, \mathbf{0}_{n \times n})$$

$$\mathbf{1} = (1, \mathbf{0}_{n \times 1}, \mathbf{0}_{n \times n})$$

- `SUM(1)` reused for all `SUM`(x_i) and `SUM`$(x_i * x_j)$
- `SUM`(x_i) reused for all `SUM`$(x_i * x_j)$

Thank you!