Provenance in Databases

… and Links to Knowledge Compilation

Antoine Amarilli

December 17, 2019

Kocoon Workshop

TELECOM Paris

INSTITUT POLYTECHNIQUE DE PARIS IP PARIS

Provenance management

- Common task on databases: query evaluation

Provenance management

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

Provenance management

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

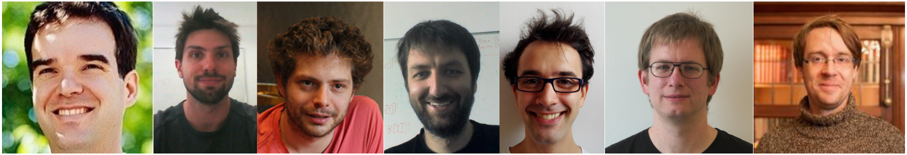
Goal of this talk

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

Goal of this talk

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

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



Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion

Relational DBMSs

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

ORACLE
Microsoft
IBM
SQLite
MySQL
MariaDB
PostgreSQL

Example

Guest

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

Reservation

id	guest	room	arrival	nights
1	1	504	2019-01-01	5
2	2	107	2019-01-10	3
3	3	302	2019-01-15	6
4	2	504	2019-01-15	2
5	2	107	2019-01-30	1

Relations and databases

Formally:

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

The positive relational algebra

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

Relation name

Guest

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

Reservation

id	guest	room	arrival	nights
1	1	504	2019-01-01	5
2	2	107	2019-01-10	3
3	3	302	2019-01-15	6
4	2	504	2019-01-15	2
5	2	107	2019-01-30	1

Expression: `Guest`

Result:

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

Renaming

Guest

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

Reservation

id	guest	room	arrival	nights
1	1	504	2019-01-01	5
2	2	107	2019-01-10	3
3	3	302	2019-01-15	6
4	2	504	2019-01-15	2
5	2	107	2019-01-30	1

Expression: $\rho_{\texttt{id}\to\texttt{guest}}(\texttt{Guest})$

Result:

guest	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

Projection

Guest

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

Reservation

id	guest	room	arrival	nights
1	1	504	2019-01-01	5
2	2	107	2019-01-10	3
3	3	302	2019-01-15	6
4	2	504	2019-01-15	2
5	2	107	2019-01-30	1

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

Result:

email	id
john.smith@gmail.com	1
alice@black.name	2
john.smith@ens.fr	3

Selection

Guest

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

Reservation

id	guest	room	arrival	nights
1	1	504	2019-01-01	5
2	2	107	2019-01-10	3
3	3	302	2019-01-15	6
4	2	504	2019-01-15	2
5	2	107	2019-01-30	1

Expression: $\sigma_{\texttt{arrival}>\text{2019-01-12}\wedge\texttt{guest}=\texttt{2}}(\texttt{Reservation})$

Result:

id	guest	room	arrival	nights
4	2	504	2019-01-15	2
5	2	107	2019-01-30	1

The formula used in the selection can be any **Boolean combination** of **comparisons** of attributes to attributes or constants

Cross product

Guest

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

Reservation

id	guest	room	arrival	nights
1	1	504	2019-01-01	5
2	2	107	2019-01-10	3
3	3	302	2019-01-15	6
4	2	504	2019-01-15	2
5	2	107	2019-01-30	1

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

Result:

id	name
1	Alice Black
2	Alice Black
3	Alice Black
1	John Smith
2	John Smith
3	John Smith

Natural join

Guest

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

Reservation

id	guest	room	arrival	nights
1	1	504	2019-01-01	5
2	2	107	2019-01-10	3
3	3	302	2019-01-15	6
4	2	504	2019-01-15	2
5	2	107	2019-01-30	1

Not a basic operator, but a **useful shorthand!**

Expression: $\texttt{Reservation} \bowtie \rho_{\texttt{id}\to\texttt{guest}}(\texttt{Guest})$

Result:

id	guest	room	arrival	nights	name	email
1	1	504	2019-01-01	5	John Smith	john.smith@gmail.com
2	2	107	2019-01-10	3	Alice Black	alice@black.name
3	3	302	2019-01-15	6	John Smith	john.smith@ens.fr
4	2	504	2019-01-15	2	Alice Black	alice@black.name
5	2	107	2019-01-30	1	Alice Black	alice@black.name

Equivalent to:

$\Pi_{\texttt{id},\texttt{guest},\texttt{room},\texttt{arrival},\texttt{nights},\texttt{name},\texttt{email}}(\sigma_{\texttt{temp}=\texttt{guest}}(\rho_{\texttt{id}\to\texttt{temp}}(\texttt{Guest}) \times \texttt{Reservation}))$.

Union

Guest

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

Reservation

id	guest	room	arrival	nights
1	1	504	2019-01-01	5
2	2	107	2019-01-10	3
3	3	302	2019-01-15	6
4	2	504	2019-01-15	2
5	2	107	2019-01-30	1

Expression: $\Pi_{\texttt{room}}(\sigma_{\texttt{guest}=2}(\texttt{Reservation})) \cup \Pi_{\texttt{room}}(\sigma_{\texttt{arrival}=\text{2019-01-15}}(\texttt{Reservation}))$

Result:

room
107
302
504

Relational algebra vs relational calculus

Sometimes we write tuples as ground facts rather than tables

Guest

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

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

Relational algebra vs relational calculus

Sometimes we write tuples as ground facts rather than tables

Guest

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

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

Sometimes we write queries in relational calculus rather than algebra

$$\Pi_{\texttt{id}}(\texttt{Guest}) \times \Pi_{\texttt{name}}(\texttt{Guest})$$

$$Q(x, y') : \exists y\, z\, x'\, z' \;\; \mathit{Guest}(x, y, z) \wedge \mathit{Guest}(x', y', z')$$

Relational algebra vs relational calculus

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

Guest

id	name	email
1	John Smith	john.smith@gmail.com
2	Alice Black	alice@black.name
3	John Smith	john.smith@ens.fr

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

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

$$\Pi_{\texttt{id}}(\texttt{Guest}) \times \Pi_{\texttt{name}}(\texttt{Guest})$$

$$Q(x, y') : \exists y\, z\, x'\, z'\ \mathit{Guest}(x, y, z) \wedge \mathit{Guest}(x', y', z')$$

→ Relational algebra and calculus have the **same expressive power**!

Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion

Data model

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

Data model

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

name	position	city	classification
John	Director	New York	unclassified
Paul	Janitor	New York	restricted
Dave	Analyst	Paris	confidential
Ellen	Field agent	Berlin	secret
Magdalen	Double agent	Paris	top secret
Nancy	HR director	Paris	restricted
Susan	Analyst	Berlin	secret

Data model

- **Relational data model**: data decomposed into relations, with labeled attributes…
- … with an extra **provenance annotation** for each tuple (think of it as a Boolean variable)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

Boolean valuations

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

Example of possible worlds

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

$$\nu\colon \quad \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ \top & \top & \top & \top & \top & \top & \top \end{array}$$

Example of possible worlds

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Dave	Analyst	Paris	confidential	x_3
Magdalen	Double agent	Paris	top secret	x_5
Susan	Analyst	Berlin	secret	x_7

$$\nu: \quad \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ \top & \bot & \top & \bot & \top & \bot & \top \end{array}$$

Boolean provenance of query results

- Goal: Evaluate a positive relational algebra query Q on a database D...

Boolean provenance of query results

- Goal: Evaluate a positive relational algebra query Q on a database D... whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$

Boolean provenance of query results

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

Boolean provenance of query results

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

Boolean provenance of query results

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

Boolean provenance of query results

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

Example (What cities are in the table?)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
New York	$x_1 \lor x_2$
Paris	$x_3 \lor x_5 \lor x_6$
Berlin	$x_4 \lor x_7$

Boolean provenance of query results

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

Example (What cities are in the table?)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
New York	$x_1 \lor x_2$
Paris	$x_3 \lor x_5 \lor x_6$
Berlin	$x_4 \lor x_7$

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

Selection, renaming

Provenance annotations of selected tuples are **unchanged**

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

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

n	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2

Projection

Take the OR of provenance annotations of identical, merged tuples

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

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
New York	$x_1 \vee x_2$
Paris	$x_3 \vee x_5 \vee x_6$
Berlin	$x_4 \vee x_7$

Union

Take the OR of provenance annotations of identical, merged tuples

Example

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

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
Paris	$x_3 \vee x_5$
Berlin	$x_4 \vee x_7$

Cross product

Take the AND of provenance annotations of combined tuples

Example

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

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
Paris	$x_3 \land x_5$
Berlin	$x_4 \land x_7$

How is provenance actually represented?

Provenance annotations are Boolean functions

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

Example

Is there a city with two different agents?

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_6) \vee (x_3 \wedge x_5) \vee (x_4 \wedge x_7) \vee (x_5 \wedge x_6)$$

Theorem (PTIME overhead)

For any fixed positive relational algebra expression, given an input database, we can compute in PTIME the provenance annotation of every tuple in the result

Other representation: Provenance circuits

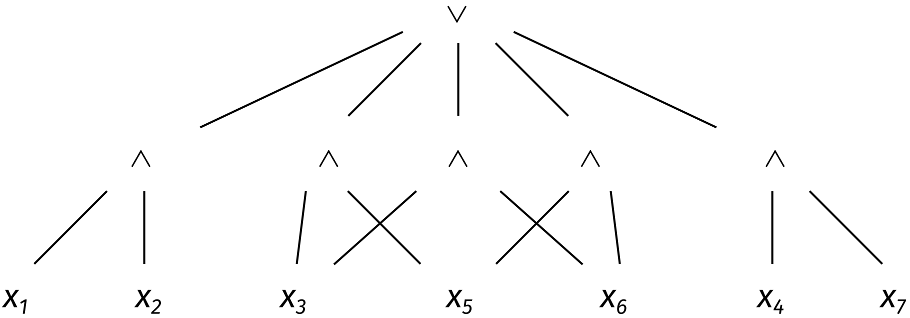
[Deutch, Milo, Roy, and Tannen 2014]

- Use **Boolean circuits** to represent provenance
- Every time an operation reuses a previously computed result, link to the **previously created circuit gate**
- **Never larger** than provenance formulas
- Sometimes **more concise**: provenance circuits can be…

Other representation: Provenance circuits [Deutch, Milo, Roy, and Tannen 2014]

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

Example provenance circuit



What can we do with Boolean provenance?

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_6) \vee (x_3 \wedge x_5) \vee (x_4 \wedge x_7) \vee (x_5 \wedge x_6)$$

- The provenance describes, for each result tuple, the subsets of the input database for which it appears in the query result

What can we do with Boolean provenance?

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

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

What can we do with Boolean provenance?

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

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

What can we do with Boolean provenance?

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_6) \vee (x_3 \wedge x_5) \vee (x_4 \wedge x_7) \vee (x_5 \wedge x_6)$$

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

Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion

Commutative semiring $(K, \mathbb{0}, \mathbb{1}, \oplus, \otimes)$

- Set K with distinguished elements $\mathbb{0}$, $\mathbb{1}$
- $\oplus$ associative, commutative operator, with identity $\mathbb{0}_K$:
  - $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
  - $a \oplus b = b \oplus a$
  - $a \oplus \mathbb{0} = \mathbb{0} \oplus a = a$
- $\otimes$ associative, commutative operator, with identity $\mathbb{1}_K$:
  - $a \otimes (b \otimes c) = (a \otimes b) \otimes c$
  - $a \otimes b = b \otimes a$
  - $a \otimes \mathbb{1} = \mathbb{1} \otimes a = a$
- $\otimes$ distributes over $\oplus$:

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

- $\mathbb{0}$ is annihilating for $\otimes$:

$$a \otimes \mathbb{0} = \mathbb{0} \otimes a = \mathbb{0}$$

Example semirings

- $(\mathbb{N}, 0, 1, +, \times)$: **counting** semiring
- $(\{\bot, \top\}, \bot, \top, \vee, \wedge)$: **Boolean** semiring
- $(\{\mathit{unclassified}, \mathit{restricted}, \mathit{confidential}, \mathit{secret}, \mathit{top\ secret}\}, \mathit{top\ secret}, \mathit{unclassified}, \min, \max)$: **security** semiring
- $(\mathbb{N} \cup \{\infty\}, \infty, 0, \min, +)$: **tropical** semiring
- $(\{\text{Boolean functions over } \mathcal{X}\}, \bot, \top, \vee, \wedge)$: semiring of **Boolean functions** over $\mathcal{X}$
- $(\mathbb{N}[\mathcal{X}], 0, 1, +, \times)$: semiring of integer-valued **polynomials** with variables in $\mathcal{X}$ (also called **How**-semiring or **universal** semiring)

Semiring provenance [Green, Karvounarakis, and Tannen 2007]

- We fix a semiring $(K, \mathbb{0}, \mathbb{1}, \oplus, \otimes)$
- We assume provenance annotations are in K
- We consider a query Q from the positive relational algebra (selection, projection, renaming, product, union)
- We define a semantics for the provenance of a tuple $t \in Q(D)$ inductively on the structure of Q just like before

Selection, renaming

Provenance annotations of selected tuples are **unchanged**

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

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

n	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2

Projection

Provenance annotations of identical, merged, tuples are $\oplus$-ed

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

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
New York	$x_1 \oplus x_2$
Paris	$x_3 \oplus x_5 \oplus x_6$
Berlin	$x_4 \oplus x_7$

Union

Provenance annotations of identical, merged, tuples are $\oplus$-ed

Example

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

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
Paris	$x_3 \oplus x_5$
Berlin	$x_4 \oplus x_7$

Cross product

Provenance annotations of combined tuples are $\otimes$-ed

Example

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

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
Paris	$x_3 \otimes x_5$
Berlin	$x_4 \otimes x_7$

What can we do with semiring provenance?

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

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

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

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

Boolean functions: Boolean provenance, as previously defined

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

Example of security provenance

$$\pi_{\text{city}}(\sigma_{\text{name}<\text{name2}}(\pi_{\text{name,city}}(R) \bowtie \rho_{\text{name}\to\text{name2}}(\pi_{\text{name,city}}(R))))$$

name	position	city	prov
John	Director	New York	unclassified
Paul	Janitor	New York	restricted
Dave	Analyst	Paris	confidential
Ellen	Field agent	Berlin	secret
Magdalen	Double agent	Paris	top secret
Nancy	HR director	Paris	restricted
Susan	Analyst	Berlin	secret

city	prov
New York	restricted
Paris	confidential
Berlin	secret

Properties [Green, Karvounarakis, and Tannen 2007]

- Semiring provenance still has PTIME data overhead

Properties [Green, Karvounarakis, and Tannen 2007]

- Semiring provenance still has PTIME data overhead
- Semiring homomorphisms commute with provenance computation: if $K \xrightarrow{\text{hom}} K'$, then one can compute the provenance in K, apply the homomorphism, and obtain the same result as when computing provenance in K'

Properties [Green, Karvounarakis, and Tannen 2007]

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

Properties [Green, Karvounarakis, and Tannen 2007]

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

Properties [Green, Karvounarakis, and Tannen 2007]

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

Extensions

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

Extensions

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

Extensions

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

Extensions

- Beyond positive relational algebra…
    - Allow **relational difference**: need a semiring with **monus**, but complicated semantics [Amer 1984; Geerts and Poggi 2010; Amsterdamer, Deutch, and Tannen 2011a; Amarilli and Monet 2016]
    - Allow **aggregate queries**: extend semirings to **semimodules** [Amsterdamer, Deutch, and Tannen 2011b; Fink, Han, and Olteanu 2012]
    - Allow **recursive queries**: representation as **formal power series** or **cycluits** [Amarilli, Bourhis, Monet, and Senellart 2017]
- Beyond semiring provenance…
    - **Where-provenance**: capture which output **value** comes from which input **value** [Buneman, Khanna, and Tan 2001]

Extensions

- Beyond positive relational algebra…
    - Allow relational difference: need a semiring with monus, but complicated semantics [Amer 1984; Geerts and Poggi 2010; Amsterdamer, Deutch, and Tannen 2011a; Amarilli and Monet 2016]
    - Allow aggregate queries: extend semirings to semimodules [Amsterdamer, Deutch, and Tannen 2011b; Fink, Han, and Olteanu 2012]
    - Allow recursive queries: representation as formal power series or cycluits [Amarilli, Bourhis, Monet, and Senellart 2017]
- Beyond semiring provenance…
    - Where-provenance: capture which output value comes from which input value [Buneman, Khanna, and Tan 2001]
    - Why-not provenance: capture why an output tuple was not produced, usually as a function of the query [Chapman and Jagadish 2009]

Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

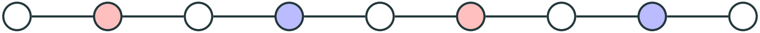
Applications to Enumeration

Conclusion

Motivation and definition

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

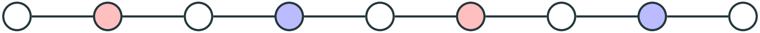
Assume our data is a sequence of events:



Motivation and definition

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

Assume our data is a sequence of events:

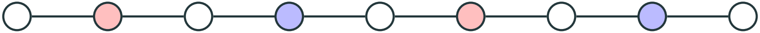


- Formal model: a **word** where each node has a **color**

Motivation and definition

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

Assume our data is a sequence of events:

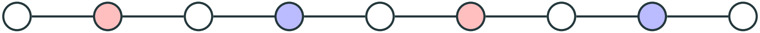


- Formal model: a word where each node has a color
- We could represent this in the relational setting:
  - One 2-ary table for the successor relation
  - One 1-ary table to list the nodes for each color

Motivation and definition

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

Assume our data is a sequence of events:

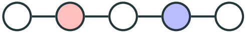


- Formal model: a **word** where each node has a **color**
- We could represent this in the relational setting:
  - One 2-ary table for the **successor relation**
  - One 1-ary table to list the nodes for each **color**
- Some natural queries **cannot be expressed** in relational algebra!

  → *“Is there a blue node after each pink node?”*

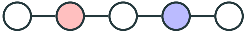
Query evaluation on words

Database: a **word** w where nodes have a color from an alphabet ○ 🔴 🔵



Query evaluation on words

Database: a **word** w where nodes have a color from an alphabet ○ 🔴 🔵

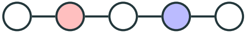


Query Q: a **sentence** (YES/NO question) in **monadic second-order logic** (MSO) *(to be defined)*

“Is there a blue node after each pink node?”

Query evaluation on words

Database: a **word** w where nodes have a color from an alphabet ⚪ 🔴 🔵



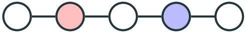
Query Q: a **sentence** (YES/NO question) in **monadic second-order logic** (MSO) *(to be defined)*

“Is there a blue node after each pink node?”

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

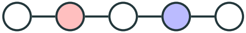
→ Note that we have restricted to **Boolean queries** for simplicity

Monadic second-order logic (MSO)



- $P_{\text{🔵}}(x)$ means “x is blue”; also $P_{\text{🔴}}(x)$, $P_{\text{○}}(x)$
- $x \to y$ means “x is the predecessor of y”

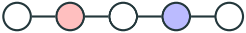
Monadic second-order logic (MSO)



- $P_{\textcolor{blue}{\bullet}}(x)$ means "*x* is blue"; also $P_{\textcolor{pink}{\bullet}}(x)$, $P_{\bigcirc}(x)$
- $x \to y$ means "*x* is the predecessor of *y*"

- Propositional logic: formulas with AND $\wedge$, OR $\vee$, NOT $\neg$
  - $P_{\textcolor{pink}{\bullet}}(x) \wedge P_{\textcolor{blue}{\bullet}}(y)$ means *"Node* ***x*** *is pink and node* ***y*** *is blue"*

Monadic second-order logic (MSO)

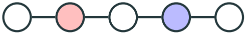


- $P_{🔵}(x)$ means "x is blue"; also $P_{🔴}(x)$, $P_{○}(x)$
- $x \rightarrow y$ means "x is the predecessor of y"

- **Propositional logic:** formulas with **AND** $\wedge$, **OR** $\vee$, **NOT** $\neg$
  - $P_{🔴}(x) \wedge P_{🔵}(y)$ means *"Node x is pink and node y is blue"*

- **First-order logic:** adds **existential quantifier** $\exists$ and **universal quantifier** $\forall$
  - $\exists x\, y\; P_{🔴}(x) \wedge P_{🔵}(y)$ means *"There is both a pink and a blue node"*

Monadic second-order logic (MSO)



- $P_{🔵}(x)$ means "x is blue"; also $P_{🔴}(x)$, $P_{⚪}(x)$
- $x \to y$ means "x is the predecessor of y"

- **Propositional logic:** formulas with **AND** $\land$, **OR** $\lor$, **NOT** $\neg$
  - $P_{🔴}(x) \land P_{🔵}(y)$ means *"Node x is pink and node y is blue"*

- **First-order logic:** adds **existential quantifier** $\exists$ and **universal quantifier** $\forall$
  - $\exists x\, y\; P_{🔴}(x) \land P_{🔵}(y)$ means *"There is both a pink and a blue node"*

- **Monadic second-order logic (MSO):** adds **quantifiers over sets**
  - $\exists S\; \forall x\; S(x)$ means *"there is a set S containing every element x"*
  - Can express **transitive closure** $x \to^* y$, i.e., *"x is before y"*
  - $\forall x\, P_{🔴}(x) \Rightarrow \exists y\, P_{🔵}(y) \land x \to^* y$
    means *"There is a blue node after each pink node"*

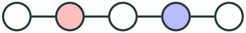
Word automata

Translate the query Q to a deterministic word automaton

Alphabet: ○ 🔴 🔵 w: ○—🔴—○—🔵—○ Q: $\exists x\, y\; P_{🔴}(x) \wedge P_{🔵}(y)$

Word automata

Translate the query Q to a deterministic word automaton

Alphabet: ○ 🔴 🔵 *w*:  *Q*: $\exists x\, y\ P_{🔴}(x) \wedge P_{🔵}(y)$

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

Word automata

Translate the query Q to a deterministic word automaton

Alphabet: ○ 🔴 🔵 *w*: ○—🔴—○—🔵—○ *Q*: $\exists x\, y\, P_{\text{🔴}}(x) \wedge P_{\text{🔵}}(y)$

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

Word automata

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

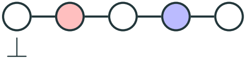
Alphabet: ○ 🔴 🔵   *w*: ○—🔴—○—🔵—○   *Q*: ∃*x* *y* *P*🔴(*x*) ∧ *P*🔵(*y*)

- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** ○ $\bot$ 🔴 P 🔵 B

Word automata

Translate the query Q to a deterministic word automaton

Alphabet: ○ 🔴 🔵   *w*:



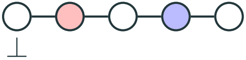
Q: $\exists x\, y\; P_{🔴}(x) \wedge P_{🔵}(y)$

- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: ○ $\bot$ 🔴 P 🔵 B

Word automata

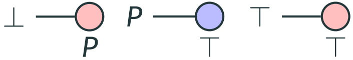
Translate the query Q to a deterministic word automaton

Alphabet: ○ 🔴 🔵 *w*:



Q: $\exists x\, y\; P$🔴$(x) \wedge P$🔵(y)

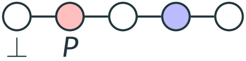
- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: ○ $\bot$ 🔴 P 🔵 B
- Transitions (examples):



Word automata

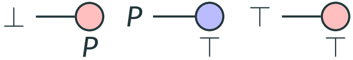
Translate the query *Q* to a **deterministic word automaton**

Alphabet: ○ 🔴 🔵   ***w***:



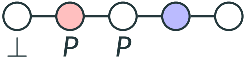
Q: ∃*x* *y* *P*🔴(*x*) ∧ *P*🔵(*y*)

- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** ○ ⊥   🔴 *P*   🔵 *B*
- **Transitions** (examples):

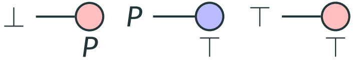


Word automata

Translate the query Q to a deterministic word automaton

Alphabet: ○ 🔴 🔵 *w*: 

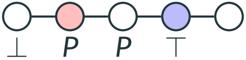
 Q: $\exists x\,y\; P_{\text{🔴}}(x) \wedge P_{\text{🔵}}(y)$

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

Word automata

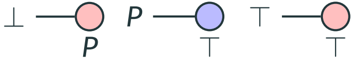
Translate the query Q to a deterministic word automaton

Alphabet: ○ 🔴 🔵 *w*:



Q: $\exists x\, y\ P_{🔴}(x) \wedge P_{🔵}(y)$

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

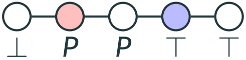


Word automata

Translate the query Q to a deterministic word automaton

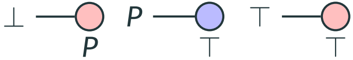
Alphabet: ○ 🔴 🔵

w:



Q: $\exists x\, y\, P_{🔴}(x) \wedge P_{🔵}(y)$

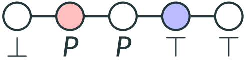
- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: ○ $\bot$ 🔴 P 🔵 B
- Transitions (examples):



Word automata

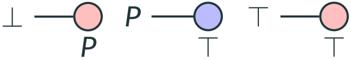
Translate the query Q to a deterministic word automaton

Alphabet: ○ 🔴 🔵 *w*:



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

- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: ○ $\bot$ 🔴 P 🔵 B
- Transitions (examples):

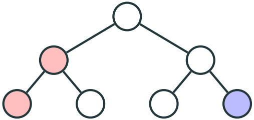


Theorem (Büchi, 1960)

MSO and word automata and regular expressions have the same expressive power on words

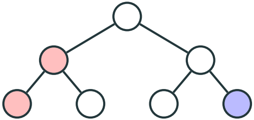
Query evaluation on trees

Database: a tree T where nodes have a color from an alphabet ○ 🔴 🔵



Query evaluation on trees

Database: a **tree** T where nodes have a color from an alphabet ○ 🔴 🔵



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

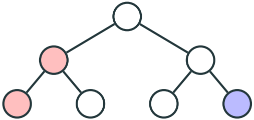
- $P_{\text{🔵}}(x)$ means "x is blue"
- $x \rightarrow y$ means "x is the parent of y"

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

$\exists x\, y\; P_{\text{🔴}}(x) \wedge P_{\text{🔵}}(y)$

Query evaluation on trees

Database: a **tree** T where nodes have a color from an alphabet ○ 🔴 🔵



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

- $P_{🔵}(x)$ means "x is blue"
- $x \rightarrow y$ means "x is the parent of y"

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

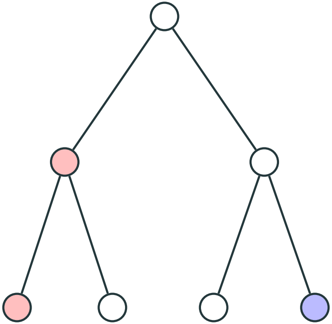
$\exists x\, y\; P_{🔴}(x) \wedge P_{🔵}(y)$

Result: YES/NO indicating if the tree T satisfies the query Q

Tree automata

Tree alphabet:

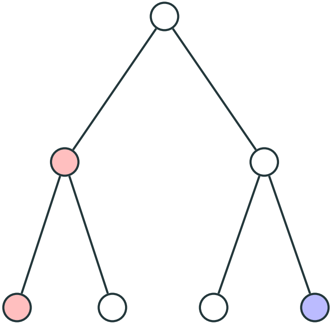
○ 🔴 🔵



Tree automata

Tree alphabet:

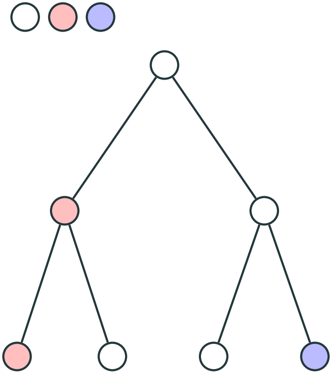
○ 🔴 🔵



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

Tree automata

Tree alphabet:

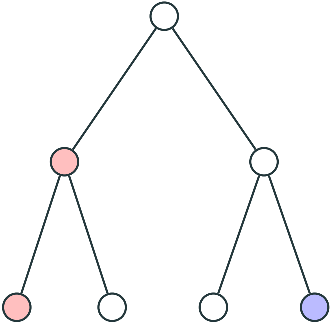


- Bottom-up deterministic **tree automaton**
- *“Is there both a pink and a blue node?”*
- States: $\{\bot, B, P, \top\}$

Tree automata

Tree alphabet:

○ 🔴 🔵

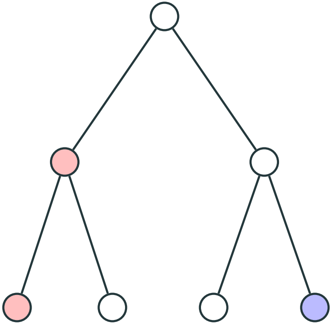


- Bottom-up deterministic tree automaton
- *"Is there both a pink and a blue node?"*
- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$

Tree automata

Tree alphabet:

○ 🔴 🔵

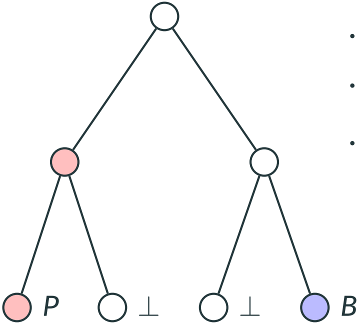


- Bottom-up deterministic **tree automaton**
- *“Is there both a pink and a blue node?”*
- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: ○ $\bot$ 🔴 P 🔵 B

Tree automata

Tree alphabet:

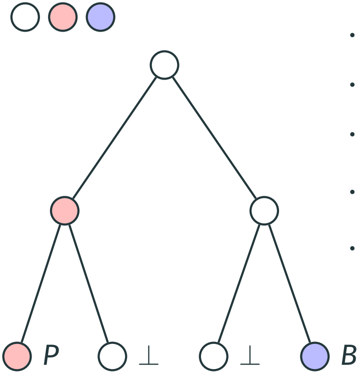
○ 🔴 🔵



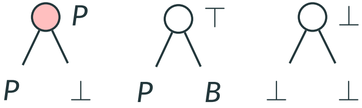
- Bottom-up deterministic **tree automaton**
- *“Is there both a pink and a blue node?”*
- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: ○ $\bot$ 🔴 P 🔵 B

Tree automata

Tree alphabet:



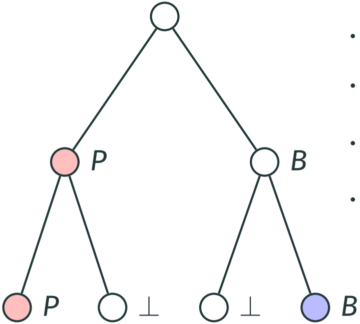
- Bottom-up deterministic **tree automaton**
- *“Is there both a pink and a blue node?”*
- **States:** $\{\perp, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** ○ $\perp$ 🔴 P 🔵 B
- **Transitions** (examples):



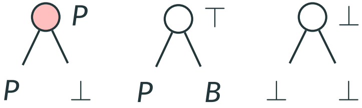
Tree automata

Tree alphabet:

○ 🔴 🔵



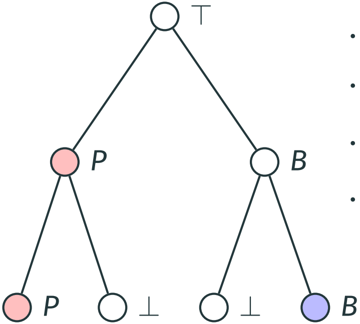
- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- States: $\{\perp, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: ○ $\perp$ 🔴 P 🔵 B
- Transitions (examples):



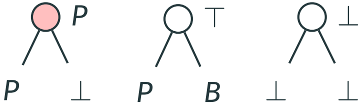
Tree automata

Tree alphabet:

○ 🔴 🔵



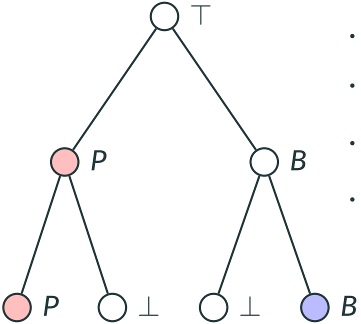
- Bottom-up deterministic tree automaton
- *“Is there both a pink and a blue node?”*
- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: ○ $\bot$ 🔴 P 🔵 B
- Transitions (examples):



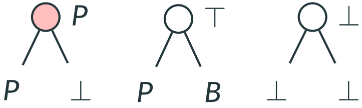
Tree automata

Tree alphabet:

○ 🔴 🔵



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



Theorem ([Thatcher and Wright 1968])

MSO and tree automata have the same expressive power on trees

Summary: Queries on Trees and Words

- We study data that has the shape of a **word** or **tree**
  - → e.g., sequences of events, XML documents, etc.
- Some queries **cannot be expressed** in relational algebra
  - → e.g., *"is there a blue node after each pink node?"*
- We restrict to **Boolean queries** (YES/NO question)
- The queries can be specified:
  - In a logical language (MSO)
  - On words, as a **regular expression**
  - → As an **automaton**

Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion

Motivation and definition

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

Motivation and definition

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

Remarks:

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

Motivation and definition

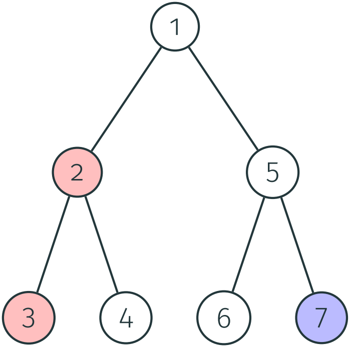
- Goal: notion of **provenance** for queries on trees/words expressed as **automata**
- We show how to **define** Boolean provenance in this context and how to **compute** it

Remarks:

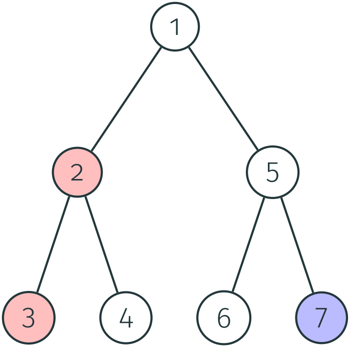
→ We work with **Boolean queries** (YES/NO) so the provenance will just describe when we get the answer YES

→ We restrict to **Boolean provenance** – but generalizations possible [Amarilli, Bourhis, and Senellart 2015a]

Defining provenance: Uncertain trees

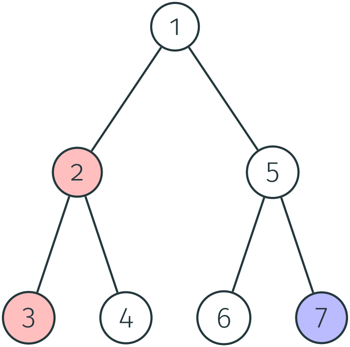


Defining provenance: Uncertain trees



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

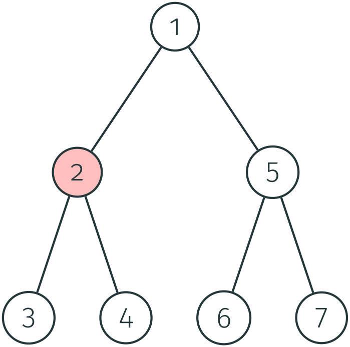
Defining provenance: Uncertain trees



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

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

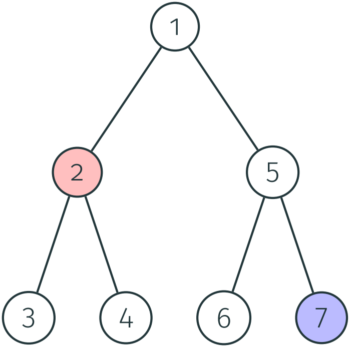
Defining provenance: Uncertain trees



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

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

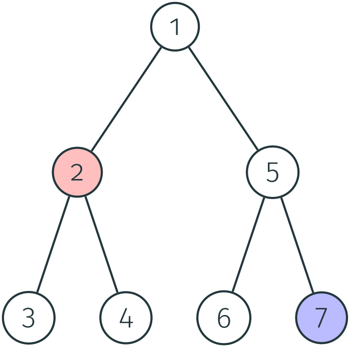
Defining provenance: Uncertain trees



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

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

Defining provenance: Uncertain trees

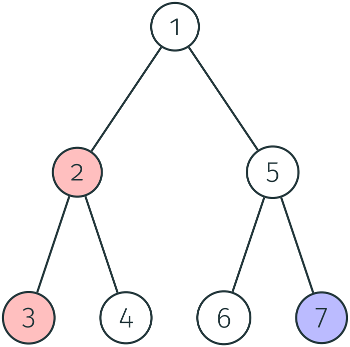


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

Defining provenance: Uncertain trees



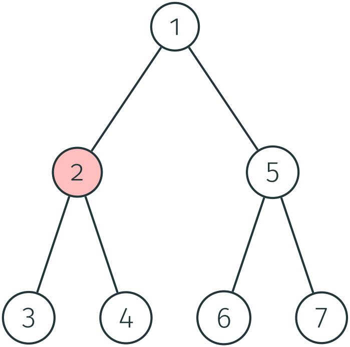
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

The query *Q* returns **YES**

Defining provenance: Uncertain trees



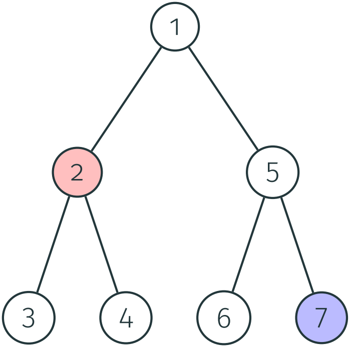
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

The query *Q* returns NO

Defining provenance: Uncertain trees



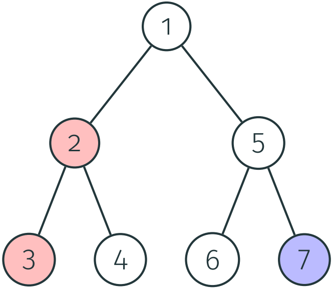
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

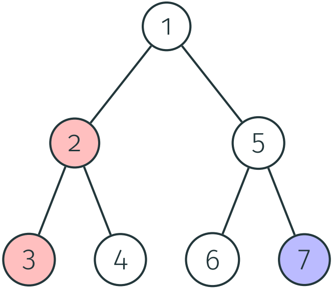
The query Q returns YES

Example: Provenance circuit



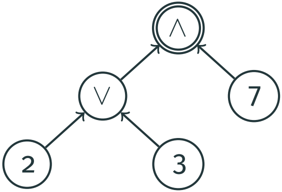
Query: *Is there both a pink and a blue node?*

Example: Provenance circuit

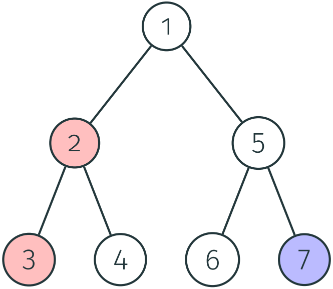


Query: *Is there both a pink and a blue node?*

Provenance circuit:

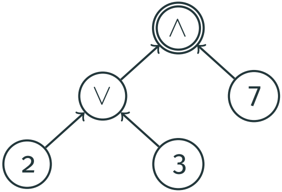


Example: Provenance circuit



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

Provenance circuit:



Formally:

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

Provenance circuits on trees [Amarilli, Bourhis, and Senellart 2015b]

Theorem

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

Provenance circuits on trees [Amarilli, Bourhis, and Senellart 2015b]

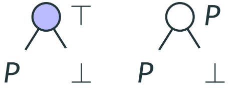
Theorem

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

- Alphabet: ○ ● ●
- Automaton: *"Is there both a pink and a blue node?"*

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

- Transitions:



Provenance circuits on trees [Amarilli, Bourhis, and Senellart 2015b]

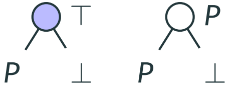
Theorem

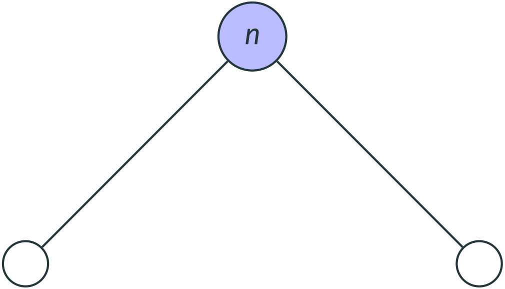
For any bottom-up tree automaton A and input tree T, we can build a Boolean provenance circuit of A on T in $O(|A| \times |T|)$

- Alphabet: ○ 🔴 🔵
- Automaton: *"Is there both a pink and a blue node?"*

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

- Transitions:





Provenance circuits on trees [Amarilli, Bourhis, and Senellart 2015b]

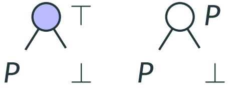
Theorem

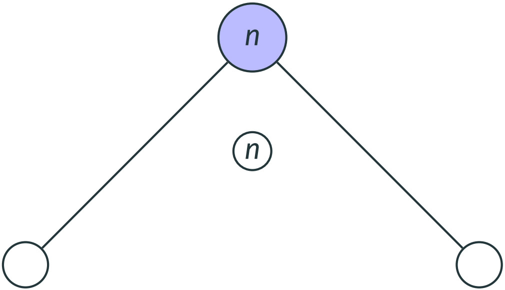
*For any bottom-up tree automaton A and input tree T,
we can build a Boolean provenance circuit of A on T in $O(|A| \times |T|)$*

- Alphabet: ○ 🔴 🔵
- Automaton: *"Is there both a pink and a blue node?"*

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

- Transitions:





Provenance circuits on trees [Amarilli, Bourhis, and Senellart 2015b]

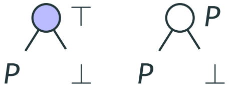
Theorem

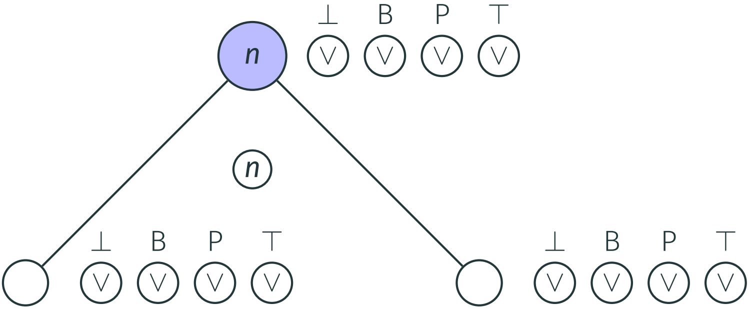
> *For any bottom-up tree automaton A and input tree T,*
> *we can build a Boolean provenance circuit of A on T in $O(|A| \times |T|)$*

- Alphabet: ⚪ 🔴 🔵
- Automaton: *"Is there both a pink and a blue node?"*

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

- Transitions:





Provenance circuits on trees [Amarilli, Bourhis, and Senellart 2015b]

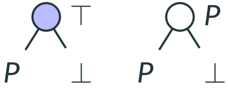
Theorem

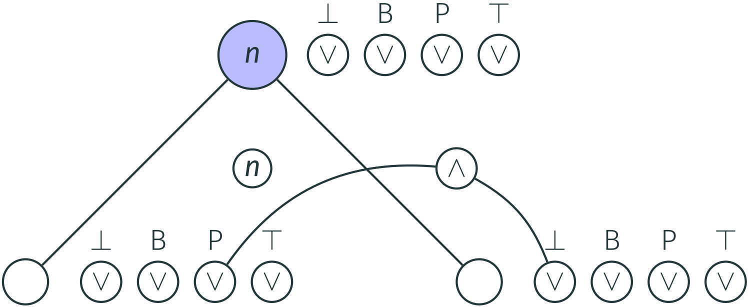
For any bottom-up tree automaton A and input tree T,
we can build a Boolean provenance circuit of A on T in $O(|A| \times |T|)$

- Alphabet: ○ 🔴 🔵
- Automaton: *“Is there both a pink and a blue node?”*

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

- Transitions:



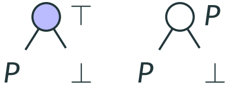


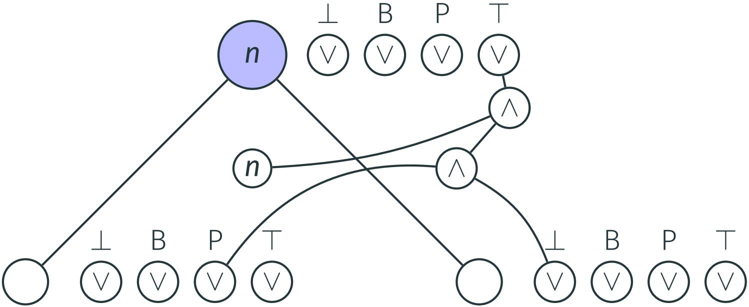
Provenance circuits on trees [Amarilli, Bourhis, and Senellart 2015b]

Theorem

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

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





Provenance circuits on trees [Amarilli, Bourhis, and Senellart 2015b]

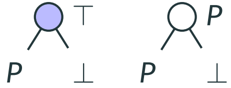
Theorem

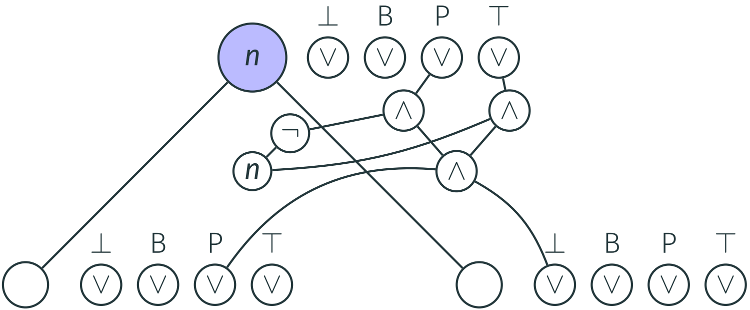
For any bottom-up ***tree automaton*** *A and input* ***tree*** *T,*
we can build a Boolean ***provenance circuit*** *of A on T in $O(|A| \times |T|)$*

- **Alphabet:** ○ 🔴 🔵
- **Automaton:** *“Is there both a pink and a blue node?”*

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

- **Transitions:**





Connections to knowledge compilation

The provenance circuits of automata on trees are...

Connections to knowledge compilation

The provenance circuits of automata on trees are...

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

Connections to knowledge compilation

The provenance circuits of automata on trees are...

- DNNF circuits:
  - → Negations only at the leaves
  - → Conjunctions are between disjoint subtrees
- Structured circuits
  - → The v-tree follows the shape of the input tree

Connections to knowledge compilation

The provenance circuits of automata on trees are...

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

Connections to knowledge compilation

The provenance circuits of automata on trees are...

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

Connections to knowledge compilation

The provenance circuits of automata on trees are...

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

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

Connections to knowledge compilation

The provenance circuits of automata on trees are…

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

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

→ Ongoing work: investigating these connections in more detail

Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion

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

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

name	position	city	classification	prob
John	Director	New York	unclassified	0.5
Paul	Janitor	New York	restricted	0.7
Dave	Analyst	Paris	confidential	0.3
Ellen	Field agent	Berlin	secret	0.2
Magdalen	Double agent	Paris	top secret	1.0
Nancy	HR director	Paris	restricted	0.8
Susan	Analyst	Berlin	secret	0.2

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

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

name	position	city	classification	prob
John	Director	New York	unclassified	0.5
Paul	Janitor	New York	restricted	0.7
Dave	Analyst	Paris	confidential	0.3
Ellen	Field agent	Berlin	secret	0.2
Magdalen	Double agent	Paris	top secret	1.0
Nancy	HR director	Paris	restricted	0.8
Susan	Analyst	Berlin	secret	0.2

→ Probability of a possible world $D' \subseteq D$:

$$\Pr(D') = \prod_{t\in D'} \Pr(t) \times \prod_{t\in D'\setminus D}(1 - \Pr(t'))$$

Query evaluation on probabilistic databases (PQE)

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

Query evaluation on probabilistic databases (PQE)

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

- Probability of a tuple for a query Q over D:

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

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

Query evaluation on probabilistic databases (PQE)

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

- Probability of a tuple for a query Q over D:

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

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

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

→ Idea: we can do this using Boolean provenance:
the probability of t is the probability of its annotation

Example of PQE

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

city	prov
New York	$x_1 \vee x_2$
Paris	$x_3 \vee x_5 \vee x_6$
Berlin	$x_4 \vee x_7$

Example of PQE

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

city	prov	prob
New York	$x_1 \lor x_2$	$1 - (1 - 0.5) \times (1 - 0.7) = 0.85$
Paris	$x_3 \lor x_5 \lor x_6$	1.00
Berlin	$x_4 \lor x_7$	$1 - (1 - 0.2) \times (1 - 0.2) = 0.36$

Complexity of PQE

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

Complexity of PQE

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

Complexity of PQE

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

→ Intensional vs extensional conjecture

More about the intensional vs extensional conjecture

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

More about the intensional vs extensional conjecture

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

- For OBDDs: there is a characterization of the queries with polynomial-sized OBDDs [Jha and Suciu 2013]

More about the intensional vs extensional conjecture

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

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

More about the intensional vs extensional conjecture

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

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

More about the intensional vs extensional conjecture

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

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

More about the intensional vs extensional conjecture

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

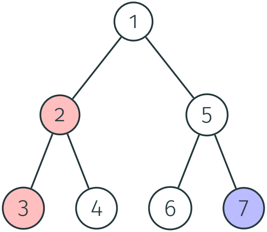
- For **OBDDs**: there is a characterization of the queries with polynomial-sized OBDDs [Jha and Suciu 2013]
- For **DLDDs** (e.g., dec-DNNFs), some safe queries have no tractable provenance representation in this class [Beame, Li, Roy, and Suciu 2017]
- For **d-SDNNF**, some safe queries have no tractable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

More about the intensional vs extensional conjecture

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

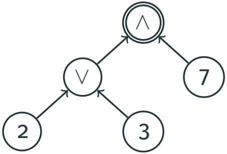
- For OBDDs: there is a characterization of the queries with polynomial-sized OBDDs [Jha and Suciu 2013]
- For DLDDs (e.g., dec-DNNFs), some safe queries have no tractable provenance representation in this class [Beame, Li, Roy, and Suciu 2017]
- For d-SDNNF, some safe queries have no tractable provenance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

Probabilistic query evaluation on trees

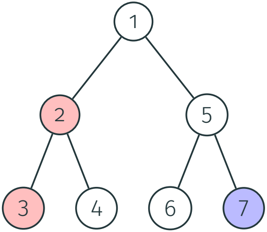


Query: *Is there both a pink and a blue node?*

Provenance circuit:

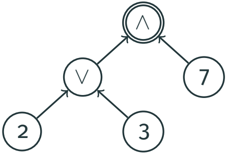


Probabilistic query evaluation on trees



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

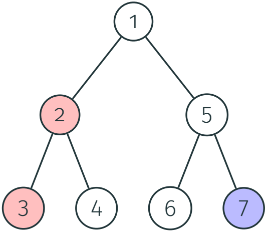
Provenance circuit:



- Consider a query Q on a **probabilistic tree** (each node has an independent probability of keeping its color)
- For queries given as **unambiguous tree automata**, we can construct a d-SDNNF **provenance circuit**

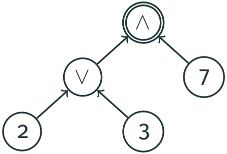
  → PQE is **tractable** for tree automata on trees

Probabilistic query evaluation on trees



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

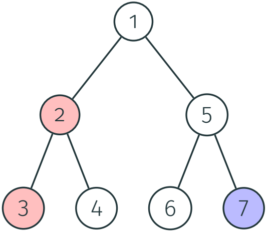
Provenance circuit:



- Consider a query Q on a probabilistic tree (each node has an independent probability of keeping its color)
- For queries given as unambiguous tree automata, we can construct a d-SDNNF provenance circuit
  - → PQE is tractable for tree automata on trees

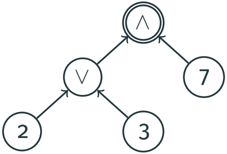
→ Extends to bounded treewidth databases – and essentially only to them [Amarilli, Bourhis, and Senellart 2016]

Probabilistic query evaluation on trees



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

Provenance circuit:



- Consider a query Q on a **probabilistic tree** (each node has an independent probability of keeping its color)
- For queries given as **unambiguous tree automata**, we can construct a d-SDNNF **provenance circuit**
  - → PQE is **tractable** for tree automata on trees

→ Extends to **bounded treewidth** databases – and essentially only to them [Amarilli, Bourhis, and Senellart 2016]

→ Relates to probability computation on bounded-treewidth **graphical models** [Amarilli, Capelli, Monet, and Senellart 2019]

Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion

Enumerating query results

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

Enumerating query results

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

how to find patterns

Enumerating query results

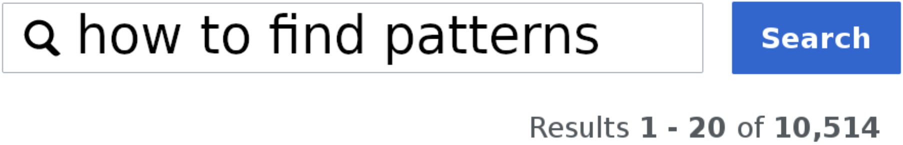
Idea: Often, we do not need to compute **all results** of a query
we just need to be able to **enumerate** results quickly

how to find patterns

Results **1 - 20** of **10,514**

Enumerating query results

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



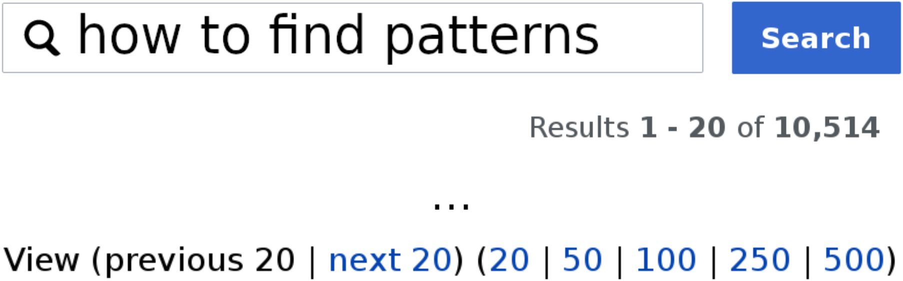
how to find patterns

Results **1 - 20** of **10,514**

...

Enumerating query results

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



how to find patterns

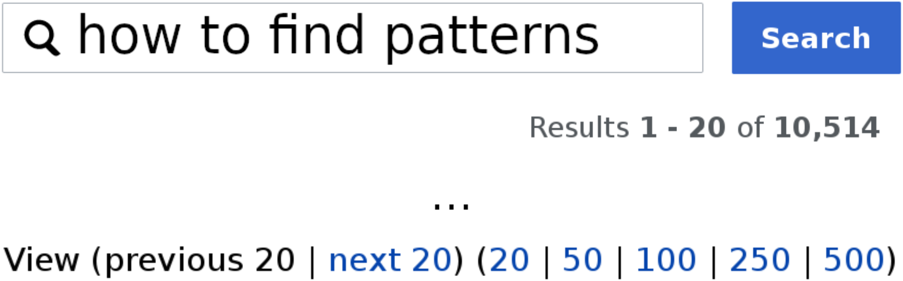
Results **1 - 20** of **10,514**

...

View (previous 20 | next 20) (20 | 50 | 100 | 250 | 500)

Enumerating query results

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



how to find patterns

Results **1 - 20** of **10,514**

...

View (previous 20 | next 20) (20 | 50 | 100 | 250 | 500)

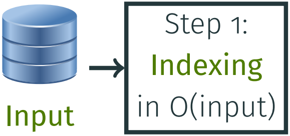
→ Formalization: **enumeration algorithms**

→ Currently a pretty important topic in database theory

Enumeration algorithm (linear preprocessing, constant delay)

Input

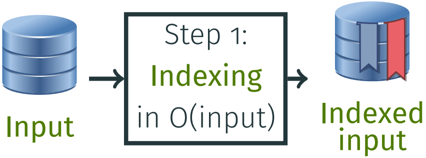
Enumeration algorithm (linear preprocessing, constant delay)



Input

Step 1:
Indexing
in O(input)

Enumeration algorithm (linear preprocessing, constant delay)



Input

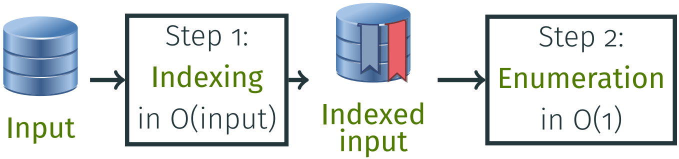
Step 1:

Indexing

in O(input)

Indexed input

Enumeration algorithm (linear preprocessing, constant delay)



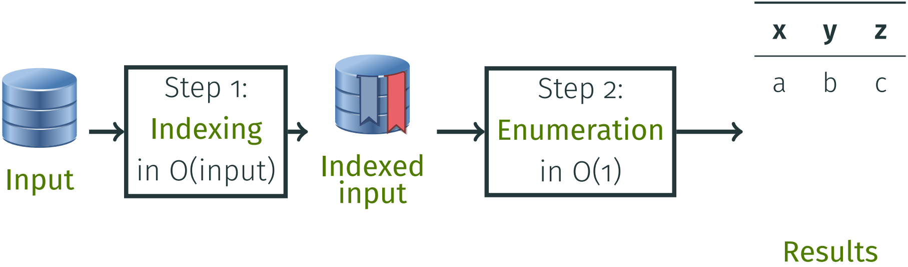
Input

Step 1: Indexing in O(input)

Indexed input

Step 2: Enumeration in O(1)

Enumeration algorithm (linear preprocessing, constant delay)



Input

Step 1:
Indexing
in O(input)

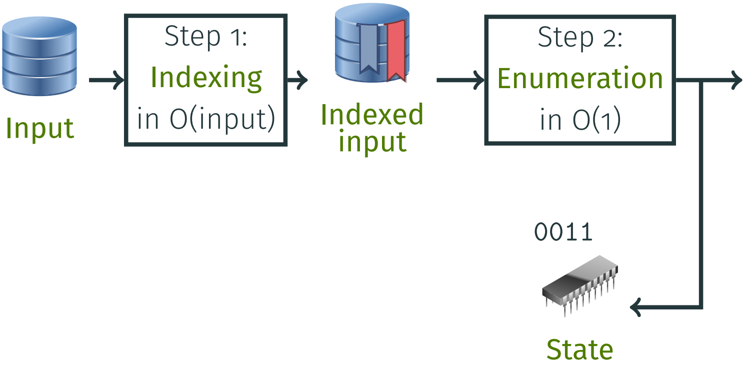
Indexed input

Step 2:
Enumeration
in O(1)

x	**y**	**z**
a	b	c

Results

Enumeration algorithm (linear preprocessing, constant delay)



Input

Step 1:
Indexing
in O(input)

Indexed input

Step 2:
Enumeration
in O(1)

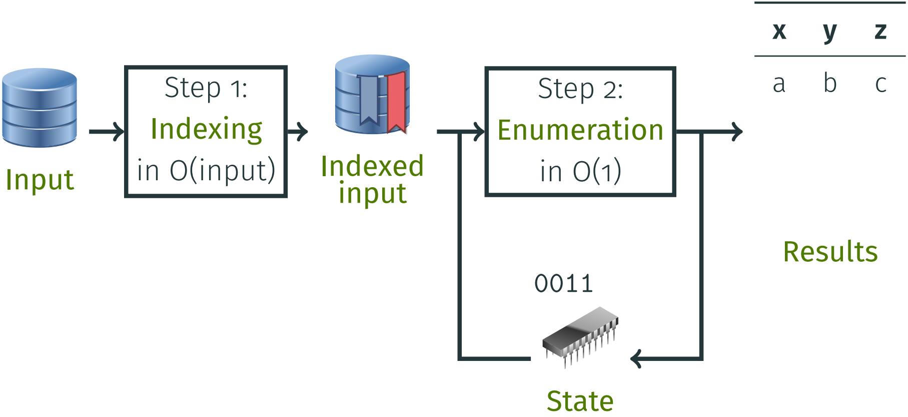
0011

State

x	y	z
a	b	c

Results

Enumeration algorithm (linear preprocessing, constant delay)



Input

Step 1: Indexing in O(input)

Indexed input

Step 2: Enumeration in O(1)

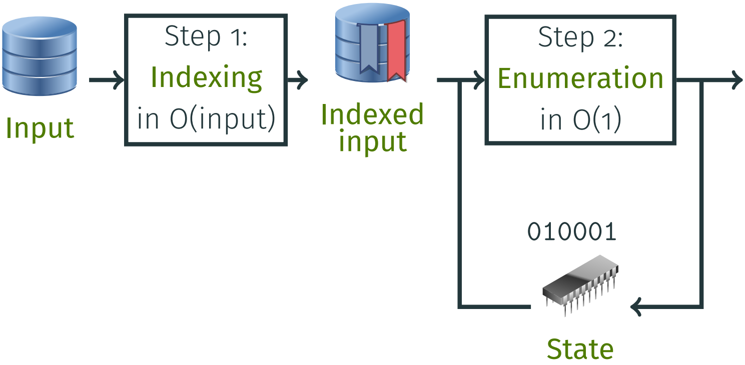
0011

State

x	**y**	**z**
a	b	c

Results

Enumeration algorithm (linear preprocessing, constant delay)



Input

Step 1:
Indexing
in O(input)

Indexed input

Step 2:
Enumeration
in O(1)

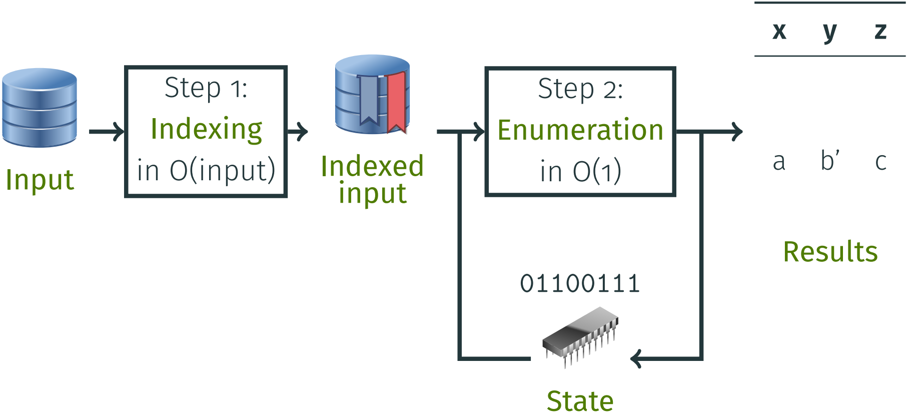
010001

State

x	**y**	**z**
a’	b	c

Results

Enumeration algorithm (linear preprocessing, constant delay)



Input

Step 1:
Indexing
in O(input)

Indexed input

Step 2:
Enumeration
in O(1)

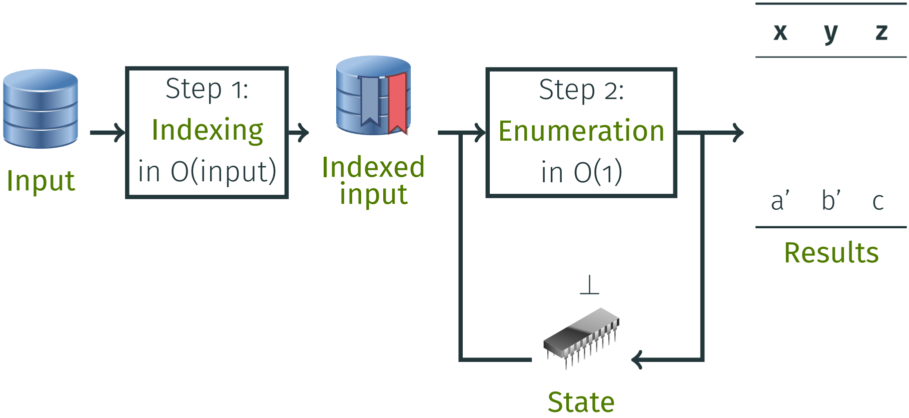
01100111

State

x	y	z
a	b’	c

Results

Enumeration algorithm (linear preprocessing, constant delay)



Input

Step 1:
Indexing
in O(input)

Indexed input

Step 2:
Enumeration
in O(1)

x	y	z
a’	b’	c

Results

$\perp$

State

Connection to provenance

Provenance can also represent query answers!

Connection to provenance

Provenance can also represent **query answers**!

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

Connection to provenance

Provenance can also represent query answers!

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

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

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

Connection to provenance

Provenance can also represent **query answers**!

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

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

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

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

Connection to provenance

Provenance can also represent query answers!

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

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

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

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

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

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

Connection to provenance

Provenance can also represent **query answers**!

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

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

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

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

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

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

- Consider the **Boolean query** $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$

Connection to provenance

Provenance can also represent **query answers**!

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

  $$Q(x, y) : \exists z\, R(x, y) \wedge S(y, z)$$
  $$D : R(a, b), R(a', b), S(b, c)$$

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

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

- Consider the **Boolean query** $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$

  $$X(x) \wedge Y(y) \wedge (\exists z\, R(x, y) \wedge S(y, z))$$

Connection to provenance

Provenance can also represent query answers!

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

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

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

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

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

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

- Consider the Boolean query $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$

  $X(x) \wedge Y(y) \wedge (\exists z\, R(x, y) \wedge S(y, z))$

- Compute the provenance C' of Q' on D plus assignment facts

Connection to provenance

Provenance can also represent **query answers**!

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

  $$Q(x, y) : \exists z\, R(x, y) \land S(y, z)$$
  $$D : R(a, b), R(a', b), S(b, c)$$

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

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

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

  $$X(x) \land Y(y) \land (\exists z\, R(x, y) \land S(y, z))$$

- Compute the **provenance** C' of Q' on D plus assignment facts

  $$(X(a) \land R(a, b) \lor X(a') \land R(a', b))$$
  $$\land Y(b) \land S(b, c)$$

Connection to provenance

Provenance can also represent **query answers**!

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

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

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

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

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

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

- Consider the **Boolean query** $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$

  $X(x) \wedge Y(y) \wedge (\exists z\ R(x, y) \wedge S(y, z))$

- Compute the **provenance** C' of Q' on D plus assignment facts

  $$(X(a) \wedge R(a, b) \vee X(a') \wedge R(a', b)) \wedge Y(b) \wedge S(b, c)$$

- Define C by replacing all variables by $\mathbf{1}$ except assignment facts

Connection to provenance

Provenance can also represent **query answers**!

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

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

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

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

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

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

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

  $X(x) \land Y(y) \land (\exists z\, R(x, y) \land S(y, z))$

- Compute the **provenance** C' of Q' on D plus assignment facts

  $(X(a) \land R(a, b) \lor X(a') \land R(a', b))$

  $\land Y(b) \land S(b, c)$

- Define C by replacing all variables by **1** except assignment facts

  $(X(a) \lor X(a')) \land Y(b)$

Connection to provenance

Provenance can also represent **query answers**!

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

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

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

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

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

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

- Consider the **Boolean query** $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$

  $X(x) \wedge Y(y) \wedge (\exists z\, R(x, y) \wedge S(y, z))$

- Compute the **provenance** C' of Q' on D plus assignment facts

  $(X(a) \wedge R(a, b) \vee X(a') \wedge R(a', b))$

  $\wedge Y(b) \wedge S(b, c)$

- Define C by replacing all variables by **1** except assignment facts

  $(X(a) \vee X(a')) \wedge Y(b)$

→ The circuit C represents the **query answers**

Connection to provenance

Provenance can also represent **query answers**!

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

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

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

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

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

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

- Consider the **Boolean query** $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$

  $X(x) \wedge Y(y) \wedge (\exists z \; R(x, y) \wedge S(y, z))$

- Compute the **provenance** C' of Q' on D plus assignment facts

  $(X(a) \wedge R(a, b) \vee X(a') \wedge R(a', b))$

  $\wedge Y(b) \wedge S(b, c)$

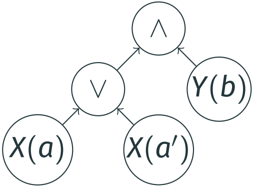
- Define C by replacing all variables by $\mathbf{1}$ except assignment facts

  $(X(a) \vee X(a')) \wedge Y(b)$

$\rightarrow$ The circuit C represents the **query answers** (a, b) and (a', b)

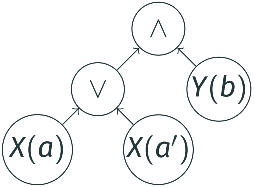
Enumeration via provenance and knowledge compilation

- We have a provenance circuit representing the query answers



Enumeration via provenance and knowledge compilation

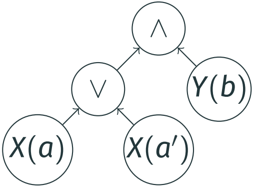
- We have a provenance circuit representing the query answers



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

Enumeration via provenance and knowledge compilation

- We have a provenance circuit representing the query answers



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

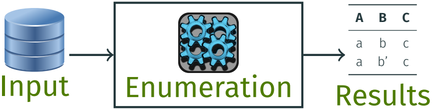
→ We want linear preprocessing and constant delay
so we had to do our own enumeration algorithm for circuits:

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

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

Enumeration via knowledge compilation

Currently:



Input

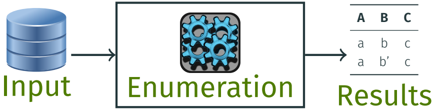
Enumeration

A	B	C
a	b	c
a	b’	c

Results

Enumeration via knowledge compilation

Currently:

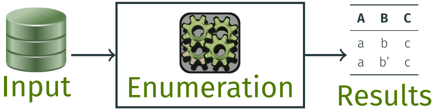


Input

Enumeration

A	B	C
a	b	c
a	b’	c

Results



Input

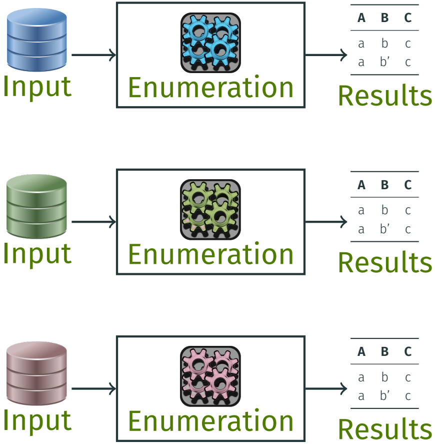
Enumeration

A	B	C
a	b	c
a	b’	c

Results

Enumeration via knowledge compilation

Currently:



Input

Enumeration

A	B	C
a	b	c
a	b’	c

Results

Input

Enumeration

A	B	C
a	b	c
a	b’	c

Results

Input

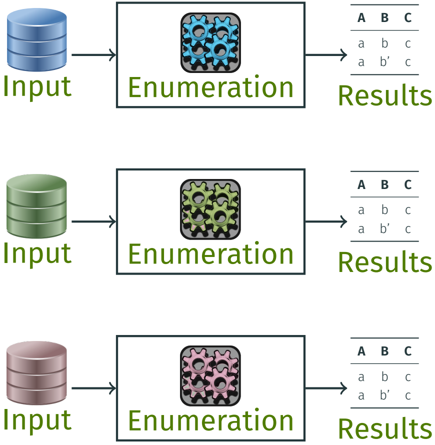
Enumeration

A	B	C
a	b	c
a	b’	c

Results

Enumeration via knowledge compilation

Currently:



Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

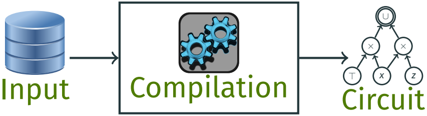
Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

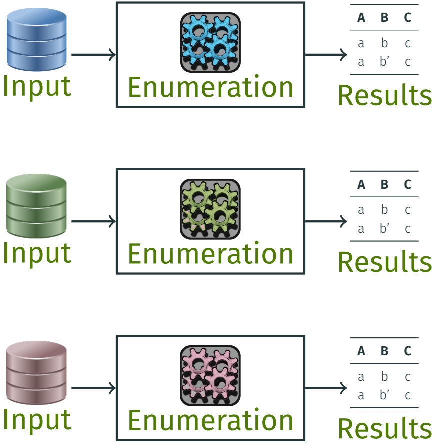
Our idea:



Input → Compilation → Circuit

Enumeration via knowledge compilation

Currently:



Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

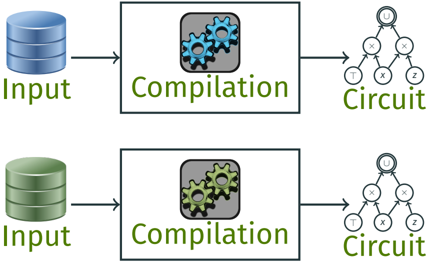
Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

Our idea:

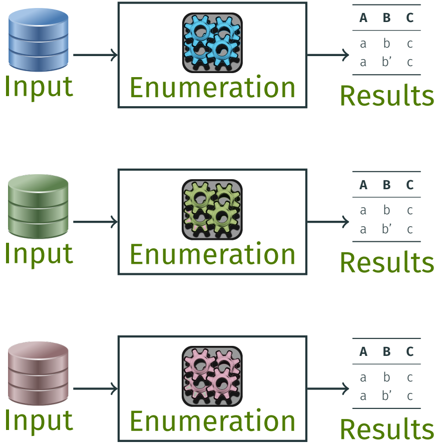


Input → Compilation → Circuit

Input → Compilation → Circuit

Enumeration via knowledge compilation

Currently:



Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

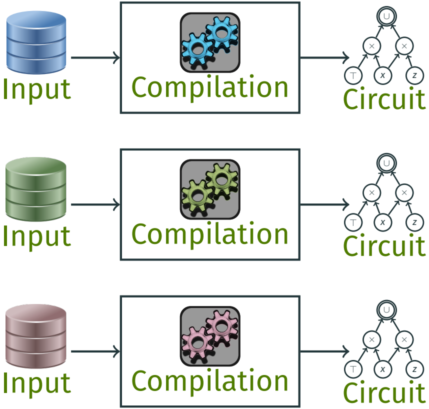
Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

Our idea:



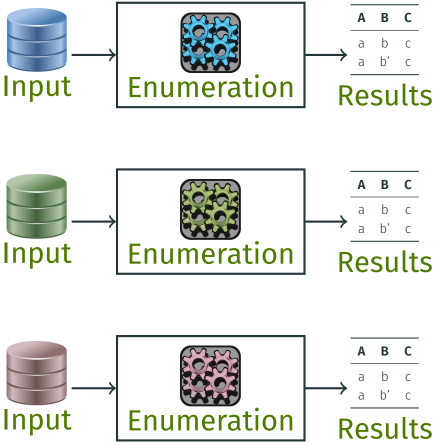
Input → Compilation → Circuit

Input → Compilation → Circuit

Input → Compilation → Circuit

Enumeration via knowledge compilation

Currently:



Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

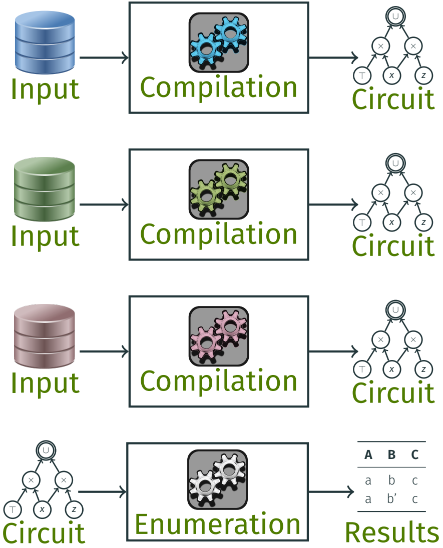
Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

Input → Enumeration → Results

A	B	C
a	b	c
a	b’	c

Our idea:



Input → Compilation → Circuit

Input → Compilation → Circuit

Input → Compilation → Circuit

Circuit → Enumeration → Results

A	B	C
a	b	c
a	b’	c

Results and extensions

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

Results and extensions

- Remember that, for tree automata on trees, we can build d-SDNNF provenance representations in linear time
- With our enumeration result, this shows that we can enumerate query results with linear preprocessing and constant-delay
  - → Was already known in database theory [Bagan 2006; Kazana and Segoufin 2013]

Results and extensions

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

Results and extensions

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

Ongoing work: provenance-based enumeration for relational algebra

Outline

Query Evaluation on Relational Databases

Boolean Provenance on Relational Databases

Semiring Provenance on Relational Databases

Query Evaluation on Trees and Words

Boolean Provenance on Trees and Words

Applications to Probability Computation

Applications to Enumeration

Conclusion

Provenance in practice

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

Provenance in practice

- How can we compute provenance in practice?
    - ProvSQL module for PostgreSQL, by Pierre Senellart et al.
    - Keeps track of provenance as a circuit
    - `https://github.com/PierreSenellart/provsql`
- How can we do probabilistic query evaluation via provenance?
    - ProvSQL is interfaced with c2d, d4, and dsharp

Provenance in practice

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

Provenance in practice

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

Provenance in theory

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

Provenance in theory

- Confession: as a theoretical topic, provenance feels definitional
  - → Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...
- Which directions are less definitional?
  - Using provenance for computational tasks

Provenance in theory

- Confession: as a theoretical topic, provenance feels definitional

  → Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile…

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

Provenance in theory

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

Provenance in theory

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

Provenance in theory

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

Thanks for your attention!

Bibliography i

Amarilli, Antoine, Pierre Bourhis, Louis Jachiet, and Stefan Mengel (2017). “A Circuit-Based Approach to Efficient Enumeration”. In: *ICALP*.

Amarilli, Antoine, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth (2019). “Enumeration on Trees with Tractable Combined Complexity and Efficient Updates”. In: *PODS*.

Amarilli, Antoine, Pierre Bourhis, Mikaël Monet, and Pierre Senellart (2017). “Combined Tractability of Query Evaluation via Tree Automata and Cycluits”. In: *ICDT*.

Amarilli, Antoine, Pierre Bourhis, and Pierre Senellart (July 2015a). “Provenance Circuits for Trees and Treelike Instances”. In: *Proc. ICALP*. Kyoto, Japan, pp. 56–68.

Bibliography ii

Amarilli, Antoine, Pierre Bourhis, and Pierre Senellart (Nov. 2015b). *Provenance Circuits for Trees and Treelike Instances (Extended Version)*. CoRR abs/1511.08723.

– (June 2016). “Tractable Lineages on Treelike Instances: Limits and Extensions”. In: *Proc. PODS*. San Francisco, USA, pp. 355–370.

Amarilli, Antoine, Florent Capelli, Mikaël Monet, and Pierre Senellart (2019). “Connecting Knowledge Compilation Classes and Width Parameters”. In: *ToCS* 2019.

Amarilli, Antoine and Mikaël Monet (2016). *Example of a naturally ordered semiring which is not an m-semiring*. `http://math.stackexchange.com/questions/1966858`.

Amer, K. (1984). “Equationally complete classes of commutative monoids with monus”. In: *Algebra Universalis* 18.1.

Bibliography iii

Amsterdamer, Yael, Daniel Deutch, and Val Tannen (2011a). “On the Limitations of Provenance for Queries with Difference.”. In: *TaPP*.

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

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

Beame, Paul, Jerry Li, Sudeepa Roy, and Dan Suciu (2017). “Exact model counting of query expressions: Limitations of propositional methods”. In: *TODS* 42.1, p. 1.

Bova, Simone and Stefan Szeider (2017). “Circuit treewidth, sentential decision, and query compilation”. In: *PODS*. ACM, pp. 233–246.

Bibliography iv

Buneman, Peter, Sanjeev Khanna, and Wang Chiew Tan (2001). “Why and Where: A Characterization of Data Provenance”. In: *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*

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

Chapman, Adriane and H. V. Jagadish (2009). “Why not?” In: *SIGMOD.*

Dalvi, Nilesh and Dan Suciu (2007). “Efficient Query Evaluation on Probabilistic Databases”. In: *VLDBJ* 16.4.

– (2012). “The dichotomy of probabilistic inference for unions of conjunctive queries”. In: *J. ACM* 59.6.

Deutch, Daniel, Tova Milo, Sudeepa Roy, and Val Tannen (2014). “Circuits for Datalog Provenance.”. In: *ICDT.*

Bibliography v

Fink, Robert, Larisa Han, and Dan Olteanu (2012). “Aggregation in probabilistic databases via knowledge compilation”. In: *Proceedings of the VLDB Endowment* 5.5, pp. 490–501.

Geerts, Floris and Antonella Poggi (2010). “On database query languages for K-relations”. In: *J. Applied Logic* 8.2.

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

Green, Todd and Val Tannen (2006). “Models for Incomplete and Probabilistic Information”. In: *IEEE Data Eng. Bull.* 29.1.

Jha, Abhay Kumar and Dan Suciu (2013). “Knowledge Compilation Meets Database Theory: Compiling Queries to Decision Diagrams”. In: *TCS* 52.3. `https://homes.cs.washington.edu/~suciu/camera_ready.pdf`.

Bibliography vi

Kazana, Wojciech and Luc Segoufin (2013). “Enumeration of monadic second-order queries on trees”. In: *TOCL* 14.4.

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

Olteanu, Dan and Jiewen Huang (2008). “Using OBDDs for Efficient Query Evaluation on Probabilistic Databases”. In: *Proc. SUM*.

Suciu, Dan, Dan Olteanu, Christopher Ré, and Christoph Koch (2011). *Probabilistic Databases*. Morgan & Claypool.

Thatcher, James W. and Jesse B. Wright (1968). “Generalized finite automata theory with an application to a decision problem of second-order logic”. In: *Mathematical systems theory* 2.1, pp. 57–81.