Provenance

MPRI 2.26.2: Web Data Management

Antoine Amarilli, Pierre Senellart

Friday, January 18th

Provenance definition

Provenance management

- Data management is all about query evaluation
- What if we want something more than the query 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 part of the input data was missing?
  - What is the minimal security clearance I need to see the result?
  - What is the most economical way of obtaining the result?
  - How can a result be explained to the user?
- Provenance management: along with query evaluation, record additional bookkeeping information allowing to answer the questions above

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 first as a tuple id)

name	position	city	classification	prov
John	Director	New York	unclassified	t_1
Paul	Janitor	New York	restricted	t_2
Dave	Analyst	Paris	confidential	t_3
Ellen	Field agent	Berlin	secret	t_4
Magdalen	Double agent	Paris	top secret	t_5
Nancy	HR director	Paris	restricted	t_6
Susan	Analyst	Berlin	secret	t_7

Relations and databases

Formally:

- A relational schema $\mathcal{R}$ is a finite sequence of distinct attribute names; the arity of $\mathcal{R}$ is $|\mathcal{R}|$
- A database schema is a finite set of relation names, each having a relational schema
- A tuple over relational schema $\mathcal{R}$ is a mapping from $\mathcal{R}$ to data values; each tuple comes with a provenance annotation
- A relation instance (or relation) over $\mathcal{R}$ is a finite set of tuples over $\mathcal{R}$
- A database instance (or database) over database schema $\mathcal{D}$ is a mapping from the support of $\mathcal{D}$ mapping each relation name R to a relation instance over $\mathcal{D}(R)$

Queries

- A query is an arbitrary function that maps databases over a fixed database schema $\mathcal{D}$ to relations over some relational schema $\mathcal{R}$
- The query does not look at the provenance annotations; we will give semantics for the provenance annotations of the output, based on that of the input
- Example of query languages:
    - First-Order logic (FO) or the relational algebra
    - Monadic-Second Order logic (MSO)
    - SQL with aggregate functions
    - etc.

Outline

Boolean provenance [Imieliński and Lipski, 1984]

- $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ finite set of Boolean events
- Provenance annotation: Boolean function over $\mathcal{X}$, i.e., a function of the form: $(\mathcal{X} \to \{\bot, \top\}) \to \{\bot, \top\}$
- Interpretation: possible-world semantics
  - every valuation $\nu : \mathcal{X} \to \{\bot, \top\}$ denotes a possible world of the database
  - the provenance of a tuple on ν evaluates to $\bot$ or $\top$ depending whether this tuple exists in that possible world
  - for example, if every tuple of a database is annotated with a different Boolean event, the set of possible worlds is the set of all subdatabases
- This is very similar to c-tables seen in the previous class

Example of possible worlds

name	position	city	classification	prov
John	Director	New York	unclassified	t_1
Paul	Janitor	New York	restricted	t_2
Dave	Analyst	Paris	confidential	t_3
Ellen	Field agent	Berlin	secret	t_4
Magdalen	Double agent	Paris	top secret	t_5
Nancy	HR director	Paris	restricted	t_6
Susan	Analyst	Berlin	secret	t_7

$$\nu: \quad \begin{array}{ccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \\ \top & \top & \top & \top & \top & \top & \top \end{array}$$

Example of possible worlds

name	position	city	classification	prov
John	Director	New York	unclassified	t_1
Dave	Analyst	Paris	confidential	t_3
Magdalen	Double agent	Paris	top secret	t_5
Susan	Analyst	Berlin	secret	t_7

$$\nu: \begin{array}{ccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \\ \top & \bot & \top & \bot & \top & \bot & \top \end{array}$$

Boolean provenance of query results

- $\nu(D)$: the subdatabase of D where all tuples whose provenance annotation evaluates to $\bot$ by ν is removed
- The Boolean provenance $\mathrm{prov}_{q,D}(t)$ of tuple $t \in q(D)$ is the function:

$$\nu \mapsto \begin{cases} \top \text{ if } t \in q(\nu(D)) \\ \bot \text{ otherwise} \end{cases}$$

Example (What cities are in the table?)

name	position	city	classification	prov
John	Director	New York	unclassified	t_1
Paul	Janitor	New York	restricted	t_2
Dave	Analyst	Paris	confidential	t_3
Ellen	Field agent	Berlin	secret	t_4
Magdalen	Double agent	Paris	top secret	t_5
Nancy	HR director	Paris	restricted	t_6
Susan	Analyst	Berlin	secret	t_7

city	prov
Berlin	$t_4 \vee t_7$
New York	$t_1 \vee t_2$
Paris	$t_3 \vee t_5 \vee t_6$

Application: Probabilistic databases [Green and Tannen, 2006, Suciu et al., 2011]

- Tuple-independent database: each tuple t in a database is annotated with independent probability $\Pr(t)$ of existing
- Probability of a possible world $D' \subseteq D$:

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

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

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

Application: Probabilistic databases [Green and Tannen, 2006, Suciu et al., 2011]

- Tuple-independent database: each tuple t in a database is annotated with independent probability $\Pr(t)$ of existing
- Probability of a possible world $D' \subseteq D$:

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

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

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

- If $\Pr(x_i) \coloneqq \Pr(t_i)$ where x_i is the provenance annotation of tuple t_i then $\Pr(t \in q(D)) = \Pr(\mathrm{prov}_{q,D}(t))$
- Computing the probability of an answer tuple databases thus amounts to computing its Boolean provenance, and then computing the probability of that Boolean function
- Also works for more complex probabilistic models

Example of probability computation

name	position	city	classification	prov	prob
John	Director	New York	unclassified	t_1	0.5
Paul	Janitor	New York	restricted	t_2	0.7
Dave	Analyst	Paris	confidential	t_3	0.3
Ellen	Field agent	Berlin	secret	t_4	0.2
Magdalen	Double agent	Paris	top secret	t_5	1.0
Nancy	HR director	Paris	restricted	t_6	0.8
Susan	Analyst	Berlin	secret	t_7	0.2

city	prov
Berlin	$t_4 \vee t_7$
New York	$t_1 \vee t_2$
Paris	$t_3 \vee t_5 \vee t_6$

Example of probability computation

name	position	city	classification	prov	prob
John	Director	New York	unclassified	t_1	0.5
Paul	Janitor	New York	restricted	t_2	0.7
Dave	Analyst	Paris	confidential	t_3	0.3
Ellen	Field agent	Berlin	secret	t_4	0.2
Magdalen	Double agent	Paris	top secret	t_5	1.0
Nancy	HR director	Paris	restricted	t_6	0.8
Susan	Analyst	Berlin	secret	t_7	0.2

city	prov	prob
Berlin	$t_4 \vee t_7$	$1-(1-0.2)\times(1-0.2)=0.36$
New York	$t_1 \vee t_2$	$1-(1-0.5)\times(1-0.7)=0.85$
Paris	$t_3 \vee t_5 \vee t_6$	1.00

What now?

- How to compute Boolean provenance for practical query languages? What complexity?
- Can we do more with provenance?
- How should we represent provenance annotations?
- How can we implement support for provenance management in a relational database management system?

Outline

Relational algebra

- Basic relations:
  - the relation names in the signature
  - constant relations, e.g., the empty relation
- Projection Π
- Selection σ
- Renaming ρ
- Union $\cup$
- Product $\times$ and join $\bowtie$
- Difference $-$

Projection: keep a subsequence of the attributes

Class

date	teacher	resp	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4
2019-01-18	Pierre	Olivier	Web Data Mgmt	5
2019-02-01	Pierre	Olivier	Web Data Mgmt	6
2019-02-08	Pierre	Olivier	Web Data Mgmt	7

Projection: keep a subsequence of the attributes

Class

date	teacher	resp	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4
2019-01-18	Pierre	Olivier	Web Data Mgmt	5
2019-02-01	Pierre	Olivier	Web Data Mgmt	6
2019-02-08	Pierre	Olivier	Web Data Mgmt	7

$\Pi_{\mathbf{teacher},\mathbf{resp}}(\text{Class})$

teacher	resp

Projection: keep a subsequence of the attributes

Class

date	teacher	resp	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4
2019-01-18	Pierre	Olivier	Web Data Mgmt	5
2019-02-01	Pierre	Olivier	Web Data Mgmt	6
2019-02-08	Pierre	Olivier	Web Data Mgmt	7

$\Pi_{\mathbf{teacher},\mathbf{resp}}(\text{Class})$

teacher	resp
Antoine	Olivier
Pierre	Olivier

Projection: keep a subsequence of the attributes

Class

date	teacher	resp	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4
2019-01-18	Pierre	Olivier	Web Data Mgmt	5
2019-02-01	Pierre	Olivier	Web Data Mgmt	6
2019-02-08	Pierre	Olivier	Web Data Mgmt	7

$\Pi_{\textbf{teacher},\textbf{resp}}(\text{Class})$

teacher	resp
Antoine	Olivier
Pierre	Olivier

→ Duplicates are removed

Selection: keep a subset of the tuples

Class

date	teacher	resp	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4
2019-01-18	Pierre	Olivier	Web Data Mgmt	5
2019-02-01	Pierre	Olivier	Web Data Mgmt	6
2019-02-08	Pierre	Olivier	Web Data Mgmt	7

Selection: keep a subset of the tuples

Class

date	teacher	resp	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4
2019-01-18	Pierre	Olivier	Web Data Mgmt	5
2019-02-01	Pierre	Olivier	Web Data Mgmt	6
2019-02-08	Pierre	Olivier	Web Data Mgmt	7

$\sigma_{\textbf{teacher}=\text{“Antoine”}}(\text{Class})$

date	teacher	resp	name	num

Selection: keep a subset of the tuples

Class

date	teacher	resp	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4
2019-01-18	Pierre	Olivier	Web Data Mgmt	5
2019-02-01	Pierre	Olivier	Web Data Mgmt	6
2019-02-08	Pierre	Olivier	Web Data Mgmt	7

$\sigma_{\textbf{teacher}=\text{“Antoine”}}(\text{Class})$

date	teacher	resp	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4

Rename: change the name of attributes

Class

date	teacher	resp	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4
2019-01-18	Pierre	Olivier	Web Data Mgmt	5
2019-02-01	Pierre	Olivier	Web Data Mgmt	6
2019-02-08	Pierre	Olivier	Web Data Mgmt	7

Rename: change the name of attributes

Class

date	teacher	resp	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4
2019-01-18	Pierre	Olivier	Web Data Mgmt	5
2019-02-01	Pierre	Olivier	Web Data Mgmt	6
2019-02-08	Pierre	Olivier	Web Data Mgmt	7

$\rho_{\mathbf{resp} \to \mathbf{boss}}$(Class)

Rename: change the name of attributes

Class

date	teacher	resp	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4
2019-01-18	Pierre	Olivier	Web Data Mgmt	5
2019-02-01	Pierre	Olivier	Web Data Mgmt	6
2019-02-08	Pierre	Olivier	Web Data Mgmt	7

$\rho_{\mathbf{resp} \to \mathbf{boss}}$(Class)

date	teacher	boss	name	num
2019-01-11	Antoine	Olivier	Web Data Mgmt	4
2019-01-18	Pierre	Olivier	Web Data Mgmt	5
2019-02-01	Pierre	Olivier	Web Data Mgmt	6
2019-02-08	Pierre	Olivier	Web Data Mgmt	7

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$\cup$

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$\bigcup$

S2

id	name
42	Zaphod B.

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$\cup$

S2

id	name
42	Zaphod B.

$=$

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$\cup$

S2

id	name
42	Zaphod B.

$=$

Students1 $\cup$ S2

id	name
1	Arthur Dent
2	Ford Prefect
42	Zaphod B.

Union

- Take tuples occurring in one of the input tables
- Applies to two tables with the same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$\cup$

S2

id	name
42	Zaphod B.

$=$

Students1 ∪ S2

id	name
1	Arthur Dent
2	Ford Prefect
42	Zaphod B.

→ Duplicates are removed here as well

Product

- Take all combinations of the input tables

Product

- Take all combinations of the input tables

Students

id	name
1	Arthur Dent
2	Ford Prefect

Product

- Take all combinations of the input tables

Students

id	name
1	Arthur Dent
2	Ford Prefect

$\times$

Product

- Take all combinations of the input tables

Students

id	name
1	Arthur Dent
2	Ford Prefect

$\times$

Rooms

room
E200
E242

Product

- Take all combinations of the input tables

Students

id	name
1	Arthur Dent
2	Ford Prefect

$\times$

Rooms

room
E200
E242

$=$

Product

- Take all combinations of the input tables

Students

id	name
1	Arthur Dent
2	Ford Prefect

$\times$

Rooms

room
E200
E242

$=$

Students $\times$ Rooms

id	name	room
1	Arthur Dent	E200
1	Arthur Dent	E242
2	Ford Prefect	E200
2	Ford Prefect	E242

Join

→ Product is useful to express joins:

Join

→ Product is useful to express joins:

Member

id	class
1	WDM
2	WDM

Join

→ Product is useful to express joins:

Member

id	class
1	WDM
2	WDM

$\bowtie$

Join

→ Product is useful to express joins:

Member

id	class
1	WDM
2	WDM

$\bowtie$

Class

class	date
WDM	Nov 21
ABC	Nov 24
WDM	Nov 28

Join

→ Product is useful to express joins:

Member

id	class
1	WDM
2	WDM

$\bowtie$

Class

class	date
WDM	Nov 21
ABC	Nov 24
WDM	Nov 28

=

Join

→ Product is useful to express joins:

Member

id	class
1	WDM
2	WDM

$\bowtie$

Class

class	date
WDM	Nov 21
ABC	Nov 24
WDM	Nov 28

$=$

Member $\bowtie$ Class

id	class	date
1	WDM	Nov 21
1	WDM	Nov 28
2	WDM	Nov 21
2	WDM	Nov 28

Join

→ Product is useful to express joins:

Member

id	class
1	WDM
2	WDM

$\bowtie$

Class

class	date
WDM	Nov 21
ABC	Nov 24
WDM	Nov 28

=

Member $\bowtie$ Class

id	class	date
1	WDM	Nov 21
1	WDM	Nov 28
2	WDM	Nov 21
2	WDM	Nov 28

Express Member $\bowtie$ Class with the previous operators:

Join

→ Product is useful to express joins:

Member

id	class
1	WDM
2	WDM

$\bowtie$

Class

class	date
WDM	Nov 21
ABC	Nov 24
WDM	Nov 28

$=$

Member $\bowtie$ Class

id	class	date
1	WDM	Nov 21
1	WDM	Nov 28
2	WDM	Nov 21
2	WDM	Nov 28

Express Member $\bowtie$ Class with the previous operators:

$$\text{Member} \times \text{Class}$$

Join

→ Product is useful to express joins:

Member

id	class
1	WDM
2	WDM

$\bowtie$

Class

class	date
WDM	Nov 21
ABC	Nov 24
WDM	Nov 28

=

Member $\bowtie$ Class

id	class	date
1	WDM	Nov 21
1	WDM	Nov 28
2	WDM	Nov 21
2	WDM	Nov 28

Express Member $\bowtie$ Class with the previous operators:

$\rho_{\textbf{class}\to\textbf{class2}}(\text{Member}) \times \text{Class}$

Join

→ Product is useful to express joins:

Member

id	class
1	WDM
2	WDM

$\bowtie$

Class

class	date
WDM	Nov 21
ABC	Nov 24
WDM	Nov 28

$=$

Member $\bowtie$ Class

id	class	date
1	WDM	Nov 21
1	WDM	Nov 28
2	WDM	Nov 21
2	WDM	Nov 28

Express Member $\bowtie$ Class with the previous operators:

$$\sigma_{\textbf{class}=\textbf{class2}} \left(\rho_{\textbf{class}\to\textbf{class2}}(\text{Member}) \times \text{Class} \right)$$

Join

$\rightarrow$ Product is useful to express joins:

Member

id	class
1	WDM
2	WDM

$\bowtie$

Class

class	date
WDM	Nov 21
ABC	Nov 24
WDM	Nov 28

$=$

Member $\bowtie$ Class

id	class	date
1	WDM	Nov 21
1	WDM	Nov 28
2	WDM	Nov 21
2	WDM	Nov 28

Express Member $\bowtie$ Class with the previous operators:

$$\Pi_{\text{id,class,date}}\Big(\sigma_{\textbf{class}=\textbf{class2}} \;\big(\; \rho_{\textbf{class}\rightarrow\textbf{class2}}(\text{Member}) \times \text{Class}\big)\Big)$$

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

−

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

−

S3

id	name
1	Arthur Dent
42	Zaphod B.

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

−

S3

id	name
1	Arthur Dent
42	Zaphod B.

=

Difference

- Take tuples that are in one table but not in the other
- Applies to two tables with same attributes

Students1

id	name
1	Arthur Dent
2	Ford Prefect

$-$

S3

id	name
1	Arthur Dent
42	Zaphod B.

$=$

Students1 $-$ S3

id	name
2	Ford Prefect

Outline

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
- $(\{$*unclassified*, *restricted*, *confidential*, *secret*, *top secret*$\}$, *top secret*, *unclassified*, $\min, \max)$: security semiring
- $(\mathbb{N} \cup \{\infty\}, \infty, 0, \min, +)$: tropical semiring
- $(\{$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, see further)
- $(\mathcal{P}(\mathcal{P}(\mathcal{X})), \emptyset, \{\emptyset\}, \cup, \Cup)$: Why-semiring over $\mathcal{X}$ $(A \Cup B := \{a \cup b \mid a \in A, b \in B\})$

Semiring provenance [Green et al., 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, cross product, union; joins can be simulated with renaming, cross product, selection, projection)
- We define a semantics for the provenance of a tuple $t \in q(D)$ inductively on the structure of q

Selection, renaming

Provenance annotations of selected tuples are unchanged

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

name	position	city	classification	prov
John	Director	New York	unclassified	t_1
Paul	Janitor	New York	restricted	t_2
Dave	Analyst	Paris	confidential	t_3
Ellen	Field agent	Berlin	secret	t_4
Magdalen	Double agent	Paris	top secret	t_5
Nancy	HR director	Paris	restricted	t_6
Susan	Analyst	Berlin	secret	t_7

n	position	city	classification	prov
John	Director	New York	unclassified	t_1
Paul	Janitor	New York	restricted	t_2

Projection

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

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

name	position	city	classification	prov
John	Director	New York	unclassified	t_1
Paul	Janitor	New York	restricted	t_2
Dave	Analyst	Paris	confidential	t_3
Ellen	Field agent	Berlin	secret	t_4
Magdalen	Double agent	Paris	top secret	t_5
Nancy	HR director	Paris	restricted	t_6
Susan	Analyst	Berlin	secret	t_7

city	prov
Berlin	$t_4 \oplus t_7$
New York	$t_1 \oplus t_2$
Paris	$t_3 \oplus t_5 \oplus t_6$

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	t_1
Paul	Janitor	New York	restricted	t_2
Dave	Analyst	Paris	confidential	t_3
Ellen	Field agent	Berlin	secret	t_4
Magdalen	Double agent	Paris	top secret	t_5
Nancy	HR director	Paris	restricted	t_6
Susan	Analyst	Berlin	secret	t_7

city	prov
Berlin	$t_4 \oplus t_7$
Paris	$t_3 \oplus t_5$

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	t_1
Paul	Janitor	New York	restricted	t_2
Dave	Analyst	Paris	confidential	t_3
Ellen	Field agent	Berlin	secret	t_4
Magdalen	Double agent	Paris	top secret	t_5
Nancy	HR director	Paris	restricted	t_6
Susan	Analyst	Berlin	secret	t_7

city	prov
Berlin	$t_4 \otimes t_7$
Paris	$t_3 \otimes t_5$

What can we do with it?

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

Boolean semiring: indicates in which possible worlds the tuple appears

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: universal provenance, see further

Why-semiring: Why-provenance [Buneman et al., 2001], set of combinations of tuples needed for a tuple to exist

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
Berlin	secret
New York	restricted
Paris	confidential

Notes [Green et al., 2007]

- Computing provenance has a PTIME data complexity overhead
- Semiring homomorphisms commute with provenance computation: if there is a homomorphism from K to 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 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

Outline

Semirings with monus [Amer, 1984, Geerts and Poggi, 2010]

- Some semirings can be equipped with a $\ominus$ verifying:
  - $a \oplus (b \ominus a) = b \oplus (a \ominus b)$
  - $(a \ominus b) \ominus c = a \ominus (b + c)$
  - $a \ominus a = \mathbb{0} \ominus a = \mathbb{0}$
- Boolean function semiring with $\land\neg$, Why-semiring with $\setminus$, counting semiring with truncated difference...
- Most natural semirings (but not all semirings [Amarilli and Monet, 2016]!) can be extended into semirings with monus
- Sometimes strange things happen [Amsterdamer et al., 2011]: e.g, $\otimes$ does not always distribute over $\ominus$
- Allows supporting full relational algebra with the $\setminus$ operator, still PTIME
- Semantics for Boolean function semiring coincides with that of Boolean provenance

Difference

Provenance annotations of diff-ed tuples are $\ominus$-ed

Example

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

name	position	city	classification	prov
John	Director	New York	unclassified	t_1
Paul	Janitor	New York	restricted	t_2
Dave	Analyst	Paris	confidential	t_3
Ellen	Field agent	Berlin	secret	t_4
Magdalen	Double agent	Paris	top secret	t_5
Nancy	HR director	Paris	restricted	t_6
Susan	Analyst	Berlin	secret	t_7

city	prov
Berlin	$t_4 \ominus t_7$
Paris	$t_5 \ominus t_3$

Provenance for aggregates [Amsterdamer et al., 2011, Fink et al., 2012]

- Trickier to define provenance for queries with aggregation, even in the Boolean case
- One can construct a K-semimodule $K * M$ for each monoid aggregate M over a provenance database with a semiring in K
- Data values become elements of the semimodule

Example ($\mathrm{count}(\pi_{\mathbf{name}}(\sigma_{\mathbf{city=``Paris"}}(R)))$)

$$t_3 * 1 + t_5 * 1 + t_6 * 1$$

Where-provenance [Buneman et al., 2001]

- Different form of provenance: captures from which database values come which output values
- Bipartite graph of provenance: two attribute values are connected if one can be produced from the other
- Axiomatized in [Buneman et al., 2001, Cheney et al., 2009]
- Cannot be captured by provenance semirings [Cheney et al., 2009], because of renaming (does not keep track of relation attributes), projection (does not remember which attribute values still exist), join (in a join, an output value comes from two different input values)

Outline

Representation systems

- In the Boolean semiring, the counting semiring, the security semiring: provenance annotations are elementary
- In the Boolean function semiring, the universal semiring, etc., provenance annotations can become quite complex
- Needs for compact representation of provenance annotations
- Lower the provenance computation complexity as much as possible

Provenance formulas

- Quite straightforward
- Formalism used in most of the provenance literature
- PTIME data complexity
- Expanding formulas (e.g., computing the monomials of a $\mathbb{N}[\mathcal{X}]$ provenance annotation) can result in an exponential blowup

Example

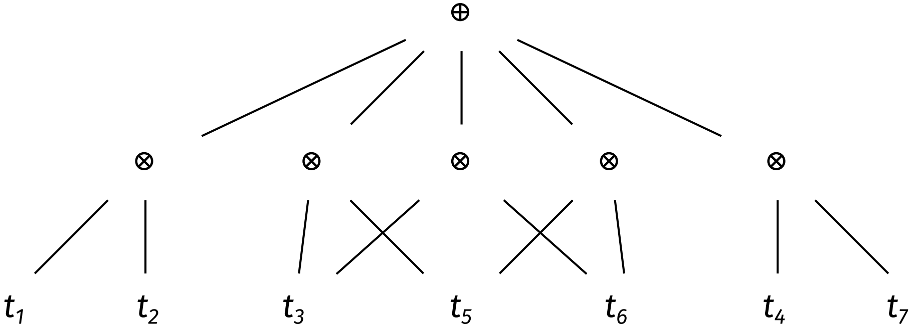
Is there a city with two different agents?

$$(t_1 \otimes t_2) \oplus (t_3 \otimes t_6) \oplus (t_3 \otimes t_5) \oplus (t_4 \otimes t_7) \oplus (t_5 \otimes t_6)$$

Provenance circuits [Deutch et al., 2014, Amarilli et al., 2015a]

- Use arithmetic circuits (Boolean circuits for Boolean provenance) 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...
  - Super-polynomially more concise than monotone provenance formula for linear Datalog programs [Deutch et al., 2014]
  - Quadratically more concise than provenance formulas for monadic second-order (MSO) formulas
  - More concise by a $\log$ factor than monotone provenance formulas for positive relational algebra queries, and by a $\log \log$ factor for non-monotone formulas [Wegener, 1987, Amarilli et al., 2016]

Example provenance circuit



OBDD and d-DNNF

- Various subclasses of Boolean circuits commonly used:

  OBDD: Ordered Binary Decision Diagrams

  d-DNNF: deterministic Decomposable Negation Normal Form
- OBDDs can be obtained in PTIME data complexity on bounded-treewidth databases [Amarilli et al., 2016]
- d-DNNFs can be obtained in linear-time data complexity on bounded-treewidth databases
- Applications to probabilistic query evaluation (see next)

Provenance cycluits [Amarilli et al., 2017b]

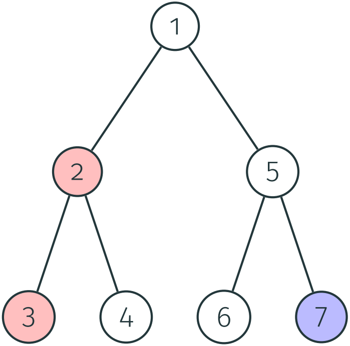
- Cycluit (cyclic circuit): arithmetic circuit with cycles
- Well-defined semantics on the Boolean semiring and some semirings where infinite loops do not matter
- Allows computing provenance in linear-time combined complexity for recursive queries of a certain form (ICG-Datalog of bounded body size [Amarilli et al., 2017b], capturing α-acyclic conjunctive queries, 2RPQs, etc.), on bounded tree-width databases
- Related to provenance equation systems and formal series introduced in [Green et al., 2007]

Outline

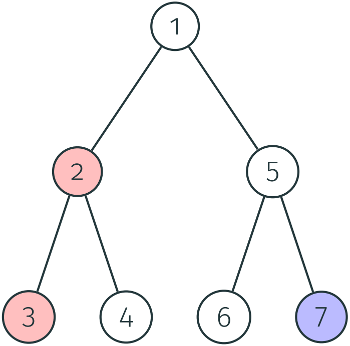
Definition

- The previous definitions of provenance work on relational data
- What if the data is a tree, e.g., an XML document, or a word?
- Problem: we can now have recursive queries, e.g., descendant queries
- A naive representation of provenance may no longer be polynomial in the database

Example: uncertain trees

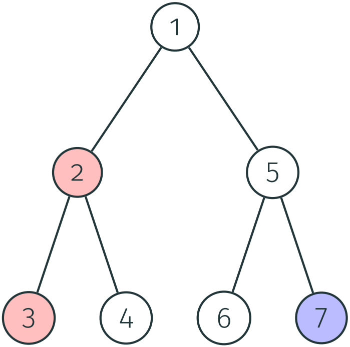


Example: uncertain trees



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

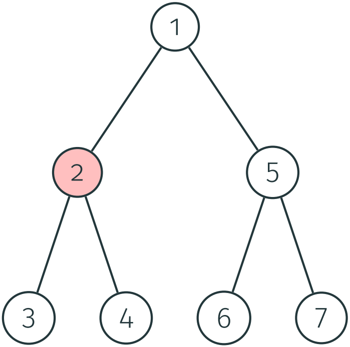
Example: uncertain trees



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

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

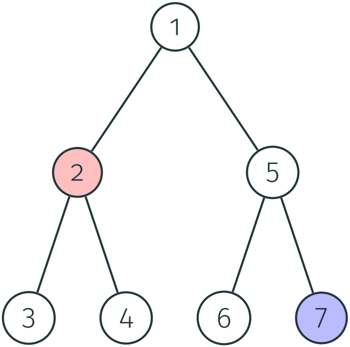
Example: uncertain trees



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

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

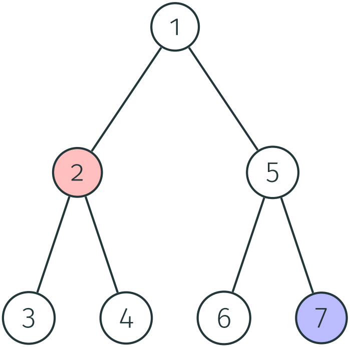
Example: uncertain trees



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

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

Example: uncertain trees

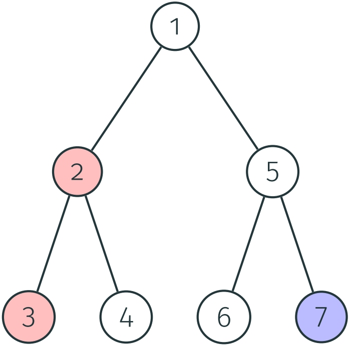


A valuation of a tree decides whether to keep (1) or discard (0) node labels

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

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

Example: uncertain trees



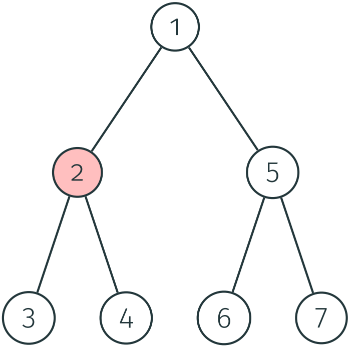
A valuation of a tree decides whether to keep (1) or discard (0) node labels

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

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

The tree automaton A accepts

Example: uncertain trees



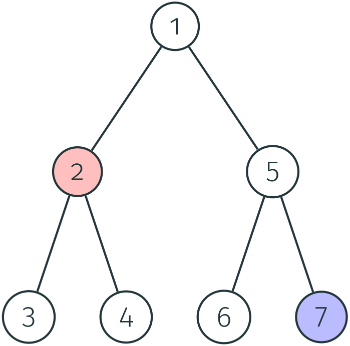
A valuation of a tree decides whether to keep (1) or discard (0) node labels

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

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

The tree automaton A rejects

Example: uncertain trees



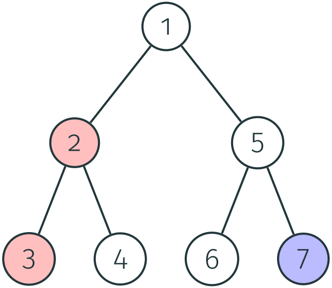
A valuation of a tree decides whether to keep (1) or discard (0) node labels

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

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

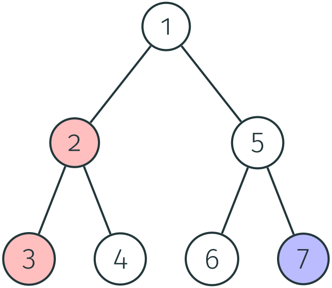
The tree automaton A accepts

Example: Provenance circuit



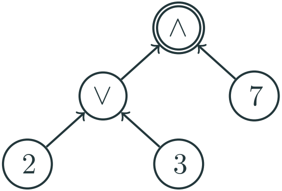
Query: *Is there both a pink and a blue node?*

Example: Provenance circuit

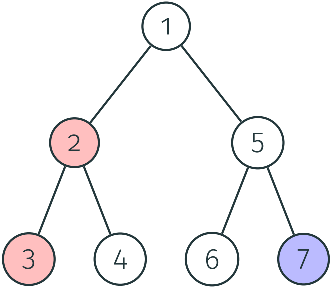


Query: *Is there both a pink and a blue node?*

Provenance circuit:

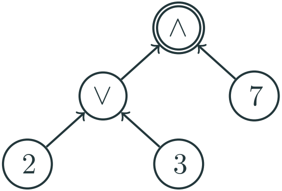


Example: Provenance circuit



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

Provenance circuit:



Formally:

- Tree automaton A, uncertain tree T, circuit C
- Variable gates of C: nodes of T
- Condition: Let ν be a valuation of T, then $\nu(C)$ iff A accepts $\nu(T)$

Building provenance circuits on trees [Amarilli et al., 2015b]

Theorem

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

Building provenance circuits on trees [Amarilli et al., 2015b]

Theorem

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

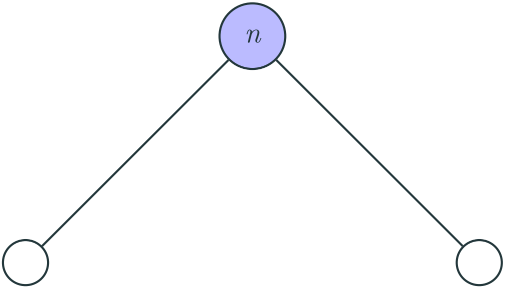
- Alphabet: 
- Automaton: *"Is there both a pink and a blue node?"*
- States: $\{\bot, B, P, \top\}$
- Final: $\{\top\}$
- Transitions:

Building provenance circuits on trees [Amarilli et al., 2015b]

Theorem

For any bottom-up tree automaton *A and input* tree *T,*
we can build a 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:

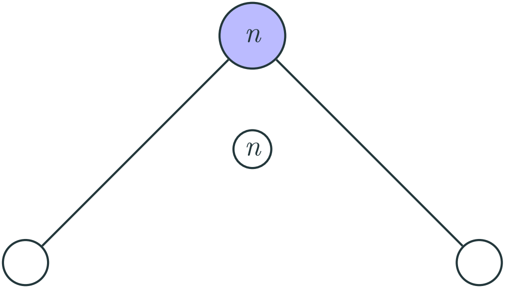


Building provenance circuits on trees [Amarilli et al., 2015b]

Theorem

For any bottom-up tree automaton A and input tree T,
we can build a 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:

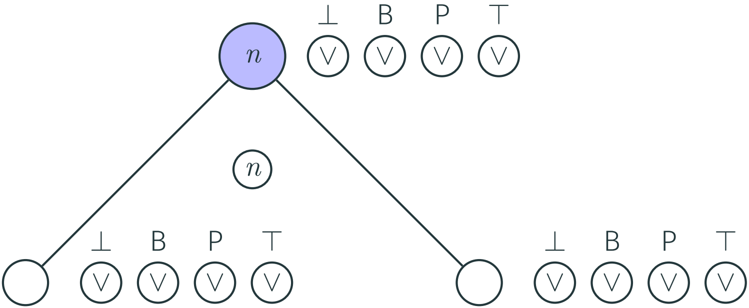


Building provenance circuits on trees [Amarilli et al., 2015b]

Theorem

For any bottom-up tree automaton A and input tree T, we can build a 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:

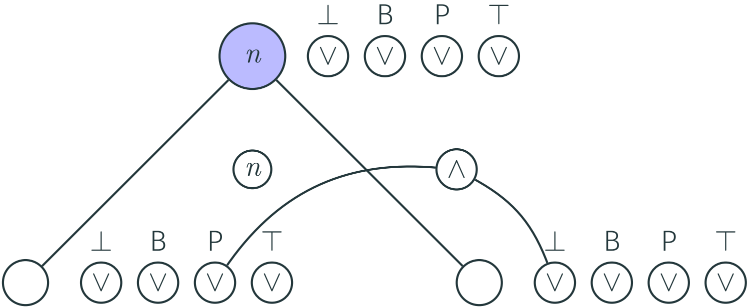


Building provenance circuits on trees [Amarilli et al., 2015b]

Theorem

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

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

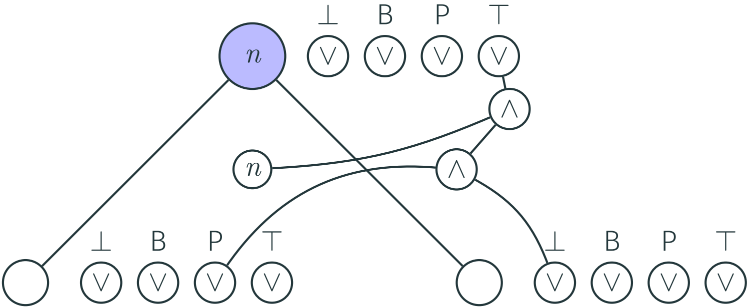


Building provenance circuits on trees [Amarilli et al., 2015b]

Theorem

For any bottom-up tree automaton A and input tree T, we can build a 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:

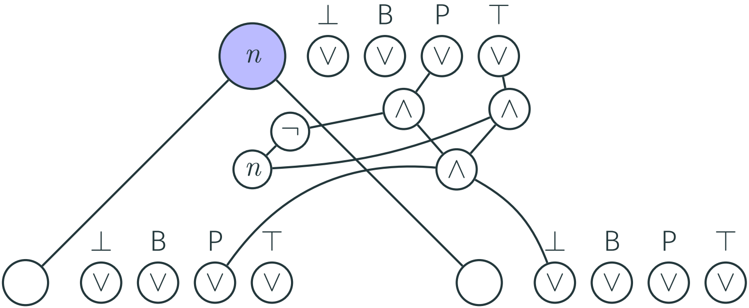


Building provenance circuits on trees [Amarilli et al., 2015b]

Theorem

For any bottom-up tree automaton A and input tree T, we can build a 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:



d-DNNFs

Lemma

If the tree automaton is unambiguous then the circuit is a d-DNNF

The circuit is a d-DNNF...

d-DNNFs

Lemma

If the tree automaton is unambiguous then the circuit is a d-DNNF

The circuit is a d-DNNF...

- (¬) gates only have variables as inputs

d-DNNFs

Lemma

If the tree automaton is unambiguous then the circuit is a d-DNNF

The circuit is a d-DNNF...

- $\neg$ gates only have variables as inputs
- $\vee$ gates always have mutually exclusive inputs

d-DNNFs

Lemma

If the tree automaton is unambiguous then the circuit is a d-DNNF

The circuit is a d-DNNF...

- ¬ gates only have variables as inputs
- ∨ gates always have mutually exclusive inputs
- ∧ gates are all on independent inputs

d-DNNFs

Lemma

If the tree automaton is unambiguous then the circuit is a d-DNNF

The circuit is a d-DNNF… … so probability computation is easy!

- (¬) gates only have variables as inputs
- (∨) gates always have mutually exclusive inputs
- (∧) gates are all on independent inputs

d-DNNFs

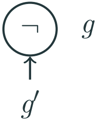
Lemma

If the tree automaton is unambiguous then the circuit is a d-DNNF

The circuit is a d-DNNF…

- $\neg$ gates only have variables as inputs
- $\vee$ gates always have mutually exclusive inputs
- $\wedge$ gates are all on independent inputs

… so probability computation is easy!



d-DNNFs

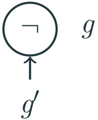
Lemma

If the tree automaton is unambiguous then the circuit is a d-DNNF

The circuit is a d-DNNF...

- ¬ gates only have variables as inputs
- ∨ gates always have mutually exclusive inputs
- ∧ gates are all on independent inputs

... so probability computation is easy!



$$P(g) := 1 - P(g')$$

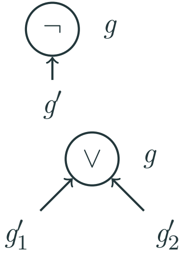
Lemma

If the tree automaton is unambiguous then the circuit is a d-DNNF

The circuit is a d-DNNF…

… so probability computation is easy!

- $\neg$ gates only have variables as inputs
- $\vee$ gates always have mutually exclusive inputs
- $\wedge$ gates are all on independent inputs



$$P(g) := 1 - P(g')$$

d-DNNFs

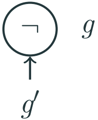
Lemma

If the tree automaton is unambiguous then the circuit is a d-DNNF

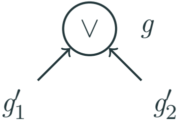
The circuit is a d-DNNF…

- $\neg$ gates only have variables as inputs
- $\lor$ gates always have mutually exclusive inputs
- $\land$ gates are all on independent inputs

… so probability computation is easy!



$$P(g) := 1 - P(g')$$



$$P(g) := P(g'_1) + P(g'_2)$$

d-DNNFs

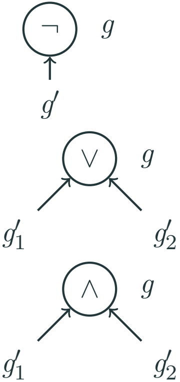
Lemma

If the tree automaton is unambiguous then the circuit is a d-DNNF

The circuit is a d-DNNF...

- (¬) gates only have variables as inputs
- (∨) gates always have mutually exclusive inputs
- (∧) gates are all on independent inputs

... so probability computation is easy!



$$P(g) := 1 - P(g')$$

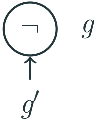
$$P(g) := P(g'_1) + P(g'_2)$$

Lemma

If the tree automaton is unambiguous then the circuit is a d-DNNF

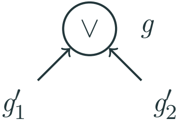
The circuit is a d-DNNF... ... so probability computation is easy!

- (¬) gates only have variables as inputs



$$P(g) := 1 - P(g')$$

- (∨) gates always have mutually exclusive inputs



$$P(g) := P(g'_1) + P(g'_2)$$

- (∧) gates are all on independent inputs

$$P(g) := P(g'_1) \times P(g'_2)$$

Extensions of provenance circuits on trees

- Can be extended to more general provenance than Boolean provenance, for some query languages [Amarilli et al., 2015b]
- Results extend to databases of bounded treewidth (extension of Courcelle's theorem), and essentially only to such databases [Amarilli et al., 2016]
- Has connections with probability computation methods on bounded-treewidth graphical models [Amarilli et al., 2019c]
- Provenance representation also useful for efficient enumeration of the answers to non-Boolean queries (ongoing work!) [Amarilli et al., 2017a, 2019a,b]

Queries with free variables

- We have talked about Boolean provenance for Boolean queries:

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

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

Queries with free variables

- We have talked about Boolean provenance for Boolean queries:

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

$$Q() : \exists x\, y\; P_{\color{pink}\bullet}(x) \wedge P_{\color{blue}\bullet}(y)$$

- For enumeration, we consider queries with free variables:

"Find all pairs of a pink and a blue node?"

$$Q(x, y) : P_{\color{pink}\bullet}(x) \wedge P_{\color{blue}\bullet}(y)$$

Circuits as factorized representations of query results

- Query: $Q(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$

Circuits as factorized representations of query results

- Query: $Q(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

Circuits as factorized representations of query results

- Query: $Q(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

$\rightarrow$ Add special facts to materialize all possible assignments

  - e.g., $X_i(a_j)$ means element a_i is mapped to variable X_j

Circuits as factorized representations of query results

- Query: $Q(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

→ Add special facts to materialize all possible assignments
  - e.g., $X_i(a_j)$ means element a_i is mapped to variable X_j

→ The provenance circuit of Q is now a factorized representation which describes all the tuples that make Q true

Circuits as factorized representations of query results

- Query: $Q(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

→ Add special facts to materialize all possible assignments
  - e.g., $X_i(a_j)$ means element a_i is mapped to variable X_j

→ The provenance circuit of Q is now a factorized representation which describes all the tuples that make Q true

Example query:

$Q(X_1, X_2) : P_{\color{pink}\bullet}(x) \wedge P_{\color{blue}\bullet}(y)$

Circuits as factorized representations of query results

- Query: $Q(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

→ Add special facts to materialize all possible assignments
  - e.g., $X_i(a_j)$ means element a_i is mapped to variable X_j

→ The provenance circuit of Q is now a factorized representation which describes all the tuples that make Q true

Example query:

$Q(X_1, X_2) : P_{\color{red}\bullet}(x) \wedge P_{\color{blue}\bullet}(y)$

Database:

Circuits as factorized representations of query results

- Query: $Q(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

→ Add special facts to materialize all possible assignments
  - e.g., $X_i(a_j)$ means element a_i is mapped to variable X_j

→ The provenance circuit of Q is now a factorized representation which describes all the tuples that make Q true

Example query:

$Q(X_1, X_2) : P_{\bigcirc}(x) \wedge P_{\bigcirc}(y)$

Database:

Results:

X_1	X_2
1	3
1	5

Circuits as factorized representations of query results

- Query: $Q(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

→ Add special facts to materialize all possible assignments
  - e.g., $X_i(a_j)$ means element a_i is mapped to variable X_j

→ The provenance circuit of Q is now a factorized representation which describes all the tuples that make Q true

Example query:

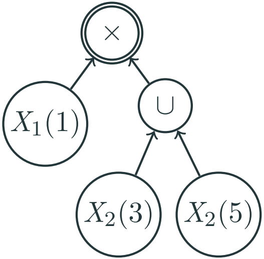
$Q(X_1, X_2) : P_{\bigcirc}(x) \wedge P_{\bigcirc}(y)$

Database:

Results:

X_1	X_2
1	3
1	5

Provenance circuit:



Circuits as factorized representations of query results

- Query: $Q(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

→ Add special facts to materialize all possible assignments
  - e.g., $X_i(a_j)$ means element a_i is mapped to variable X_j

→ The provenance circuit of Q is now a factorized representation which describes all the tuples that make Q true

Example query:

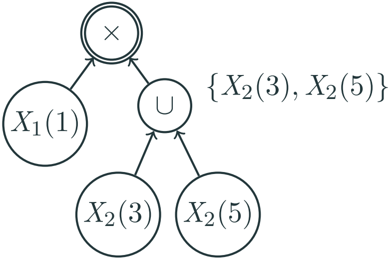
$Q(X_1, X_2) : P_{\color{pink}\bullet}(x) \wedge P_{\color{blue}\bullet}(y)$

Database:

Results:

X_1	X_2
1	3
1	5

Provenance circuit:



Circuits as factorized representations of query results

- Query: $Q(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

→ Add special facts to materialize all possible assignments
  - e.g., $X_i(a_j)$ means element a_i is mapped to variable X_j

→ The provenance circuit of Q is now a factorized representation which describes all the tuples that make Q true

Example query:

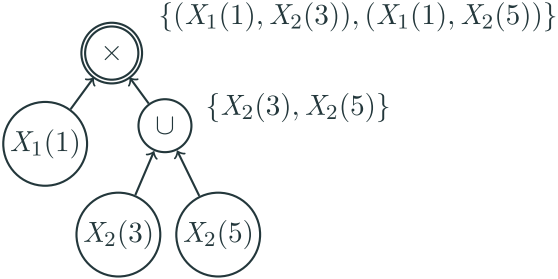
$Q(X_1, X_2) : P_{\bullet}(x) \wedge P_{\bullet}(y)$

Database:

Results:

X_1	X_2
1	3
1	5

Provenance circuit:



Application of factorized representations

This factorized representation of the results of the query can be computed in linear time in the data

Application of factorized representations

This factorized representation of the results of the query can be computed in linear time in the data

- Application: Counting query results [Arnborg et al., 1991]

Application of factorized representations

This factorized representation of the results of the query can be computed in linear time in the data

- Application: Counting query results [Arnborg et al., 1991]
  - Exclusive $\vee$ means $+$, independent $\wedge$ means $\times$
  - Reproves existing result: [Arnborg et al., 1991]

Application of factorized representations

This factorized representation of the results of the query can be computed in linear time in the data

- Application: Counting query results [Arnborg et al., 1991]
  - Exclusive $\vee$ means $+$, independent $\wedge$ means $\times$
  - Reproves existing result: [Arnborg et al., 1991]
- Application: Constant-delay enumeration of query results

Application of factorized representations

This factorized representation of the results of the query can be computed in linear time in the data

- Application: Counting query results [Arnborg et al., 1991]
  - Exclusive $\vee$ means $+$, independent $\wedge$ means $\times$
  - Reproves existing result: [Arnborg et al., 1991]

- Application: Constant-delay enumeration of query results
  - Requires some linear-time preprocessing of the input circuit
  - Exclusive $\vee$ means disjoint $\cup$, independent $\wedge$ means relational $\times$
  - New modular proof of existing enumeration result [Bagan, 2006, Kazana and Segoufin, 2013, Amarilli et al., 2017a]
  - Extensions to support updates on the database [Amarilli et al., 2019b]

Application of factorized representations

This factorized representation of the results of the query can be computed in linear time in the data

- Application: Counting query results [Arnborg et al., 1991]
  - Exclusive $\vee$ means $+$, independent $\wedge$ means $\times$
  - Reproves existing result: [Arnborg et al., 1991]
- Application: Constant-delay enumeration of query results
  - Requires some linear-time preprocessing of the input circuit
  - Exclusive $\vee$ means disjoint $\cup$, independent $\wedge$ means relational $\times$
  - New modular proof of existing enumeration result [Bagan, 2006, Kazana and Segoufin, 2013, Amarilli et al., 2017a]
  - Extensions to support updates on the database [Amarilli et al., 2019b]

Outline

Desiderata for a provenance-aware DBMS

- Extends a widely used database management system
- Easy to deploy
- Easy to use, transparent for the user
- Provenance automatically maintained as the user interacts with the database management system
- Provenance computation benefits from query optimization within the DBMS
- Allow probability computation based on provenance
- Any form of provenance can be computed: Boolean provenance, semiring provenance in any semiring (possibly, with monus), aggregate provenance, where-provenance, on demand

ProvSQL: Provenance within PostgreSQL (1/2) [Senellart et al., 2018]

- Lightweight extension/plugin for PostgreSQL ≥ 9.5
- Provenance annotations stored as UUIDs, in an extra attribute of each provenance-aware relation
- A provenance circuit relating UUIDs of elementary provenance annotations and arithmetic gates stored as table
- All computations done in the universal semiring (more precisely, with monus, in the free semiring with monus; for where-provenance, in a free term algebra)

ProvSQL: Provenance within PostgreSQL (2/2) [Senellart et al., 2018]

- Query rewriting to automatically compute output provenance attributes in terms of the query and input provenance attributes:
  - Duplicate elimination (DISTINCT, set union) results in aggregation of provenance values with $\oplus$
  - Cross products, joins results in combination of provenance values with $\otimes$
  - Difference results in combination of provenance values with $\ominus$
- Additional circuit gates on projection, join for support of where-provenance
- Probability computation from the provenance circuits, via various methods (naive, sampling, compilation to d-DNNFs)

Challenges

- Low-level access to PostgreSQL data structures in extensions
- No simple query rewriting mechanism
- SQL is much less clean than the relational algebra
- Multiset semantics by default in SQL
- SQL is a very rich language, with many different ways of expressing the same thing
- Inherent limitations: e.g., no aggregation within recursive queries
- Implementing provenance computation should not slow down the computation
- User-defined functions, updates, etc.: unclear how provenance should work

ProvSQL: Current status

- Supported SQL language features:
  - Regular SELECT-FROM-WHERE queries (aka conjunctive queries with multiset semantics)
  - JOIN queries (regular joins and outer joins; semijoins and antijoins are not currently supported)
  - SELECT queries with nested SELECT subqueries in the FROM clause
  - GROUP BY queries (without aggregation)
  - SELECT DISTINCT queries (i.e., set semantics)
  - UNION's or UNION ALL's of SELECT queries
  - EXCEPT queries
- Longer term project: aggregate computation
- Try it (and see a demo) from `https://github.com/PierreSenellart/provsql`

Outline

Relational Data Provenance [Senellart, 2017]

- Quite rich foundations of provenance management:
  - Different types of provenance
  - Semiring formalism to unify most provenance forms
  - (Partial) extensions for difference, recursive queries, aggregation
  - Compact provenance representation formalisms
- Some theory still missing:
  - Provenance and updates
  - Going beyond the relational algebra for full semiring provenance
- Now is the time to work on concrete implementation
- Need good implementation to convince users they should track provenance!
- How to combine provenance computation and efficient query evaluation, e.g., through tree decompositions?

Merci.

https://github.com/PierreSenellart/provsql

https://youtu.be/iqzSNfGHbEE?vq=hd1080

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

Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *Proc. ICALP*, pages 56–68, Kyoto, Japan, July 2015a.

Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances (extended version), November 2015b. CoRR abs/1511.08723.

Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Tractable lineages on treelike instances: Limits and extensions. In *Proc. PODS*, pages 355–370, San Francisco, USA, June 2016.

Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A Circuit-Based Approach to Efficient Enumeration. In *ICALP*, 2017a.

Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart. Combined tractability of query evaluation via tree automata and cycluits. In *ICDT*, 2017b.

Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-Delay Enumeration for Nondeterministic Document Spanners. In *ICDT*, 2019a.

Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on Trees with Tractable Combined Complexity and Efficient Updates. Under review, 2019b.

Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. Connecting Knowledge Compilation Classes and Width Parameters. Under review, 2019c.

K. Amer. *Algebra Universalis*, 18, 1984.

Yael Amsterdamer, Daniel Deutch, and Val Tannen. On the limitations of provenance for queries with difference. In *TaPP*, 2011.

Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *CSL*, 2006.

Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings.*, 2001.

James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4), 2009.

Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for Datalog provenance. In *ICDT*, 2014.

Robert Fink, Larisa Han, and Dan Olteanu. Aggregation in probabilistic databases via knowledge compilation. *Proceedings of the VLDB Endowment*, 5(5):490–501, 2012.

Floris Geerts and Antonella Poggi. On database query languages for k-relations. *J. Applied Logic*, 8(2), 2010.

Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. *IEEE Data Eng. Bull.*, 29(1), 2006.

Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, 2007.

Tomasz Imieliński and Jr. Lipski, Witold. Incomplete information in relational databases. *J. ACM*, 31(4), 1984.

Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *TOCL*, 14(4), 2013.

Pierre Senellart. Provenance and probabilities in relational databases: From theory to practice. *SIGMOD Record*, 46(4), December 2017.

Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. ProvSQL: provenance and probability management in postgresql. 2018. Demonstration.

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

Ingo Wegener. *The Complexity of Boolean Functions*. Wiley, 1987.