Query Lineages and Knowledge Compilation

Antoine Amarilli[1]

November 13, 2019

[1]Télécom Paris

Query Lineage Definitions

- Relational database instance I: set of facts
- Boolean query Q: take an instance and answer yes/no

Query Lineage Definitions

- Relational database instance I: set of facts
- Boolean query Q: take an instance and answer yes/no

Example: query Q:

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

Query Lineage Definitions

- **Relational database instance** I: set of facts
- **Boolean query** Q: take an instance and answer yes/no

Example: query Q:

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

R	
a	b
a'	b

S	
b	c

Query Lineage Definitions

- Relational database instance I: set of facts
- Boolean query Q: take an instance and answer yes/no
- Lineage of Q on I: a Boolean circuit over the facts of I accepting exactly the subsets of I where Q is true

Example: query Q:

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

R	
a	b
a'	b

S	
b	c

Query Lineage Definitions

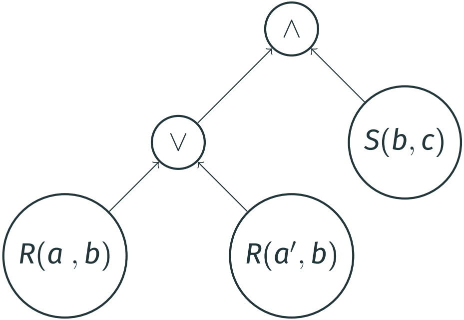
- **Relational database instance** I: set of facts
- **Boolean query** Q: take an instance and answer yes/no
- **Lineage** of Q on I: a **Boolean circuit** over the facts of I accepting exactly the **subsets** of I where Q is true

Example: query Q:

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

R	
a	b
a'	b

S	
b	c



Why bother? Applications of query lineages

- Evaluation: the lineage gives you the query answer

Why bother? Applications of query lineages

- **Evaluation:** the lineage gives you the query answer
- **Counting:**
  - Compute the **probability** that the query is true
  - Count **how many subinstances** satisfy the query

Why bother? Applications of query lineages

- Evaluation: the lineage gives you the query answer
- Counting:
  - Compute the probability that the query is true
  - Count how many subinstances satisfy the query
- Enumeration: efficiently enumerate the subinstances

Why bother? Applications of query lineages

- **Evaluation:** the lineage gives you the query answer
- **Counting:**
  - Compute the **probability** that the query is true
  - Count **how many subinstances** satisfy the query
- **Enumeration:** efficiently **enumerate** the subinstances
- **Explanation:**
  - Representation of **why** the query is true
  - **What-if:** is the query still true without these facts?

Another application: Query answers

Lineages can also represent the answers to non-Boolean queries!

Another application: Query answers

Lineages can also represent the answers to non-Boolean queries!

- Study answers of non-Boolean query $Q(x, y)$ on instance I

Another application: Query answers

Lineages can also represent the **answers** to **non-Boolean queries**!

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

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

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

Another application: Query answers

Lineages can also represent the **answers** to **non-Boolean queries**!

- Study answers of **non-Boolean query** $Q(x, y)$ on instance I
- Add **assignment facts** $X(v), Y(v)$ to I for each element v (linear)

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

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

Another application: Query answers

Lineages can also represent the **answers** to **non-Boolean queries**!

- Study answers of **non-Boolean query** $Q(x, y)$ on instance I — $Q(x, y) : \exists z\, R(x, y) \wedge S(y, z)$; $I : R(a, b), R(a', b), S(b, c)$
- Add **assignment facts** $X(v), Y(v)$ to I for each element v (linear) — $X(a), X(a'), X(b), X(c)$; $Y(a), Y(a'), Y(b), Y(c)$

Another application: Query answers

Lineages can also represent the **answers** to **non-Boolean queries**!

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

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

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

- Add **assignment facts** $X(v), Y(v)$ to I 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** $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$

Another application: Query answers

Lineages can also represent the answers to non-Boolean queries!

- Study answers of non-Boolean query $Q(x, y)$ on instance I

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

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

- Add assignment facts $X(v), Y(v)$ to I 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 $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$

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

Another application: Query answers

Lineages can also represent the answers to non-Boolean queries!

- Study answers of non-Boolean query $Q(x, y)$ on instance I

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

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

- Add assignment facts $X(v), Y(v)$ to I 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 a lineage C' of Q' on I plus assignment facts

Another application: Query answers

Lineages can also represent the answers to non-Boolean queries!

- Study answers of non-Boolean query $Q(x, y)$ on instance I

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

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

- Add assignment facts $X(v), Y(v)$ to I 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 a lineage C' of Q' on I plus assignment facts

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

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

Another application: Query answers

Lineages can also represent the answers to non-Boolean queries!

- Study answers of non-Boolean query $Q(x, y)$ on instance I

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

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

- Add assignment facts $X(v), Y(v)$ to I 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 a lineage C' of Q' on I 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

Another application: Query answers

Lineages can also represent the answers to non-Boolean queries!

- Study answers of non-Boolean query $Q(x, y)$ on instance I

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

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

- Add assignment facts $X(v), Y(v)$ to I 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 a lineage C' of Q' on I 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

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

Another application: Query answers

Lineages can also represent the **answers** to **non-Boolean queries**!

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

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

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

- Add **assignment facts** $X(v), Y(v)$ to I 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 a **lineage** C' of Q' on I 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 $\mathbf{1}$ except assignment facts

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

$\rightarrow$ The circuit C represents the **query answers**

Another application: Query answers

Lineages can also represent the answers to non-Boolean queries!

- Study answers of non-Boolean query $Q(x, y)$ on instance I $\quad Q(x, y) : \exists z\, R(x, y) \land S(y, z)$ $\quad I : R(a, b), R(a', b), S(b, c)$
- Add assignment facts $X(v), Y(v)$ to I for each element v (linear) $\quad X(a), X(a'), X(b), X(c)$ $\quad Y(a), Y(a'), Y(b), Y(c)$
- Consider the Boolean query $Q' : X(x) \land Y(y) \land Q(x, y)$ $\quad X(x) \land Y(y) \land (\exists z\, R(x, y) \land S(y, z))$
- Compute a lineage C' of Q' on I plus assignment facts $\quad (X(a) \land R(a, b) \lor X(a') \land R(a', b)) \land Y(b) \land S(b, c)$
- Define C by replacing all variables by $\mathbf{1}$ except assignment facts $\quad (X(a) \lor X(a')) \land Y(b)$

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

Another application: Query answers

Lineages can also represent the answers to non-Boolean queries!

- Study answers of non-Boolean query $Q(x, y)$ on instance I — $Q(x, y) : \exists z \, R(x, y) \wedge S(y, z)$; $I : R(a, b), R(a', b), S(b, c)$
- Add assignment facts $X(v), Y(v)$ to I 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 a lineage C' of Q' on I 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)

We can count the answers, enumerate them, etc.

Related work: Semiring provenance

Semiring provenance ([Green et al., 2007], PODS ToT award): annotate results of a relational algebra query with a semiring expression

A B C	
$a\ b\ c$	p
$d\ b\ e$	r
$f\ g\ e$	s

(a)

A C	
$a\ c$	$\{p\}$
$a\ e$	$\{p, r\}$
$d\ c$	$\{p, r\}$
$d\ e$	$\{r, s\}$
$f\ e$	$\{r, s\}$

(b)

A C	
$a\ c$	$2p^2$
$a\ e$	pr
$d\ c$	pr
$d\ e$	$2r^2 + rs$
$f\ e$	$2s^2 + rs$

(c)

Figure 5: Why-prov. and provenance polynomials

Related work: Semiring provenance

Semiring provenance ([Green et al., 2007], PODS ToT award): annotate results of a relational algebra query with a semiring expression

A B C	
$a\ b\ c$	p
$d\ b\ e$	r
$f\ g\ e$	s

(a)

A C	
$a\ c$	$\{p\}$
$a\ e$	$\{p, r\}$
$d\ c$	$\{p, r\}$
$d\ e$	$\{r, s\}$
$f\ e$	$\{r, s\}$

(b)

A C	
$a\ c$	$2p^2$
$a\ e$	pr
$d\ c$	pr
$d\ e$	$2r^2 + rs$
$f\ e$	$2s^2 + rs$

(c)

Figure 5: Why-prov. and provenance polynomials

What is the difference?

- Lineage = provenance in the semiring of Boolean functions

Related work: Semiring provenance

Semiring provenance ([Green et al., 2007], PODS ToT award): annotate results of a relational algebra query with a semiring expression

(a)

A B C	
$a\ b\ c$	p
$d\ b\ e$	r
$f\ g\ e$	s

(b)

A C	
$a\ c$	$\{p\}$
$a\ e$	$\{p, r\}$
$d\ c$	$\{p, r\}$
$d\ e$	$\{r, s\}$
$f\ e$	$\{r, s\}$

(c)

A C	
$a\ c$	$2p^2$
$a\ e$	pr
$d\ c$	pr
$d\ e$	$2r^2 + rs$
$f\ e$	$2s^2 + rs$

Figure 5: Why-prov. and provenance polynomials

What is the difference?

- Lineage = provenance in the semiring of Boolean functions
  - No multiplicity of facts or derivations

$\rightarrow$ Essentially only make sense for relational algebra

Related work: Semiring provenance

Semiring provenance ([Green et al., 2007], PODS ToT award): annotate results of a relational algebra query with a semiring expression

A B C	
$a\ b\ c$	p
$d\ b\ e$	r
$f\ g\ e$	s

(a)

A C	
$a\ c$	$\{p\}$
$a\ e$	$\{p, r\}$
$d\ c$	$\{p, r\}$
$d\ e$	$\{r, s\}$
$f\ e$	$\{r, s\}$

(b)

A C	
$a\ c$	$2p^2$
$a\ e$	pr
$d\ c$	pr
$d\ e$	$2r^2 + rs$
$f\ e$	$2s^2 + rs$

(c)

Figure 5: Why-prov. and provenance polynomials

What is the difference?

- Lineage = provenance in the semiring of Boolean functions
  - No multiplicity of facts or derivations
  - → Essentially only make sense for relational algebra
- Circuit representation: more concise

Computing lineages: theory

- Unions of Conjunctive Queries (UCQ)

Theorem

For any UCQ, given an instance, we can construct its lineage in polynomial time.

Computing lineages: theory

- Unions of Conjunctive Queries (UCQ)

Theorem

For any UCQ, given an instance, we can construct its lineage in polynomial time. *(disjunction of all matches)*

Computing lineages: theory

- Unions of Conjunctive Queries (UCQ)

Theorem

For any UCQ, given an instance, we can construct its lineage in polynomial time. *(disjunction of all matches)*

- Acyclic Conjunctive Queries (ACQ)

Theorem

For any ACQ, given an instance, we can construct its lineage in linear time.

Computing lineages: theory

- Unions of Conjunctive Queries (UCQ)

Theorem

For any UCQ, given an instance, we can construct its lineage in polynomial time. *(disjunction of all matches)*

- Acyclic Conjunctive Queries (ACQ)

Theorem

For any ACQ, given an instance, we can construct its lineage in linear time. *(following a join tree)*

Computing lineages: theory

- Unions of Conjunctive Queries (UCQ)

Theorem

For any UCQ, given an instance, we can construct its lineage in polynomial time. *(disjunction of all matches)*

- Acyclic Conjunctive Queries (ACQ)

Theorem

For any ACQ, given an instance, we can construct its lineage in linear time. *(following a join tree)*

- Monadic Second Order queries (MSO)

Theorem

For any MSO query, given a tree (or word), we can construct its lineage in linear time.

Computing lineages: theory

- Unions of Conjunctive Queries (UCQ)

Theorem

For any UCQ, given an instance, we can construct its lineage in polynomial time. *(disjunction of all matches)*

- Acyclic Conjunctive Queries (ACQ)

Theorem

For any ACQ, given an instance, we can construct its lineage in linear time. *(following a join tree)*

- Monadic Second Order queries (MSO)

Theorem

For any MSO query, given a tree (or word), we can construct its lineage in linear time. *(automaton product)*

Computing lineages: theory

- Unions of Conjunctive Queries (UCQ)

Theorem

For any UCQ, given an instance, we can construct its lineage in polynomial time. *(disjunction of all matches)*

- Acyclic Conjunctive Queries (ACQ)

Theorem

For any ACQ, given an instance, we can construct its lineage in linear time. *(following a join tree)*

- Monadic Second Order queries (MSO)

Theorem

For any MSO query, given a tree (or word), we can construct its lineage in linear time. *(automaton product)*

- Datalog: see [Deutch et al., 2014], PTIME

Computing lineages: practice

- ProvSQL: PostgreSQL extension to compute query lineages
- Keeps track of the lineage of query results as a circuit

Computing lineages: practice

- ProvSQL: PostgreSQL extension to compute query lineages
- Keeps track of the lineage of query results as a circuit

```
a3nm=# SELECT id, name, city FROM personnel;
 id |   name   |   city
----+----------+----------
  1 | John     | New York
  2 | Paul     | New York
  3 | Dave     | Paris
  4 | Ellen    | Berlin
  5 | Magdalen | Paris
  6 | Nancy    | Paris
  7 | Susan    | Berlin
(7 rows)

a3nm=# SELECT *,formula(provenance(), 'personnel_id') FROM
(SELECT DISTINCT city FROM personnel) t;
   city   |   formula
----------+-------------
 Paris    | (3 ⊕ 5 ⊕ 6)
 Berlin   | (4 ⊕ 7)
 New York | (1 ⊕ 2)
(3 rows)
```

You can run it! https://github.com/PierreSenellart/provsql

Knowledge compilation and tractable circuit classes

Knowledge compilation:

- Translate your problem to a circuit
- Design general-purpose algorithms on the circuits
  - $\rightarrow$ Satisfiability, counting, probability computation, enumeration...

Knowledge compilation and tractable circuit classes

Knowledge compilation:

- Translate your problem to a **circuit**
- Design **general-purpose algorithms** on the circuits
  - $\rightarrow$ Satisfiability, counting, probability computation, enumeration...

Without knowledge compilation:

$O(n^2)$ algorithms

Setting A → Task 1
Setting A → Task 2

Setting B → Task 1
Setting B → Task 2

Knowledge compilation and tractable circuit classes

Knowledge compilation:

- Translate your problem to a **circuit**
- Design **general-purpose algorithms** on the circuits
  - $\rightarrow$ Satisfiability, counting, probability computation, enumeration…

Without knowledge compilation:
$O(n^2)$ algorithms

Setting A $\rightarrow$ Task 1
Setting A $\rightarrow$ Task 2

Setting B $\rightarrow$ Task 1
Setting B $\rightarrow$ Task 2

With knowledge compilation:
$O(n)$ algorithms

Setting A $\longrightarrow$ **Circuit**

Setting B $\longrightarrow$ **Circuit**

Knowledge compilation and tractable circuit classes

Knowledge compilation:

- Translate your problem to a **circuit**
- Design **general-purpose algorithms** on the circuits
  - $\rightarrow$ Satisfiability, counting, probability computation, enumeration...

Without knowledge compilation: $O(n^2)$ algorithms

Setting A $\rightarrow$ Task 1

Setting A $\rightarrow$ Task 2

Setting B $\rightarrow$ Task 1

Setting B $\rightarrow$ Task 2

With knowledge compilation: $O(n)$ algorithms

Setting A $\longrightarrow$ **Circuit**

Setting B $\longrightarrow$ **Circuit**

Circuit $\longrightarrow$ Task 1

Circuit $\longrightarrow$ Task 2

Knowledge compilation and tractable circuit classes

Knowledge compilation:

- Translate your problem to a **circuit**
- Design **general-purpose algorithms** on the circuits
  - → Satisfiability, counting, probability computation, enumeration...

Without knowledge compilation:
$O(n^2)$ algorithms

$$\begin{array}{l} \text{Setting A} \to \text{Task 1} \\ \text{Setting A} \to \text{Task 2} \\ \text{Setting B} \to \text{Task 1} \\ \text{Setting B} \to \text{Task 2} \end{array}$$

With knowledge compilation:
$O(n)$ algorithms

$$\begin{array}{l} \text{Setting A} \longrightarrow \textbf{Circuit} \\ \text{Setting B} \longrightarrow \textbf{Circuit} \\ \textbf{Circuit} \longrightarrow \text{Task 1} \\ \textbf{Circuit} \longrightarrow \text{Task 2} \end{array}$$

→ Tractability: use tractable **circuit classes**

Tractable circuit classes: Theory vs practice

There is a whole zoo of tractable circuit classes...

$$
\begin{tikzcd}
 & \text{dec-DNNF} \arrow[rr, two heads] & & \text{d-DNNF} \arrow[rr, two heads] & & \text{DNNF} \\
\text{dec-SDNNF} \arrow[ur, two heads] \arrow[rr, two heads] & & \text{d-SDNNF} \arrow[ur, two heads] \arrow[rr, two heads] & & \text{SDNNF} \arrow[ur, two heads] & \\
 & \text{FBDD} \arrow[uu] \arrow[rr, two heads] & & \text{uFBDD} \arrow[uu] \arrow[rr, two heads] & & \text{nFBDD} \arrow[uu] \\
\text{OBDD} \arrow[uu] \arrow[ur, two heads] \arrow[rr, two heads] & & \text{uOBDD} \arrow[uu] \arrow[ur, two heads] \arrow[rr, two heads] & & \text{nOBDD} \arrow[uu] \arrow[ur, two heads] &
\end{tikzcd}
$$

Tractable circuit classes: Theory vs practice

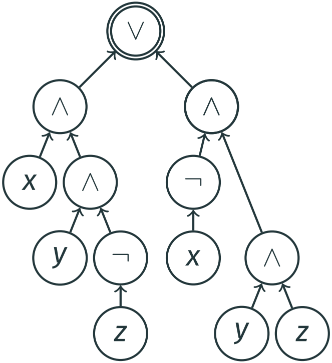
There is a whole **zoo** of tractable circuit classes...

$$
\begin{tikzcd}
 & \text{dec-DNNF} \arrow[rr, two heads] & & \text{d-DNNF} \arrow[rr, two heads] & & \text{DNNF} \\
\text{dec-SDNNF} \arrow[ur] \arrow[rr, two heads] & & \text{d-SDNNF} \arrow[ur] \arrow[rr, two heads] & & \text{SDNNF} \arrow[ur] & \\
 & \text{FBDD} \arrow[uu] \arrow[rr, two heads] & & \text{uFBDD} \arrow[uu] \arrow[rr, two heads] & & \text{nFBDD} \arrow[uu] \\
\text{OBDD} \arrow[uu] \arrow[ur] \arrow[rr, two heads] & & \text{uOBDD} \arrow[uu] \arrow[ur] \arrow[rr, two heads] & & \text{nOBDD} \arrow[uu] \arrow[ur] &
\end{tikzcd}
$$

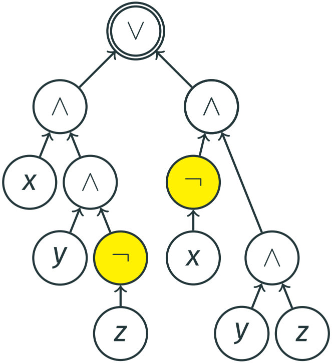
But in **practice** there are solvers for arbitrary circuits:

- **Satisfiability (SAT):** MapleSAT, Cadical, Glucose, etc.
- **Counting:** c2d, d4, dsharp, etc.

A tractable circuit class: d-SDNNF

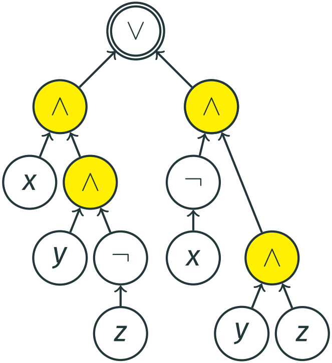


A tractable circuit class: d-SDNNF



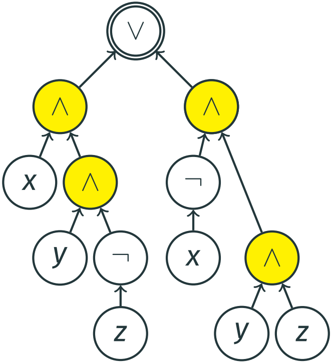
- Negation Normal Form: negations only applied to the leaves

A tractable circuit class: d-SDNNF



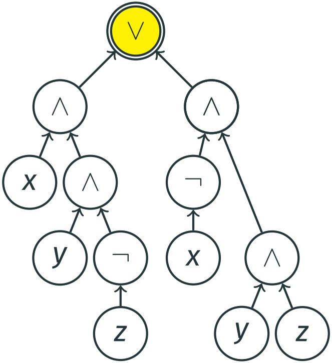
- Negation Normal Form: negations only applied to the leaves
- Decomposable: inputs of $\wedge$-gates are **independent** (no variable has a path to two different inputs of the same $\wedge$-gate)

A tractable circuit class: d-SDNNF



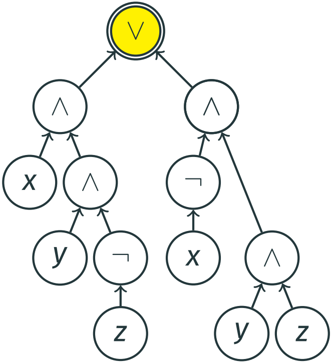
- Negation Normal Form: negations only applied to the leaves
- Decomposable: inputs of ∧-gates are **independent** (no variable has a path to two different inputs of the same ∧-gate)
  - → efficient satisfiability

A tractable circuit class: d-SDNNF



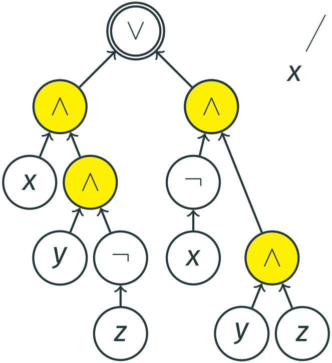
- Negation Normal Form: negations only applied to the leaves
- Decomposable: inputs of $\wedge$-gates are **independent** (no variable has a path to two different inputs of the same $\wedge$-gate)
  - $\rightarrow$ efficient satisfiability
- Deterministic: inputs of $\vee$-gates are **mutually exclusive**

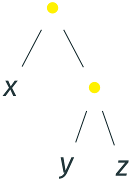
A tractable circuit class: d-SDNNF



- **Negation Normal Form**: negations only applied to the leaves
- **Decomposable**: inputs of $\wedge$-gates are **independent** (no variable has a path to two different inputs of the same $\wedge$-gate)
  - $\rightarrow$ efficient **satisfiability**
- **Deterministic**: inputs of $\vee$-gates are **mutually exclusive**
  - $\rightarrow$ efficient **counting**

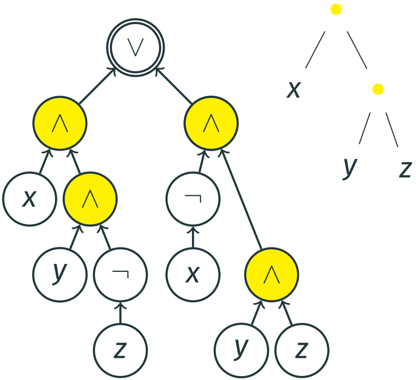
A tractable circuit class: d-SDNNF





- Negation Normal Form: negations only applied to the leaves
- Decomposable: inputs of $\wedge$-gates are **independent** (no variable has a path to two different inputs of the same $\wedge$-gate)
  - $\rightarrow$ efficient satisfiability
- Deterministic: inputs of $\vee$-gates are **mutually exclusive**
  - $\rightarrow$ efficient counting
- Structured: there is a **vtree** that structures the $\wedge$-gates

A tractable circuit class: d-SDNNF



- **Negation Normal Form**: negations only applied to the leaves
- **Decomposable**: inputs of $\wedge$-gates are **independent** (no variable has a path to two different inputs of the same $\wedge$-gate)
  - $\rightarrow$ efficient **satisfiability**
- **Deterministic**: inputs of $\vee$-gates are **mutually exclusive**
  - $\rightarrow$ efficient **counting**
- **Structured**: there is a **vtree** that structures the $\wedge$-gates
  - $\rightarrow$ efficient **enumeration**

Computing lineages in tractable classes

- Self-join-Free Conjunctive Queries (SFCQs)

Theorem [Jha and Suciu, 2013]

For any safe SFCQ, given an instance, we can construct an OBDD representation of the lineage in polynomial time.

Computing lineages in tractable classes

- Self-join-Free Conjunctive Queries (SFCQs)

Theorem [Jha and Suciu, 2013]

For any safe SFCQ, given an instance, we can construct an OBDD representation of the lineage in polynomial time.

- Monadic Second Order queries (MSO)

Theorem [Amarilli et al., 2015]

For any MSO query, given a tree (or word), we can construct a d-SDNNF representation (or OBDD) of the lineage in linear time.

Computing lineages in tractable classes

- Self-join-Free Conjunctive Queries (SFCQs)

Theorem [Jha and Suciu, 2013]

For any safe SFCQ, given an instance, we can construct an OBDD representation of the lineage in polynomial time.

- Monadic Second Order queries (MSO)

Theorem [Amarilli et al., 2015]

For any MSO query, given a tree (or word), we can construct a d-SDNNF representation (or OBDD) of the lineage in linear time.

- Unions of Conjunctive Queries (UCQ)

Conjecture (see [Monet, 2020])

For any safe UCQ, given an instance, we can construct a d-D representation of the lineage in polynomial time.

Computing lineages in tractable classes

- Self-join-Free Conjunctive Queries (SFCQs)

Theorem [Jha and Suciu, 2013]

For any safe SFCQ, given an instance, we can construct an OBDD representation of the lineage in polynomial time.

- Monadic Second Order queries (MSO)

Theorem [Amarilli et al., 2015]

For any MSO query, given a tree (or word), we can construct a d-SDNNF representation (or OBDD) of the lineage in linear time.

- Unions of Conjunctive Queries (UCQ)

Conjecture (see [Monet, 2020])

For any safe UCQ, given an instance, we can construct a d-D representation of the lineage in polynomial time.

Thanks for your attention!

References i

Amarilli, A., Bourhis, P., and Senellart, P. (2015).
Provenance Circuits for Trees and Treelike Instances.
In *ICALP*.

Deutch, D., Milo, T., Roy, S., and Tannen, V. (2014).
Circuits for Datalog Provenance.
In *ICDT*.

Green, T. J., Karvounarakis, G., and Tannen, V. (2007).
Provenance semirings.
In *PODS*.

References ii

Jha, A. and Suciu, D. (2013).
Knowledge compilation meets database theory: compiling queries to decision diagrams.
Theory of Computing Systems, 52(3).

Monet, M. (2020).
Solving a Special Case of the Intensional vs Extensional Conjecture in Probabilistic Databases.
In *PODS*.
To appear.