Query Evaluation on Probabilistic Data
A Story of Dichotomies

Antoine Amarilli

April 4th, 2022

Télécom Paris

Table of contents

Uncertain data management

Relational databases manage data, represented here as a labeled graph

Uncertain data management

Relational databases manage data, represented here as a labeled graph

WorksAt	
Antoine	Télécom Paris
Antoine	Paris Sud
Benny	Paris Sud
Benny	Technion
İsmail	U. Oxford

Uncertain data management

Relational databases manage data, represented here as a labeled graph

WorksAt	
Antoine	Télécom Paris
Antoine	Paris Sud
Benny	Paris Sud
Benny	Technion
İsmail	U. Oxford

MemberOf	
Télécom Paris	ParisTech
Télécom Paris	IP Paris
Paris Sud	IP Paris
Paris Sud	Paris-Saclay
Technion	CESAER

Uncertain data management

Relational databases manage **data**, represented here as a **labeled graph**

WorksAt	
Antoine	Télécom Paris
Antoine	Paris Sud
Benny	Paris Sud
Benny	Technion
İsmail	U. Oxford

MemberOf	
Télécom Paris	ParisTech
Télécom Paris	IP Paris
Paris Sud	IP Paris
Paris Sud	Paris-Saclay
Technion	CESAER

A. Télécom Paris ParisTech

Paris Sud IP Paris

B.

Technion Paris-Saclay

İ. U. Oxford CESAER

Uncertain data management

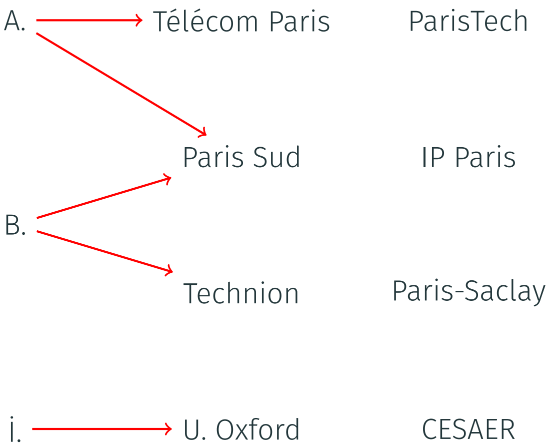
Relational databases manage **data**, represented here as a **labeled graph**

WorksAt

Antoine	Télécom Paris
Antoine	Paris Sud
Benny	Paris Sud
Benny	Technion
İsmail	U. Oxford

MemberOf

Télécom Paris	ParisTech
Télécom Paris	IP Paris
Paris Sud	IP Paris
Paris Sud	Paris-Saclay
Technion	CESAER



Uncertain data management

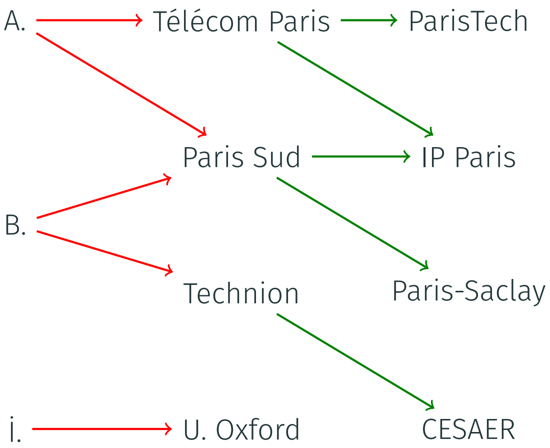
Relational databases manage **data**, represented here as a **labeled graph**

WorksAt

Antoine	Télécom Paris
Antoine	Paris Sud
Benny	Paris Sud
Benny	Technion
İsmail	U. Oxford

MemberOf

Télécom Paris	ParisTech
Télécom Paris	IP Paris
Paris Sud	IP Paris
Paris Sud	Paris-Saclay
Technion	CESAER

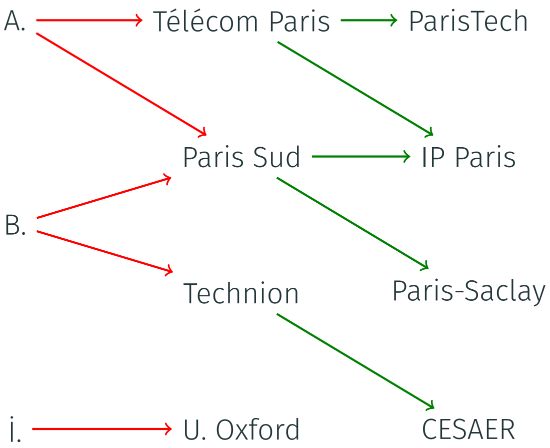


Uncertain data management

Relational databases manage **data**, represented here as a **labeled graph**

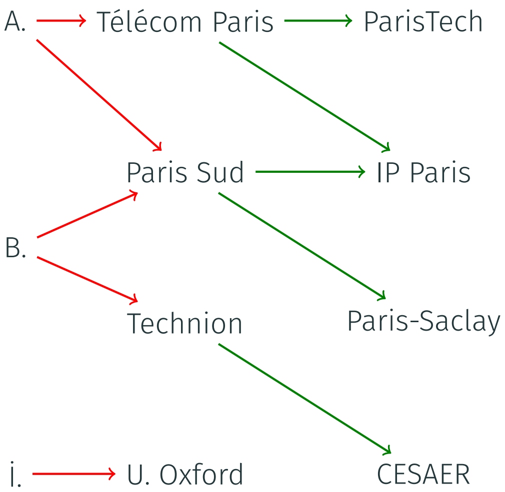
WorksAt	
Antoine	Télécom Paris
Antoine	Paris Sud
Benny	Paris Sud
Benny	Technion
İsmail	U. Oxford

MemberOf	
Télécom Paris	ParisTech
Télécom Paris	IP Paris
Paris Sud	IP Paris
Paris Sud	Paris-Saclay
Technion	CESAER



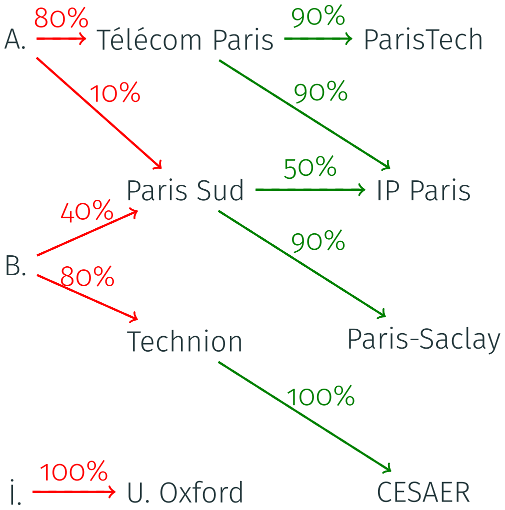
→ **Problem:** we are not **certain** about the true state of the data

Uncertain data model



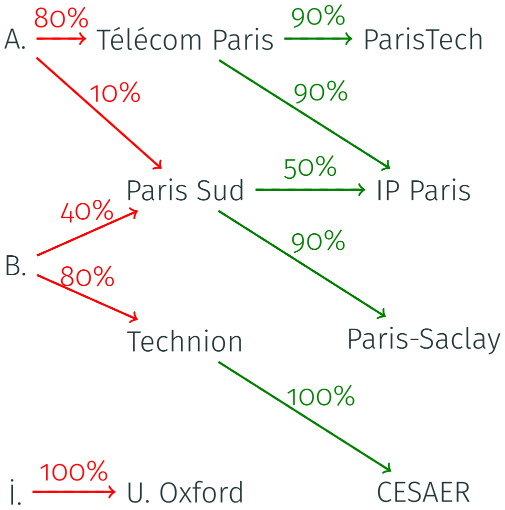
- Uncertain data model: **TID**, for **tuple-independent database**
- Each fact (edge) carries a **probability**

Uncertain data model



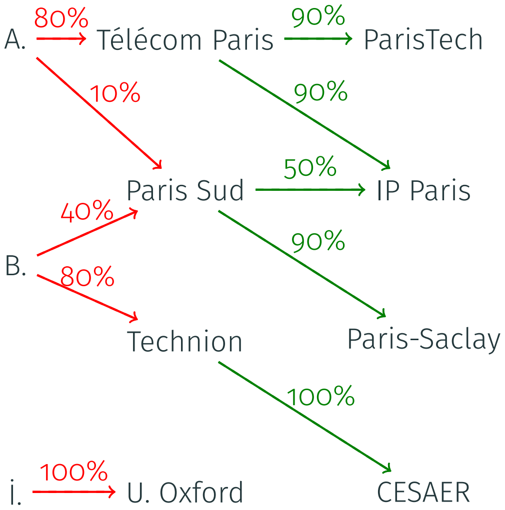
- Uncertain data model: **TID**, for **tuple-independent database**
- Each fact (edge) carries a **probability**

Uncertain data model



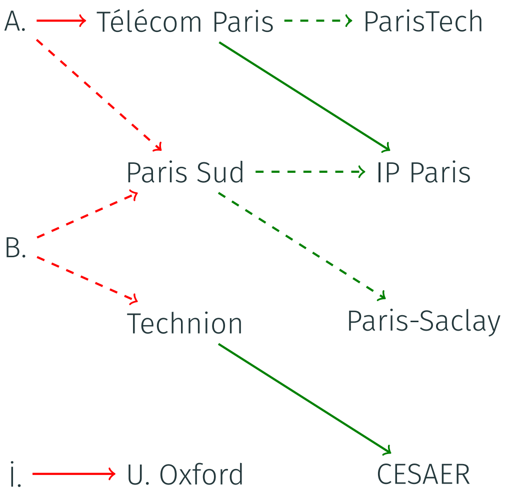
- Uncertain data model: **TID**, for **tuple-independent database**
- Each fact (edge) carries a **probability**
- Each fact exists with its given **probability**
- All facts are **independent**

Uncertain data model



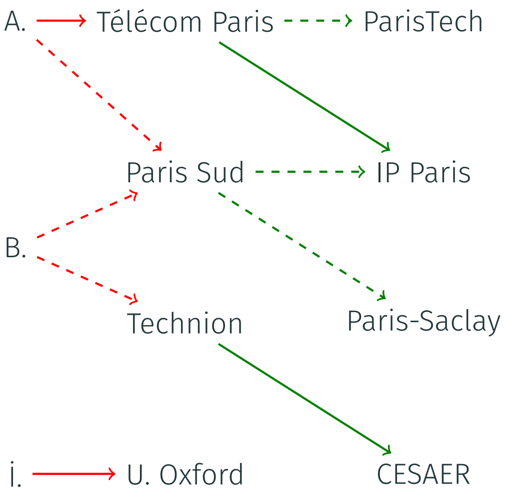
- Uncertain data model: **TID**, for **tuple-independent database**
- Each fact (edge) carries a **probability**
- Each fact exists with its given **probability**
- All facts are **independent**
- **Possible world *W*:** subset of facts

Uncertain data model



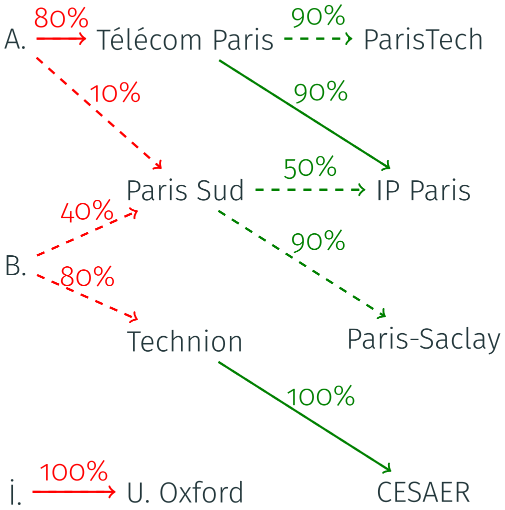
- Uncertain data model: **TID**, for **tuple-independent database**
- Each fact (edge) carries a **probability**
- Each fact exists with its given **probability**
- All facts are **independent**
- **Possible world *W*:** subset of facts

Uncertain data model



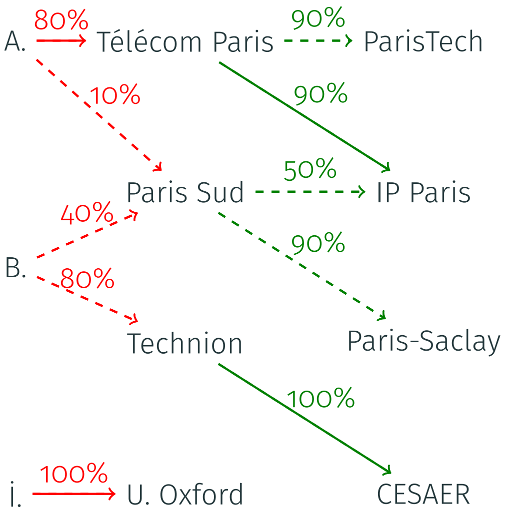
- Uncertain data model: **TID**, for **tuple-independent database**
- Each fact (edge) carries a **probability**
- Each fact exists with its given **probability**
- All facts are **independent**
- **Possible world *W***: subset of facts
- What is the **probability** of this possible world?

Uncertain data model



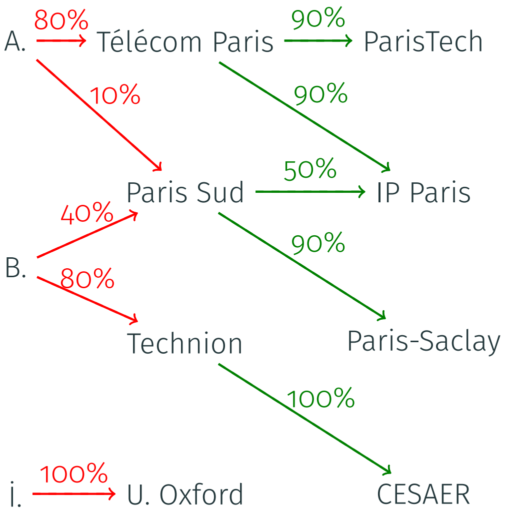
- Uncertain data model: **TID**, for **tuple-independent database**
- Each fact (edge) carries a **probability**
- Each fact exists with its given **probability**
- All facts are **independent**
- **Possible world *W*:** subset of facts
- What is the **probability** of this possible world?

Uncertain data model



- Uncertain data model: **TID**, for **tuple-independent database**
- Each fact (edge) carries a **probability**
- Each fact exists with its given **probability**
- All facts are **independent**
- **Possible world** ***W*****:** subset of facts
- What is the **probability** of this possible world? **0.03%**

Uncertain data model



- Uncertain data model: **TID**, for **tuple-independent database**
- Each fact (edge) carries a **probability**
- Each fact exists with its given **probability**
- All facts are **independent**
- **Possible world W:** subset of facts
- What is the **probability** of this possible world? **0.03%**

$$\Pr(W) = \left(\prod_{F \in W} \Pr(F)\right) \times \left(\prod_{F \notin W} (1 - \Pr(F))\right)$$

Queries

A central task in databases is to **evaluate queries**

Queries

A central task in databases is to **evaluate queries**

- **Query:** maps a graph (**without probabilities**) to YES/NO

Queries

A central task in databases is to **evaluate queries**

- **Query:** maps a graph (**without probabilities**) to YES/NO
- **Conjunctive query** (CQ): can I find a match of a **pattern**?
  - e.g., $\exists x\, y\, z \quad x \longrightarrow y \longrightarrow z$

Queries

A central task in databases is to **evaluate queries**

- **Query:** maps a graph (**without probabilities**) to YES/NO
- **Conjunctive query** (CQ): can I find a match of a **pattern**?
  - e.g., $\exists x\, y\, z \quad x \longrightarrow y \longrightarrow z$

$\rightarrow$ We want a **homomorphism** from the pattern to the graph (not necessarily **injective**)

$\rightarrow$ Formally: an **existentially quantified conjunction of atoms (edges)**

Queries

A central task in databases is to **evaluate queries**

- **Query:** maps a graph (**without probabilities**) to YES/NO
- **Conjunctive query** (CQ): can I find a match of a **pattern**?
    - e.g., $\exists x\, y\, z \quad x \longrightarrow y \longrightarrow z$

  → We want a **homomorphism** from the pattern to the graph (not necessarily **injective**)

  → Formally: an **existentially quantified conjunction of atoms (edges)**
- **Union of conjunctive queries** (UCQ): can I find a match of **some pattern**?

  → e.g., $(\exists x\, y\, z \quad x \longrightarrow y \longrightarrow z \;) \lor (\exists x\, y\, z\, w \quad x \longrightarrow y \quad z \longrightarrow w \;)$

Queries

A central task in databases is to **evaluate queries**

- **Query:** maps a graph (**without probabilities**) to YES/NO
- **Conjunctive query** (CQ): can I find a match of a **pattern**?
  - e.g., $\exists x\, y\, z \quad x \longrightarrow y \longrightarrow z$

  → We want a **homomorphism** from the pattern to the graph (not necessarily **injective**)

  → Formally: an **existentially quantified conjunction of atoms (edges)**
- **Union of conjunctive queries** (UCQ): can I find a match of **some pattern**?

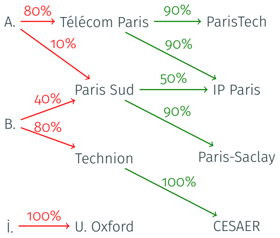
  → e.g., $(\exists x\, y\, z \quad x \longrightarrow y \longrightarrow z \;) \vee (\exists x\, y\, z\, w \quad x \longrightarrow y \quad z \longrightarrow w \;)$
  - Formally: a **finite disjunction** of CQs

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$

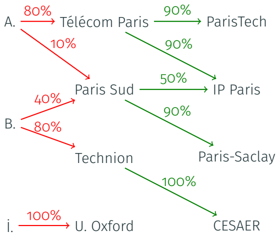
Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$
- The **input** is a TID D:



Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \xrightarrow{\quad} y \xrightarrow{\quad} z$
- The **input** is a TID D:

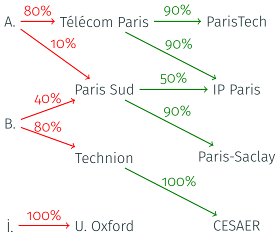


- The **output** is the **total probability** of the worlds which satisfy the query:

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \xrightarrow{\quad\quad} y \xrightarrow{\quad\quad} z$

- The **input** is a TID D:

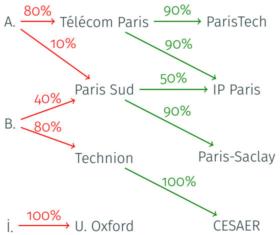


- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\rightarrow$ **Intuition:** the **probability** that the query is true

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \xrightarrow{} y \xrightarrow{} z$
- The **input** is a TID D:

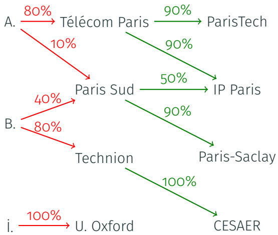


- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\rightarrow$ **Intuition:** the **probability** that the query is true
- We can always compute the probability in exponential time (go over all possibilities)

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$
- The **input** is a TID D:

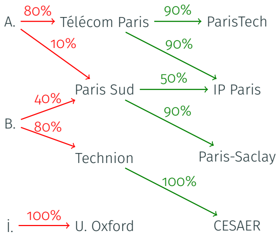


- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\rightarrow$ **Intuition:** the **probability** that the query is true
- We can always compute the probability in exponential time (go over all possibilities)
- Here we can do better (in PTIME):

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$
- The **input** is a TID D:

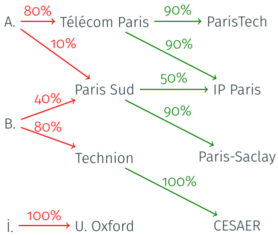


- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\rightarrow$ **Intuition:** the **probability** that the query is true
- We can always compute the probability in exponential time (go over all possibilities)
- Here we can do better (in PTIME): $1 -$

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$
- The **input** is a TID D:

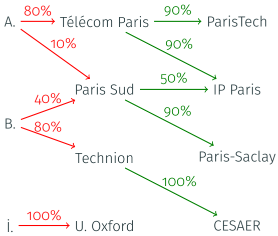


- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\rightarrow$ **Intuition:** the **probability** that the query is true
- We can always compute the probability in exponential time (go over all possibilities)
- Here we can do better (in PTIME): $1 - (1 - 80\%) \times$

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$
- The **input** is a TID D:

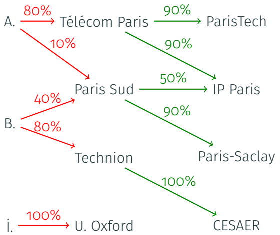


- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\rightarrow$ **Intuition:** the **probability** that the query is true
- We can always compute the probability in exponential time (go over all possibilities)
- Here we can do better (in PTIME): $1 - (1 - 80\%) \times (1 -$

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$
- The **input** is a TID D:

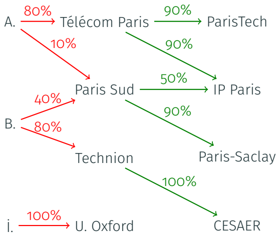


- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\rightarrow$ **Intuition:** the **probability** that the query is true
- We can always compute the probability in exponential time (go over all possibilities)
- Here we can do better (in PTIME): $1 - (1 - 80\%) \times (1 - (\qquad) \times (\qquad)$

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$
- The **input** is a TID D:

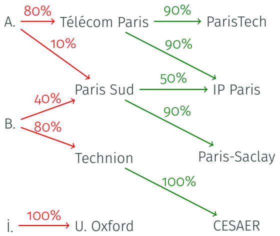


- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\rightarrow$ **Intuition:** the **probability** that the query is true
- We can always compute the probability in exponential time (go over all possibilities)
- Here we can do better (in PTIME): $1 - (1 - 80\%) \times (1 - (1 -$ $\quad$ $) \times$ ($\quad$)

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$
- The **input** is a TID D:

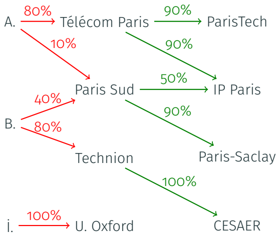


- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\rightarrow$ **Intuition:** the **probability** that the query is true
- We can always compute the probability in exponential time (go over all possibilities)
- Here we can do better (in PTIME): $1 - (1 - 80\%) \times (1 - (1 - (1 - 10\%) \times (1 - 40\%)) \times$ $(\qquad)$

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$
- The **input** is a TID D:

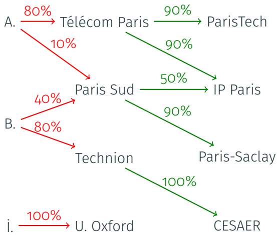


- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\to$ **Intuition:** the **probability** that the query is true
- We can always compute the probability in exponential time (go over all possibilities)
- Here we can do better (in PTIME): $1 - (1 - 80\%) \times (1 - (1 - (1 - 10\%) \times (1 - 40\%)) \times (1 - (1 - 50\%) \times (1 - 90\%)))$

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$
- The **input** is a TID D:

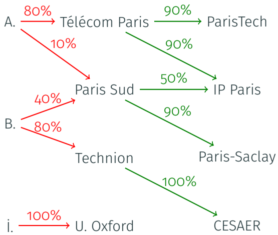


- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\rightarrow$ **Intuition:** the **probability** that the query is true
- We can always compute the probability in exponential time (go over all possibilities)
- Here we can do better (in PTIME): $1 - (1 - 80\%) \times (1 - (1 - (1 - 10\%) \times (1 - 40\%)) \times (1 - (1 - 50\%) \times (1 - 90\%))) \times (1 - 80\% \times (1 - (1 - 90\%) \times (1 - 90\%)))$,

Problem statement: Probabilistic query evaluation (PQE)

- We **fix** a query Q, for instance the CQ: $x \longrightarrow y \longrightarrow z$
- The **input** is a TID D:



- The **output** is the **total probability** of the worlds which satisfy the query:
  - Formally: $\sum_{W \subseteq D, W \models Q} \Pr(W)$

  $\rightarrow$ **Intuition:** the **probability** that the query is true
- We can always compute the probability in exponential time (go over all possibilities)
- Here we can do better (in PTIME): $1 - (1 - 80\%) \times (1 - (1 - (1 - 10\%) \times (1 - 40\%)) \times (1 - (1 - 50\%) \times (1 - 90\%))) \times (1 - 80\% \times (1 - (1 - 90\%) \times (1 - 90\%)))$, i.e., 97.65792%

Research goal: Understanding the complexity of PQE

What is the complexity of $\mathrm{PQE}(Q)$ depending on the query Q?

Research goal: Understanding the complexity of PQE

What is the complexity of $\mathrm{PQE}(Q)$ depending on the query Q?

$\rightarrow$ Note that we study **data complexity**, i.e., Q is **fixed** and the input is the **data**

Research goal: Understanding the complexity of PQE

What is the complexity of $\mathrm{PQE}(Q)$ depending on the query Q?

$\rightarrow$ Note that we study **data complexity**, i.e., Q is **fixed** and the input is the **data**

In this talk: several dichotomies on the PQE problem:

Research goal: Understanding the complexity of PQE

What is the complexity of $\mathrm{PQE}(Q)$ depending on the query Q?

$\rightarrow$ Note that we study **data complexity**, i.e., Q is **fixed** and the input is the **data**

In this talk: several dichotomies on the PQE problem:

- **Existing results**:
  - $\mathrm{PQE}(Q)$ is in #P for any UCQ Q and is **#P-hard** for some CQs
  - **Dichotomy** by Dalvi and Suciu: $\mathrm{PQE}(Q)$ for a UCQ Q is either **#P-hard** or **PTIME**

Research goal: Understanding the complexity of PQE

What is the complexity of $\mathrm{PQE}(Q)$ depending on the query Q?

$\rightarrow$ Note that we study **data complexity**, i.e., Q is **fixed** and the input is the **data**

In this talk: several dichotomies on the PQE problem:

- **Existing results**:
  - $\mathrm{PQE}(Q)$ is in #P for any UCQ Q and is **#P-hard** for some CQs
  - **Dichotomy** by Dalvi and Suciu: $\mathrm{PQE}(Q)$ for a UCQ Q is either **#P-hard** or **PTIME**
- **More general queries**: dichotomy on **homomorphism-closed queries**
  - $\mathrm{PQE}(Q)$ is **#P-hard** for all homomorphism-closed queries not equivalent to a safe UCQ

Research goal: Understanding the complexity of PQE

What is the complexity of $\mathrm{PQE}(Q)$ depending on the query Q?

$\rightarrow$ Note that we study **data complexity**, i.e., Q is **fixed** and the input is the **data**

In this talk: several dichotomies on the PQE problem:

- **Existing results**:
  - $\mathrm{PQE}(Q)$ is in #P for any UCQ Q and is **#P-hard** for some CQs
  - **Dichotomy** by Dalvi and Suciu: $\mathrm{PQE}(Q)$ for a UCQ Q is either **#P-hard** or **PTIME**
- **More general queries**: dichotomy on **homomorphism-closed queries**
  - $\mathrm{PQE}(Q)$ is **#P-hard** for all homomorphism-closed queries not equivalent to a safe UCQ
- **Restricted instances:** $\mathrm{PQE}(Q)$ for MSO queries...
  - Is **in PTIME** if the input data is restricted to have **bounded treewidth**
  - Is **intractable otherwise** under some assumptions

Research goal: Understanding the complexity of PQE

What is the complexity of $\mathrm{PQE}(Q)$ depending on the query Q?

→ Note that we study **data complexity**, i.e., Q is **fixed** and the input is the **data**

In this talk: several dichotomies on the PQE problem:

- **Existing results**:
  - $\mathrm{PQE}(Q)$ is in #P for any UCQ Q and is **#P-hard** for some CQs
  - **Dichotomy** by Dalvi and Suciu: $\mathrm{PQE}(Q)$ for a UCQ Q is either **#P-hard** or **PTIME**
- **More general queries**: dichotomy on **homomorphism-closed queries**
  - $\mathrm{PQE}(Q)$ is **#P-hard** for all homomorphism-closed queries not equivalent to a safe UCQ
- **Restricted instances:** $\mathrm{PQE}(Q)$ for MSO queries...
  - Is **in PTIME** if the input data is restricted to have **bounded treewidth**
  - Is **intractable otherwise** under some assumptions
- **Restricted instances:** if all probabilities are **50%** then the complexity is the same

Table of contents

Basic complexity results

- Whenever we can evaluate Q in PTIME, then $\mathrm{PQE}(Q)$ is in #P

Basic complexity results

- Whenever we can evaluate Q in PTIME, then $\mathrm{PQE}(Q)$ is in #P
  - #P: counting class of problems expressible as the number of accepting paths of a nondeterministic polynomial-time Turing Machine

$\rightarrow$ Nondeterministically guess a possible world, then test the query

$\rightarrow$ In particular, $\mathrm{PQE}(Q)$ is in #P for any UCQ Q

Basic complexity results

- Whenever we can evaluate Q in PTIME, then $\mathrm{PQE}(Q)$ is in #P
  - #P: counting class of problems expressible as the number of accepting paths of a nondeterministic polynomial-time Turing Machine

  $\rightarrow$ Nondeterministically guess a possible world, then test the query

  $\rightarrow$ In particular, $\mathrm{PQE}(Q)$ is in #P for any UCQ Q

- For some queries Q, the task $\mathrm{PQE}(Q)$ is in PTIME

Basic complexity results

- Whenever we can **evaluate Q in PTIME**, then $\mathrm{PQE}(Q)$ is in **#P**
  - **#P:** counting class of problems expressible as the **number of accepting paths** of a nondeterministic polynomial-time Turing Machine

  $\rightarrow$ Nondeterministically guess a possible world, then test the query

  $\rightarrow$ In particular, $\mathrm{PQE}(Q)$ is in **#P** for any **UCQ** Q

- For some queries Q, the task $\mathrm{PQE}(Q)$ is in **PTIME**

  $\rightarrow$ e.g., single-atom CQs

  $\rightarrow$ e.g., $x \longrightarrow y \longrightarrow z$

PQE is sometimes #P-hard

Let us show that $\mathrm{PQE}(Q)$ is **#P-hard** for the CQ Q :

PQE is sometimes #P-hard

Let us show that $\mathrm{PQE}(Q)$ is **#P-hard** for the CQ Q : $x \longrightarrow y \longrightarrow z \longrightarrow w$

PQE is sometimes #P-hard

Let us show that $\mathrm{PQE}(Q)$ is **#P-hard** for the CQ Q : $x \longrightarrow y \longrightarrow z \longrightarrow w$

- Reduce from the problem of **counting satisfying valuations** of a Boolean formula
  - e.g., given $(x \lor y) \land z$, compute that it has 3 satisfying valuations

PQE is sometimes #P-hard

Let us show that $\mathrm{PQE}(Q)$ is **#P-hard** for the CQ Q : $x \longrightarrow y \longrightarrow z \longrightarrow w$

- Reduce from the problem of **counting satisfying valuations** of a Boolean formula
  - e.g., given $(x \lor y) \land z$, compute that it has 3 satisfying valuations
- This problem is already **#P-hard** for so-called **PP2DNF formulas**:

PQE is sometimes #P-hard

Let us show that $\mathrm{PQE}(Q)$ is **#P-hard** for the CQ Q : $x \longrightarrow y \longrightarrow z \longrightarrow w$

- Reduce from the problem of **counting satisfying valuations** of a Boolean formula
  - e.g., given $(x \lor y) \land z$, compute that it has 3 satisfying valuations
- This problem is already **#P-hard** for so-called **PP2DNF formulas**:
  - **Positive** (no negation) and **Partitioned variables**: $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$

PQE is sometimes #P-hard

Let us show that $\mathrm{PQE}(Q)$ is **#P-hard** for the CQ Q : $x \longrightarrow y \longrightarrow z \longrightarrow w$

- Reduce from the problem of **counting satisfying valuations** of a Boolean formula
  - e.g., given $(x \lor y) \land z$, compute that it has 3 satisfying valuations
- This problem is already **#P-hard** for so-called **PP2DNF formulas**:
  - **Positive** (no negation) and **Partitioned variables**: $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$
  - **2-DNF**: disjunction of clauses like $X_i \land Y_j$

PQE is sometimes #P-hard

Let us show that $\mathrm{PQE}(Q)$ is **#P-hard** for the CQ Q : $x \longrightarrow y \longrightarrow z \longrightarrow w$

- Reduce from the problem of **counting satisfying valuations** of a Boolean formula
  - e.g., given $(x \vee y) \wedge z$, compute that it has 3 satisfying valuations
- This problem is already **#P-hard** for so-called **PP2DNF formulas**:
  - **Positive** (no negation) and **Partitioned variables**: $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$
  - **2-DNF**: disjunction of clauses like $X_i \wedge Y_j$

- Example: $\phi : (X_1 \wedge Y_1) \vee (X_1 \wedge Y_2) \vee (X_2 \wedge Y_2) \vee (X_3 \wedge Y_1) \vee (X_3 \wedge Y_2)$

PQE is sometimes #P-hard

Let us show that $\mathrm{PQE}(Q)$ is **#P-hard** for the CQ Q : $x \longrightarrow y \longrightarrow z \longrightarrow w$

- Reduce from the problem of **counting satisfying valuations** of a Boolean formula
  - e.g., given $(x \vee y) \wedge z$, compute that it has 3 satisfying valuations
- This problem is already **#P-hard** for so-called **PP2DNF formulas**:
  - **Positive** (no negation) and **Partitioned variables**: $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$
  - **2-DNF**: disjunction of clauses like $X_i \wedge Y_j$

- Example: $\phi : (X_1 \wedge Y_1) \vee (X_1 \wedge Y_2) \vee (X_2 \wedge Y_2) \vee (X_3 \wedge Y_1) \vee (X_3 \wedge Y_2)$

$$a'_1 \xrightarrow{1/2} a_1$$

$$a'_2 \xrightarrow{1/2} a_2$$

$$a'_3 \xrightarrow{1/2} a_3$$

PQE is sometimes #P-hard

Let us show that $\mathrm{PQE}(Q)$ is **#P-hard** for the CQ Q : $x \longrightarrow y \longrightarrow z \longrightarrow w$

- Reduce from the problem of **counting satisfying valuations** of a Boolean formula
  - e.g., given $(x \lor y) \land z$, compute that it has 3 satisfying valuations
- This problem is already **#P-hard** for so-called **PP2DNF formulas**:
  - **Positive** (no negation) and **Partitioned variables**: $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$
  - **2-DNF**: disjunction of clauses like $X_i \land Y_j$

- Example: $\phi : (X_1 \land Y_1) \lor (X_1 \land Y_2) \lor (X_2 \land Y_2) \lor (X_3 \land Y_1) \lor (X_3 \land Y_2)$

$$a'_1 \xrightarrow{1/2} a_1 \qquad\qquad b_1 \xrightarrow{1/2} b'_1$$

$$a'_2 \xrightarrow{1/2} a_2$$

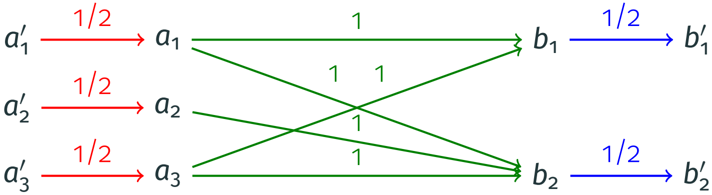
$$a'_3 \xrightarrow{1/2} a_3 \qquad\qquad b_2 \xrightarrow{1/2} b'_2$$

PQE is sometimes #P-hard

Let us show that $\mathrm{PQE}(Q)$ is **#P-hard** for the CQ Q : $x \xrightarrow{\quad} y \xrightarrow{\quad} z \xrightarrow{\quad} w$

- Reduce from the problem of **counting satisfying valuations** of a Boolean formula
  - e.g., given $(x \vee y) \wedge z$, compute that it has 3 satisfying valuations
- This problem is already **#P-hard** for so-called **PP2DNF formulas**:
  - **Positive** (no negation) and **Partitioned variables**: $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$
  - **2-DNF**: disjunction of clauses like $X_i \wedge Y_j$

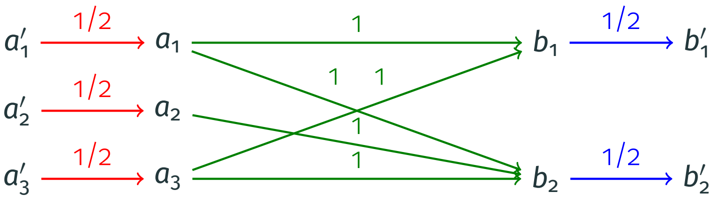
- Example: $\phi : (X_1 \wedge Y_1) \vee (X_1 \wedge Y_2) \vee (X_2 \wedge Y_2) \vee (X_3 \wedge Y_1) \vee (X_3 \wedge Y_2)$



PQE is sometimes #P-hard

Let us show that $\mathrm{PQE}(Q)$ is **#P-hard** for the CQ Q : $x \longrightarrow y \longrightarrow z \longrightarrow w$

- Reduce from the problem of **counting satisfying valuations** of a Boolean formula
  - e.g., given $(x \vee y) \wedge z$, compute that it has 3 satisfying valuations
- This problem is already **#P-hard** for so-called **PP2DNF formulas**:
  - **Positive** (no negation) and **Partitioned variables**: $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_m$
  - **2-DNF**: disjunction of clauses like $X_i \wedge Y_j$
- Example: $\phi : (X_1 \wedge Y_1) \vee (X_1 \wedge Y_2) \vee (X_2 \wedge Y_2) \vee (X_3 \wedge Y_1) \vee (X_3 \wedge Y_2)$



Idea: Satisfying valuations of ϕ correspond to **possible worlds** with a **match** of Q

The “small” Dalvi and Suciu dichotomy

- Self-join-free CQ: only one edge of each color (no repeated color)

The “small” Dalvi and Suciu dichotomy

- Self-join-free CQ: only one edge of each color (no repeated color)

Theorem (Dalvi and Suciu, see Dalvi and Suciu 2007)

Let Q be a self-join-free CQ:

- *If Q is a star, then $\mathrm{PQE}(Q)$ is in PTIME*
- *Otherwise, $\mathrm{PQE}(Q)$ is #P-hard*

The “small” Dalvi and Suciu dichotomy

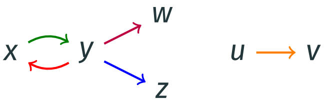
- Self-join-free CQ: only one edge of each color (no repeated color)

Theorem (Dalvi and Suciu, see Dalvi and Suciu 2007)

Let Q be a self-join-free CQ:

- *If Q is a star, then $\mathrm{PQE}(Q)$ is in PTIME*
- *Otherwise, $\mathrm{PQE}(Q)$ is #P-hard*

- A star is a CQ where each connected component has a separator variable that occurs in every edge of the component



The “small” Dalvi and Suciu dichotomy

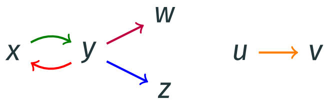
- **Self-join-free CQ**: only one edge of each color (no repeated color)

Theorem (Dalvi and Suciu, see Dalvi and Suciu 2007)

Let Q be a self-join-free CQ:

- *If Q is a star, then $\mathrm{PQE}(Q)$ is in PTIME*
- *Otherwise, $\mathrm{PQE}(Q)$ is #P-hard*

- A **star** is a CQ where each connected component has a **separator variable** that occurs in every edge of the component



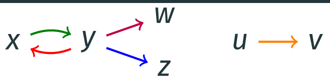
- The dichotomy generalizes to **higher-arity data** (hierarchical queries)

Proving the small dichotomy (upper bound)

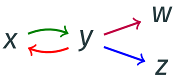
$$\begin{tikzcd} & & w \\ x \arrow[r, bend left] & y \arrow[l, bend left] \arrow[ur] \arrow[dr] & \\ & & z \end{tikzcd} \qquad u \longrightarrow v$$

How to solve $\mathrm{PQE}(Q)$ for Q a self-join-free star?

Proving the small dichotomy (upper bound)



How to solve $\mathrm{PQE}(Q)$ for Q a self-join-free star?



- We consider each connected component separately

$\rightarrow$ **Independent conjunction** over the connected components

Proving the small dichotomy (upper bound)

$$\begin{tikzcd} & & w \\ x \arrow[r, bend left] & y \arrow[l, bend left] \arrow[ur] \arrow[dr] & \\ & & z \end{tikzcd} \qquad u \longrightarrow v$$

How to solve $\mathrm{PQE}(Q)$ for Q a self-join-free star?

$$\begin{tikzcd} & & w \\ x \arrow[r, bend left] & y \arrow[l, bend left] \arrow[ur] \arrow[dr] & \\ & & z \end{tikzcd}$$

- We consider each connected component separately

$\rightarrow$ Independent conjunction over the connected components

$$\begin{tikzcd} & & w \\ x \arrow[r, bend left] & \boldsymbol{a} \arrow[l, bend left] \arrow[ur] \arrow[dr] & \\ & & z \end{tikzcd}$$

- We can test all possible values of the separator variable

$\rightarrow$ Independent disjunction over the values of the separator

Proving the small dichotomy (upper bound)

$$\begin{tikzcd} & & w \\ x \arrow[r, bend left] & y \arrow[l, bend left] \arrow[ur] \arrow[dr] & \\ & & z \end{tikzcd} \qquad u \longrightarrow v$$

How to solve $\mathrm{PQE}(\boldsymbol{Q})$ for $\boldsymbol{Q}$ a self-join-free star?

$$\begin{tikzcd} & & w \\ x \arrow[r, bend left] & y \arrow[l, bend left] \arrow[ur] \arrow[dr] & \\ & & z \end{tikzcd}$$

- We consider each connected component separately

$\rightarrow$ **Independent conjunction** over the connected components

$$\begin{tikzcd} & & w \\ x \arrow[r, bend left] & \boldsymbol{a} \arrow[l, bend left] \arrow[ur] \arrow[dr] & \\ & & z \end{tikzcd}$$

- We can test all possible values of the **separator variable**

$\rightarrow$ **Independent disjunction** over the values of the separator

$$\begin{tikzcd} x \arrow[r, bend left] & \boldsymbol{a} \arrow[l, bend left] \end{tikzcd}$$

- For every match, we consider every **other variable** separately

$\rightarrow$ **Independent conjunction** over the variables

Proving the small dichotomy (upper bound)

$$\begin{tikzcd} & & w \\ x \arrow[r, bend left] & y \arrow[l, bend left] \arrow[ur] \arrow[dr] & \\ & & z \end{tikzcd} \qquad u \longrightarrow v$$

How to solve $\mathrm{PQE}(Q)$ for Q a self-join-free star?

$$\begin{tikzcd} & & w \\ x \arrow[r, bend left] & y \arrow[l, bend left] \arrow[ur] \arrow[dr] & \\ & & z \end{tikzcd}$$

- We consider each connected component separately

$\rightarrow$ **Independent conjunction** over the connected components

$$\begin{tikzcd} & & w \\ x \arrow[r, bend left] & \boldsymbol{a} \arrow[l, bend left] \arrow[ur] \arrow[dr] & \\ & & z \end{tikzcd}$$

- We can test all possible values of the **separator variable**

$\rightarrow$ **Independent disjunction** over the values of the separator

$$\begin{tikzcd} x \arrow[r, bend left] & \boldsymbol{a} \arrow[l, bend left] \end{tikzcd}$$

- For every match, we consider every **other variable** separately

$\rightarrow$ **Independent conjunction** over the variables

$$\begin{tikzcd} \boldsymbol{b} \arrow[r, bend left] & \boldsymbol{a} \arrow[l, bend left] \end{tikzcd}$$

- We consider every value for the **other variable**

$\rightarrow$ **Independent disjunction** over the possible assignments

$\rightarrow$ **Independent conjunction** over the facts

Proving the small dichotomy (lower bound)

Every non-star self-join-free CQ contains a pattern essentially like:

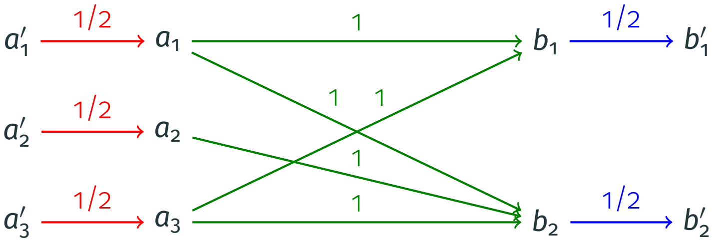
$$x \longrightarrow y \longrightarrow z \longrightarrow w$$

Proving the small dichotomy (lower bound)

Every **non-star** self-join-free CQ contains a pattern essentially like:

$$x \longrightarrow y \longrightarrow z \longrightarrow w$$

We can use this to reduce from #SAT like before:



The "big" Dalvi and Suciu dichotomy

Full dichotomy on the **unions of conjunctive queries** (UCQs):

Theorem (Dalvi and Suciu 2012)

Let Q be a UCQ:

- *If Q is handled by a complicated algorithm $\mathrm{PQE}(Q)$ is in **PTIME***
- *Otherwise, $\mathrm{PQE}(Q)$ is **#P-hard***

The "big" Dalvi and Suciu dichotomy

Full dichotomy on the **unions of conjunctive queries** (UCQs):

Theorem (Dalvi and Suciu 2012)

Let Q be a UCQ:

- *If Q is handled by a complicated algorithm $\mathrm{PQE}(Q)$ is in **PTIME***
- *Otherwise, $\mathrm{PQE}(Q)$ is **#P-hard***

This result is **far more complicated** (but still generalizes to higher arity)

- **Upper bound:**
  - an algorithm generalizing the previous case with **inclusion-exclusion**
  - many **unpleasant details** (e.g., a ranking transformation)
- **Lower bound:** hardness proof on minimal cases where the algorithm does not work

Table of contents

Going to more general queries

*The case of **UCQs** is settled! but what about **more expressive queries**?*

Going to more general queries

*The case of **UCQs** is settled! but what about **more expressive queries**?*

- Work by Fink and Olteanu 2016 about **negation**
- Some work on **ontology-mediated query answering** (Jung and Lutz 2012)

Going to more general queries

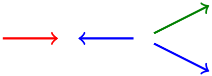
*The case of **UCQs** is settled! but what about **more expressive queries**?*

- Work by Fink and Olteanu 2016 about **negation**
- Some work on **ontology-mediated query answering** (Jung and Lutz 2012)

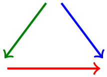
We study the case of **queries closed under homomorphisms**

Homomorphism-closed queries

- A homomorphism from a graph G to a graph G' maps the vertices of G to those of G' while preserving the edges

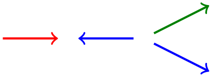
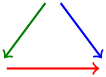


has a homomorphism to



Homomorphism-closed queries

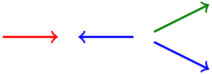
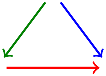
- A homomorphism from a graph G to a graph G' maps the vertices of G to those of G' while preserving the edges

 has a homomorphism to 

- Homomorphism-closed query Q: for any graph G, if G satisfies Q and G has a homomorphism to G' then G' also satisfies Q

Homomorphism-closed queries

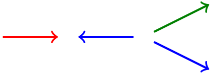
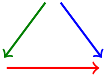
- A **homomorphism** from a graph G to a graph G' maps the vertices of G to those of G' while preserving the edges

 has a homomorphism to 

- **Homomorphism-closed query** Q: for any graph G, if G satisfies Q and G has a homomorphism to G' then G' also satisfies Q
- Homomorphism-closed queries include **all CQs**, **all UCQs**, some **recursive queries** like **regular path queries** (RPQs), **Datalog**, etc.

Homomorphism-closed queries

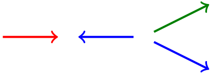
- A **homomorphism** from a graph G to a graph G' maps the vertices of G to those of G' while preserving the edges

 has a homomorphism to 

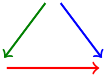
- **Homomorphism-closed query** Q: for any graph G, if G satisfies Q and G has a homomorphism to G' then G' also satisfies Q
- Homomorphism-closed queries include **all CQs**, **all UCQs**, some **recursive queries** like **regular path queries** (RPQs), **Datalog**, etc.
- Queries with **negations** or **inequalities** are not homomorphism-closed

Homomorphism-closed queries

- A **homomorphism** from a graph G to a graph G' maps the vertices of G to those of G' while preserving the edges



has a homomorphism to



- **Homomorphism-closed query** Q: for any graph G, if G satisfies Q and G has a homomorphism to G' then G' also satisfies Q
- Homomorphism-closed queries include **all CQs**, **all UCQs**, some **recursive queries** like **regular path queries** (RPQs), **Datalog**, etc.
- Queries with **negations** or **inequalities** are not homomorphism-closed
- Homomorphism-closed queries can equivalently be seen as **infinite unions of CQs** (corresponding to their models)

Our result

We show:

Theorem (Amarilli and Ceylan 2020)

For any query Q closed under homomorphisms:

- *Either* Q *is equivalent to a tractable UCQ and* $\mathrm{PQE}(Q)$ *is in PTIME*
- *In all other cases,* $\mathrm{PQE}(Q)$ *is #P-hard*

Our result

We show:

Theorem (Amarilli and Ceylan 2020)

For any query Q closed under homomorphisms:

- *Either Q is equivalent to a tractable UCQ and $\mathrm{PQE}(Q)$ is in PTIME*
- *In all other cases, $\mathrm{PQE}(Q)$ is #P-hard*

- The same holds for RPQs, Datalog queries, etc.

Our result

We show:

Theorem (Amarilli and Ceylan 2020)

For any query Q closed under homomorphisms:

- *Either Q is equivalent to a tractable UCQ and $\mathrm{PQE}(Q)$ is in PTIME*
- *In all other cases, $\mathrm{PQE}(Q)$ is #P-hard*

- The same holds for RPQs, Datalog queries, etc.
- Example: the RPQ Q: $\longrightarrow (\longrightarrow)^* \longrightarrow$

Our result

We show:

Theorem (Amarilli and Ceylan 2020)

For any query Q closed under homomorphisms:

- *Either Q is equivalent to a tractable UCQ and $\mathrm{PQE}(Q)$ is in PTIME*
- *In all other cases, $\mathrm{PQE}(Q)$ is #P-hard*

- The same holds for RPQs, Datalog queries, etc.
- Example: the RPQ Q: $\longrightarrow (\longrightarrow)^* \longrightarrow$
  - It is not equivalent to a UCQ: infinite disjunction $\longrightarrow (\longrightarrow)^i \longrightarrow$ for all $i \in \mathbb{N}$

Our result

We show:

Theorem (Amarilli and Ceylan 2020)

For any query Q closed under homomorphisms:

- *Either Q is equivalent to a tractable UCQ and $\mathrm{PQE}(Q)$ is in PTIME*
- *In all other cases, $\mathrm{PQE}(Q)$ is #P-hard*

- The same holds for RPQs, Datalog queries, etc.
- Example: the RPQ Q: $\longrightarrow (\longrightarrow)^* \longrightarrow$
  - It is not equivalent to a UCQ: infinite disjunction $\longrightarrow (\longrightarrow)^i \longrightarrow$ for all $i \in \mathbb{N}$
  - Hence, $\mathrm{PQE}(Q)$ is #P-hard

Basic idea: finding a tight pattern

The challenging part is to show:

Theorem

For any query Q closed under homomorphisms and unbounded, $\mathrm{PQE}(Q)$ is #P-hard

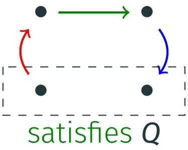
Basic idea: finding a tight pattern

The challenging part is to show:

Theorem

*For any query Q closed under homomorphisms and **unbounded**, $\mathrm{PQE}(Q)$ is **#P-hard***

Idea: find a **tight pattern**, i.e., a graph with three distinguished edges $\rightarrow$ $\rightarrow$ $\rightarrow$ such that:



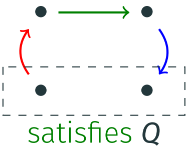
Basic idea: finding a tight pattern

The challenging part is to show:

Theorem

For any query Q closed under homomorphisms and unbounded, $\mathrm{PQE}(Q)$ is #P-hard

Idea: find a tight pattern, i.e., a graph with three distinguished edges $\rightarrow$ $\rightarrow$ $\rightarrow$ such that:



but

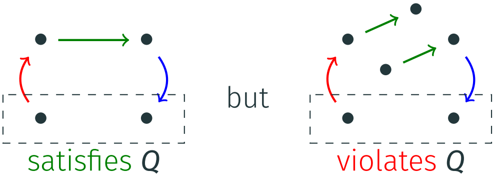
Basic idea: finding a tight pattern

The challenging part is to show:

Theorem

*For any query Q closed under homomorphisms and **unbounded**, $\mathrm{PQE}(Q)$ is **#P-hard***

Idea: find a **tight pattern**, i.e., a graph with three distinguished edges $\rightarrow$ $\rightarrow$ $\rightarrow$ such that:



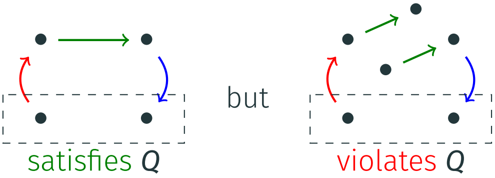
Basic idea: finding a tight pattern

The challenging part is to show:

Theorem

For any query Q closed under homomorphisms and unbounded, $\mathrm{PQE}(Q)$ is #P-hard

Idea: find a tight pattern, i.e., a graph with three distinguished edges $\rightarrow \rightarrow \rightarrow$ such that:

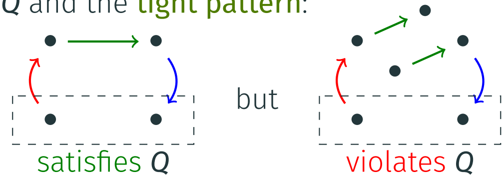


Theorem

Any unbounded query closed under homomorphisms has a tight pattern

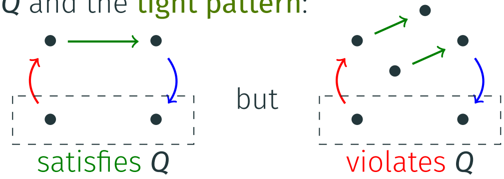
Using tight patterns to show hardness of PQE

- Fix the query Q and the **tight pattern**:



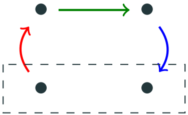
Using tight patterns to show hardness of PQE

- Fix the query Q and the **tight pattern**:



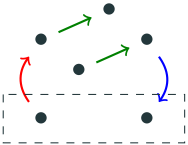
- We reduce from PQE for the **intractable** CQ: Q_0 : $x \longrightarrow y \longrightarrow z \longrightarrow w$

- Fix the query Q and the tight pattern:



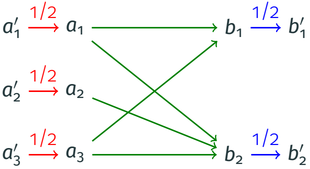
satisfies Q

but

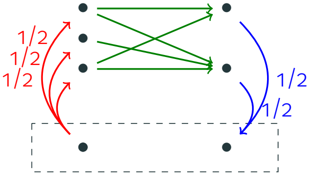


violates Q

- We reduce from PQE for the intractable CQ: $Q_0 : x \rightarrow y \rightarrow z \rightarrow w$

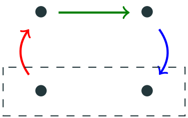


is coded as



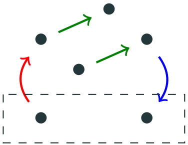
Using tight patterns to show hardness of PQE

- Fix the query Q and the **tight pattern**:



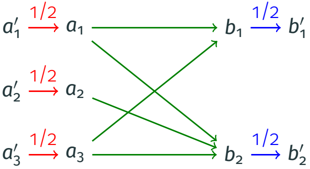
satisfies Q

but

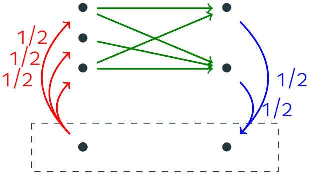


violates Q

- We reduce from PQE for the **intractable** CQ: $Q_0 : x \longrightarrow y \longrightarrow z \longrightarrow w$



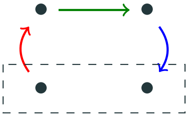
is coded as



Idea: possible worlds at the **left** have a path that matches Q_0
iff the corresponding possible world of the TID at the **right** satisfies the query Q...

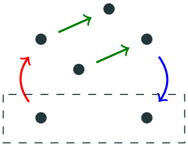
Using tight patterns to show hardness of PQE

- Fix the query Q and the **tight pattern**:



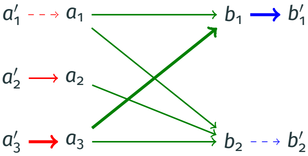
satisfies Q

but

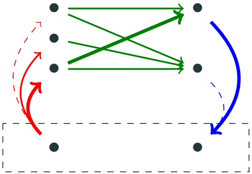


violates Q

- We reduce from PQE for the **intractable** CQ: $Q_0 : x \longrightarrow y \longrightarrow z \longrightarrow w$



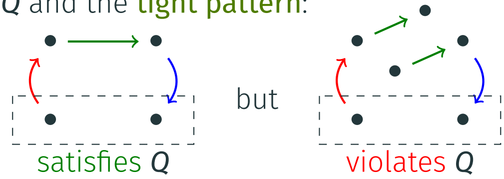
is coded as



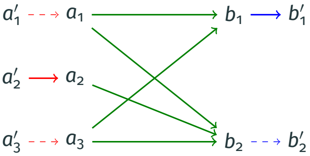
Idea: possible worlds at the **left** have a path that matches Q_0
iff the corresponding possible world of the TID at the **right** satisfies the query Q...

Using tight patterns to show hardness of PQE

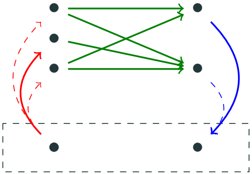
- Fix the query Q and the **tight pattern**:



- We reduce from PQE for the **intractable** CQ: $Q_0 : x \longrightarrow y \longrightarrow z \longrightarrow w$



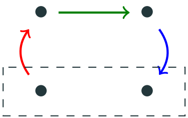
is coded as



Idea: possible worlds at the **left** have a path that matches Q_0
iff the corresponding possible world of the TID at the **right** satisfies the query Q...

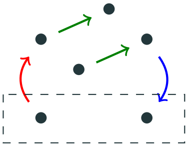
Using tight patterns to show hardness of PQE

- Fix the query Q and the **tight pattern**:



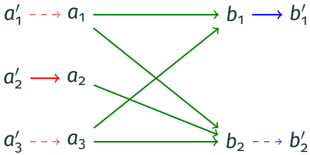
satisfies Q

but

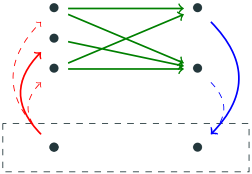


violates Q

- We reduce from PQE for the **intractable** CQ: $Q_0 : x \longrightarrow y \longrightarrow z \longrightarrow w$



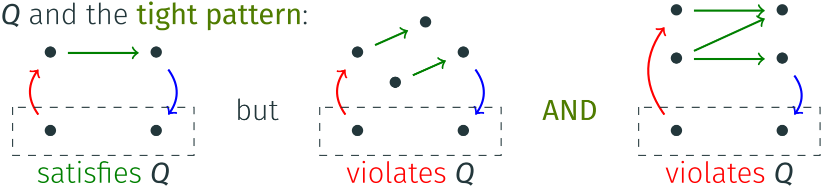
is coded as



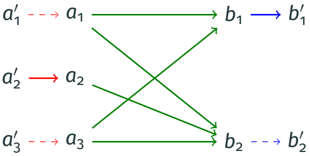
Idea: possible worlds at the **left** have a path that matches Q_0
iff the corresponding possible world of the TID at the **right** satisfies the query Q…
… except we need **more** from the tight pattern!

Using tight patterns to show hardness of PQE

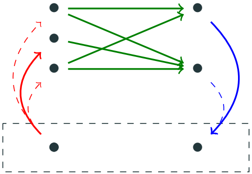
- Fix the query Q and the tight pattern:



- We reduce from PQE for the **intractable** CQ: $Q_0 : x \longrightarrow y \longrightarrow z \longrightarrow w$



is coded as



Idea: possible worlds at the **left** have a path that matches Q_0
iff the corresponding possible world of the TID at the **right** satisfies the query Q...
... except we need **more** from the tight pattern!

Table of contents

Going back to more restricted instances

OK, PQE *is **intractable** for essentially all queries. What now?*

Going back to more restricted instances

OK, PQE is ***intractable*** *for essentially all queries. What now?*

- We could restrict the **structure** of instances: instead of arbitrary graphs, focus on:
  - probabilistic **words**
  - probabilistic **trees**
  - probabilistic graphs with **bounded treewidth**

Going back to more restricted instances

OK, PQE *is **intractable** for essentially all queries. What now?*

- We could restrict the **structure** of instances: instead of arbitrary graphs, focus on:
  - probabilistic **words**
  - probabilistic **trees**
  - probabilistic graphs with **bounded treewidth**
- In the non-probabilistic case, this ensures tractability for **complex queries**

$\rightarrow$ Could the same be true in the **probabilistic case**?

Going back to more restricted instances

OK, PQE is intractable for essentially all queries. What now?

- We could restrict the **structure** of instances: instead of arbitrary graphs, focus on:
  - probabilistic **words**
  - probabilistic **trees**
  - probabilistic graphs with **bounded treewidth**
- In the non-probabilistic case, this ensures tractability for **complex queries**

$\rightarrow$ Could the same be true in the **probabilistic case**?

Theorem (Amarilli, Bourhis, and Senellart 2015; Amarilli, Bourhis, and Senellart 2016)

*Let $k \in \mathbb{N}$ be a constant bound, and let Q be a Boolean **monadic second-order** query. Then $\mathrm{PQE}(Q)$ is in **PTIME** on input TID instances with **treewidth** $\leq k$*

Going back to more restricted instances

*OK, PQE is **intractable** for essentially all queries. What now?*

- We could restrict the **structure** of instances: instead of arbitrary graphs, focus on:
  - probabilistic **words**
  - probabilistic **trees**
  - probabilistic graphs with **bounded treewidth**
- In the non-probabilistic case, this ensures tractability for **complex queries**

$\rightarrow$ Could the same be true in the **probabilistic case**?

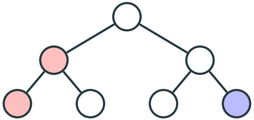
Theorem (Amarilli, Bourhis, and Senellart 2015; Amarilli, Bourhis, and Senellart 2016)

*Let $k \in \mathbb{N}$ be a constant bound, and let Q be a Boolean **monadic second-order** query. Then $\mathrm{PQE}(Q)$ is in **PTIME** on input TID instances with **treewidth** $\leq k$*

*Conversely, there is a query Q for which $\mathrm{PQE}(Q)$ is intractable on **any** input instance family of unbounded treewidth (under some technical assumptions)*

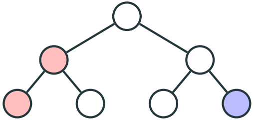
Reminder: Non-probabilistic query evaluation on trees

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



Reminder: Non-probabilistic query evaluation on trees

Database: a **tree** T where nodes have a color from an alphabet $\bigcirc$ $\bigcirc$ $\bigcirc$



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

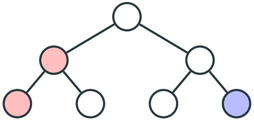
- $P_{\bigcirc}(x)$ means "x is blue"
- $x \rightarrow y$ means "x is the parent of y"

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

$\exists x\, y\ P_{\bigcirc}(x) \land P_{\bigcirc}(y)$

Reminder: Non-probabilistic query evaluation on trees

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



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

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

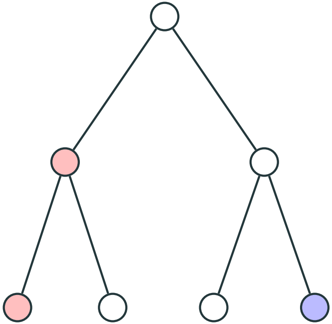
"Is there both a pink and a blue node?"

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

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

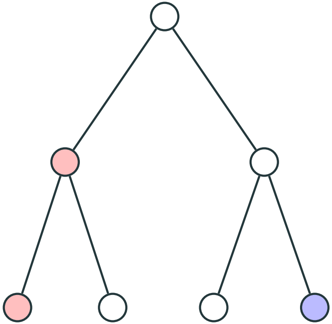
Tree automata

Tree alphabet:



Tree automata

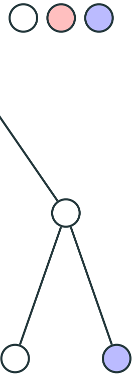
Tree alphabet: 



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

Tree automata

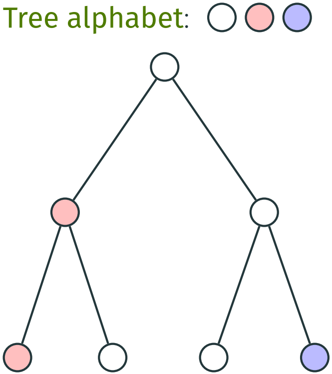
Tree alphabet:



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

Tree automata

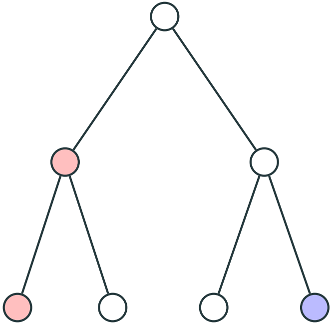
Tree alphabet:



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

Tree automata

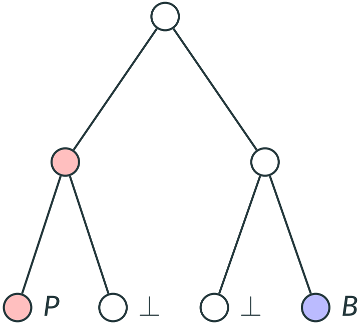
Tree alphabet:



- 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

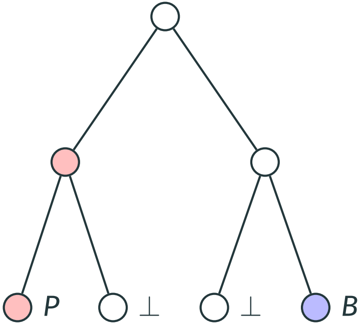
Tree alphabet: ○ ○ ○



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

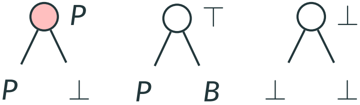
Tree automata

Tree alphabet: 



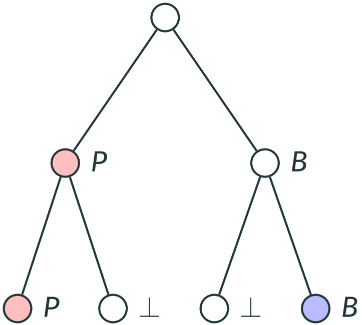
- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- **States:** $\{\perp, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** 

- **Transitions** (examples):

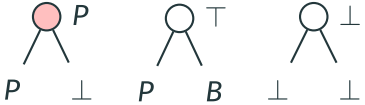


Tree automata

Tree alphabet:

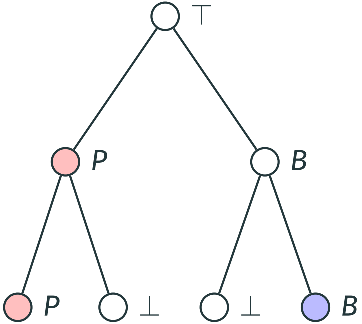


- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- **States:** $\{\perp, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** $\perp$ P B
- **Transitions** (examples):

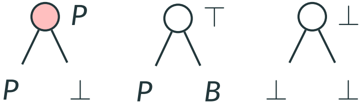


Tree automata

Tree alphabet: 

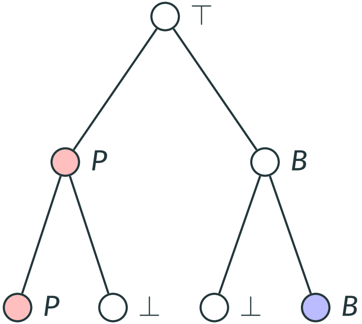


- Bottom-up deterministic **tree automaton**
- *“Is there both a pink and a blue node?”*
- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:**  $\bot$  P  B
- **Transitions** (examples):



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:** 

- **Transitions** (examples):

Theorem (Thatcher and Wright 1968)

MSO and tree automata have the same expressive power on trees

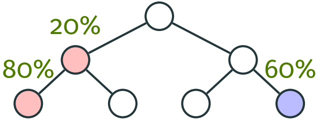
Probabilistic query evaluation on trees

Let's now define the PQE problem for MSO queries on trees:

Probabilistic query evaluation on trees

Let's now define the PQE problem for MSO queries on trees:

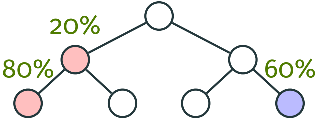
Database: a **tree** T where each node has a probability of **keeping its color** (vs taking the **default color** ○)



Probabilistic query evaluation on trees

Let's now define the **PQE problem** for MSO queries on trees:

Database: a **tree** T where each node has a probability of **keeping its color** (vs taking the **default color** ○)



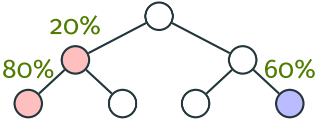
Query Q: in monadic second-order logic (MSO)

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

Probabilistic query evaluation on trees

Let's now define the PQE problem for MSO queries on trees:

Database: a **tree** T where each node has a probability of **keeping its color** (vs taking the **default color** ○)



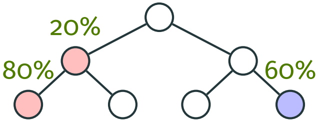
Query Q: in monadic second-order logic (MSO)

$\exists x\, y\; P_{\circ}(x) \land P_{\circ}(y)$

Result: **probability** that the probabilistic tree T satisfies the query Q

Probabilistic query evaluation on trees

Let's now define the PQE problem for MSO queries on trees:



Database: a **tree** T where each node has a probability of **keeping its color** (vs taking the **default color** ○)

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

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

Result: **probability** that the probabilistic tree T satisfies the query Q

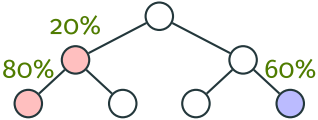
Theorem

For any fixed MSO query Q, the problem $\mathrm{PQE}(Q)$ on trees is in PTIME

Probabilistic query evaluation on trees

Let's now define the **PQE problem** for MSO queries on trees:

Database: a **tree** T where each node has a probability of **keeping its color** (vs taking the **default color**)



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

$$\exists x\, y\; P_{\circ}(x) \land P_{\circ}(y)$$

Result: **probability** that the probabilistic tree T satisfies the query Q

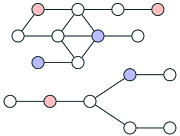
Theorem

For any fixed ***MSO query*** Q*, the problem* $\mathrm{PQE}(Q)$ *on trees is in* ***PTIME***

We can extend this to **treelike data**, i.e., data of bounded **treewidth**...

Courcelle's theorem and extension to PQE

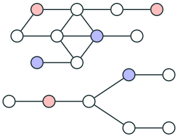
Treelike **data**



MSO query

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

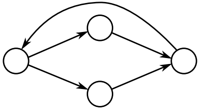
Treelike **data**



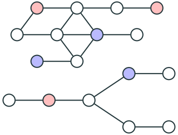
MSO query

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

Tree **automaton**

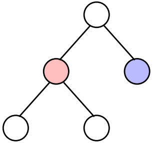


Treelike **data**



linear

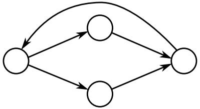
Tree **encoding**



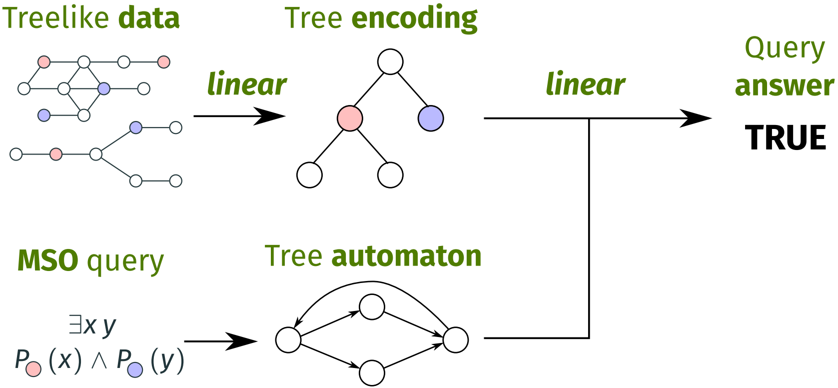
MSO query

$$\exists x\, y \quad P_{\bullet}(x) \wedge P_{\bullet}(y)$$

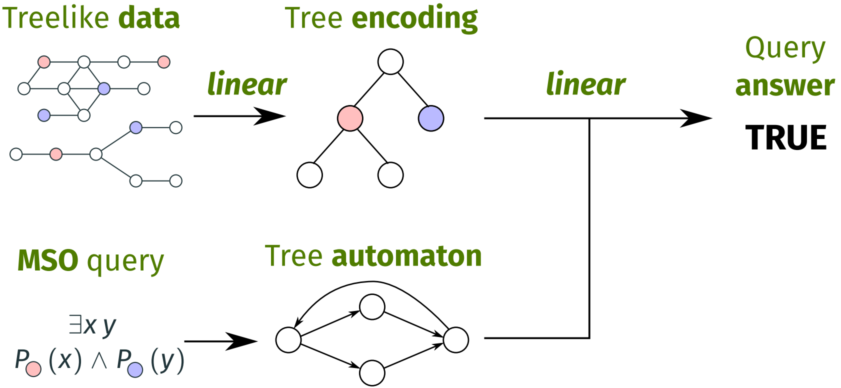
Tree **automaton**



Courcelle's theorem and extension to PQE



Courcelle's theorem and extension to PQE

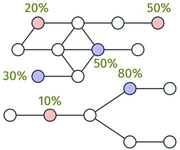


Theorem (Courcelle 1990)

For any fixed Boolean MSO query $\boldsymbol{Q}$ *and* $\boldsymbol{k} \in \mathbb{N}$*, given a database* $\boldsymbol{D}$ *of treewidth* $\leq \boldsymbol{k}$*, we can compute in linear time in* $\boldsymbol{D}$ *whether* $\boldsymbol{D}$ *satisfies* $\boldsymbol{Q}$

Courcelle's theorem and extension to PQE

Probabilistic treelike **data**

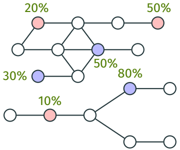


MSO query

$$\exists x\, y$$
$$P_{\bigcirc}(x) \land P_{\bigcirc}(y)$$

Courcelle's theorem and extension to PQE

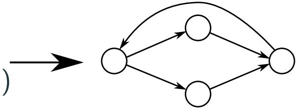
Probabilistic treelike **data**



MSO query

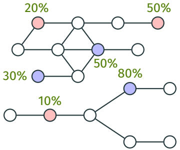
$$\exists x\, y \quad P(x) \wedge P(y)$$

Tree **automaton**



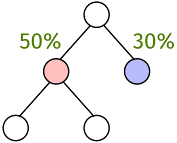
Courcelle's theorem and extension to PQE

Probabilistic treelike **data**



linear

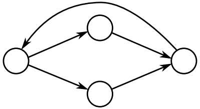
Probabilistic tree **encoding**



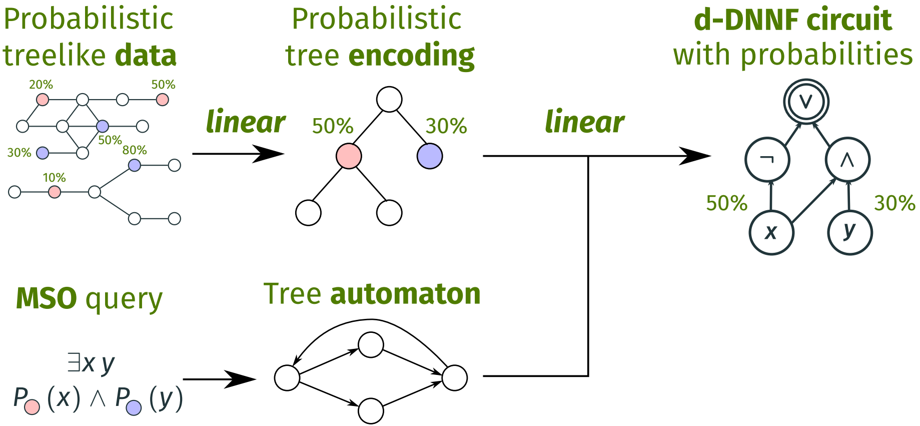
MSO query

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

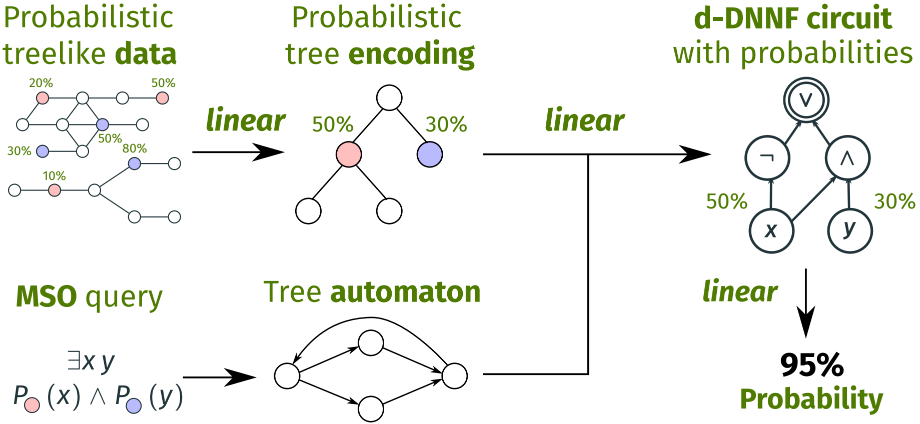
Tree **automaton**



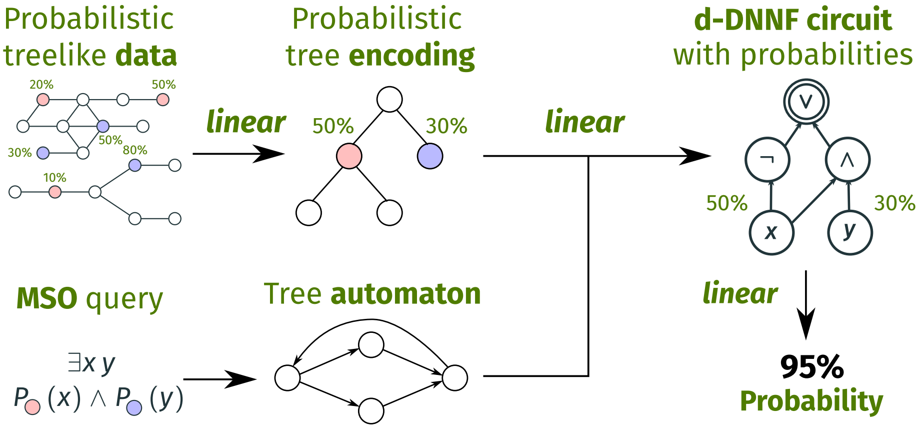
Courcelle's theorem and extension to PQE



Courcelle's theorem and extension to PQE



Courcelle's theorem and extension to PQE



Theorem (Amarilli, Bourhis, and Senellart 2015; Amarilli, Bourhis, and Senellart 2016)

For any fixed Boolean MSO query Q *and* $k \in \mathbb{N}$*, given a database* D *of treewidth* $\leq k$*, we can solve the PQE problem in linear time (assuming constant-time arithmetics)*

Table of contents

Problem statement

What if we restricted probabilities on input instances to always be $1/2$*?*

Problem statement

What if we restricted probabilities on input instances to always be $1/2$*?*

- The PQE problem becomes the subgraph counting (SC) problem:
  - $\rightarrow$ $\mathrm{SC}(Q)$: given a graph, how many of its subgraphs satisfy Q

Problem statement

What if we restricted probabilities on input instances to always be $1/2$*?*

- The PQE problem becomes the **subgraph counting** (SC) problem:
  - $\rightarrow$ $\mathrm{SC}(Q)$: given a graph, how many of its subgraphs satisfy Q
- The SC problem **reduces** to PQE, but no obvious reduction in the other direction

Problem statement

What if we restricted probabilities on input instances to always be $1/2$?

- The PQE problem becomes the **subgraph counting** (SC) problem:
  - $\rightarrow$ $\mathrm{SC}(Q)$: given a graph, how many of its subgraphs satisfy Q
- The SC problem **reduces** to PQE, but no obvious reduction in the other direction

We limit to **self-join-free CQs** and extend the "small" Dalvi and Suciu dichotomy to SC:

Theorem (Amarilli and Kimelfeld 2020)

Let Q be a self-join-free CQ:

- *If Q is a **star**, then $\mathrm{PQE}(Q)$ is in **PTIME***
- *Otherwise, even $\mathrm{SC}(Q)$ is **#P-hard***

$\rightarrow$ This also extends **beyond arity two** (hierarchical queries)

Table of contents

Conclusion and open problems

We have seen:

- PQE is **#P-hard** for all homomorphism-closed queries except safe UCQs
- PQE is **in PTIME** for MSO on bounded-treewidth graphs and intractable otherwise
- PQE behaves like **unweighted subgraph counting** for self-join-free CQs

Conclusion and open problems

We have seen:

- PQE is **#P-hard** for all homomorphism-closed queries except safe UCQs
- PQE is **in PTIME** for MSO on bounded-treewidth graphs and intractable otherwise
- PQE behaves like **unweighted subgraph counting** for self-join-free CQs

Future directions:

- Understanding **tractable UCQs** better, especially the connection to **circuits**

Conclusion and open problems

We have seen:

- PQE is **#P-hard** for all homomorphism-closed queries except safe UCQs
- PQE is **in PTIME** for MSO on bounded-treewidth graphs and intractable otherwise
- PQE behaves like **unweighted subgraph counting** for self-join-free CQs

Future directions:

- Understanding **tractable UCQs** better, especially the connection to **circuits**
- Tractable **approximation algorithms**, especially for recursive queries

Conclusion and open problems

We have seen:

- PQE is **#P-hard** for all homomorphism-closed queries except safe UCQs
- PQE is **in PTIME** for MSO on bounded-treewidth graphs and intractable otherwise
- PQE behaves like **unweighted subgraph counting** for self-join-free CQs

Future directions:

- Understanding **tractable UCQs** better, especially the connection to **circuits**
- Tractable **approximation algorithms**, especially for recursive queries
- Understand **unweighted subgraph counting** for more general classes

Conclusion and open problems

We have seen:

- PQE is **#P-hard** for all homomorphism-closed queries except safe UCQs
- PQE is **in PTIME** for MSO on bounded-treewidth graphs and intractable otherwise
- PQE behaves like **unweighted subgraph counting** for self-join-free CQs

Future directions:

- Understanding **tractable UCQs** better, especially the connection to **circuits**
- Tractable **approximation algorithms**, especially for recursive queries
- Understand **unweighted subgraph counting** for more general classes
- Extending to **arbitrary-arity data**

Conclusion and open problems

We have seen:

- PQE is **#P-hard** for all homomorphism-closed queries except safe UCQs
- PQE is **in PTIME** for MSO on bounded-treewidth graphs and intractable otherwise
- PQE behaves like **unweighted subgraph counting** for self-join-free CQs

Future directions:

- Understanding **tractable UCQs** better, especially the connection to **circuits**
- Tractable **approximation algorithms**, especially for recursive queries
- Understand **unweighted subgraph counting** for more general classes
- Extending to **arbitrary-arity data**
- Other query features: negation, inequalities, etc.

Conclusion and open problems

We have seen:

- PQE is **#P-hard** for all homomorphism-closed queries except safe UCQs
- PQE is **in PTIME** for MSO on bounded-treewidth graphs and intractable otherwise
- PQE behaves like **unweighted subgraph counting** for self-join-free CQs

Future directions:

- Understanding **tractable UCQs** better, especially the connection to **circuits**
- Tractable **approximation algorithms**, especially for recursive queries
- Understand **unweighted subgraph counting** for more general classes
- Extending to **arbitrary-arity data**
- Other query features: negation, inequalities, etc.
- Connections to other problems, especially **enumeration** of query results and **maintenance under updates**

Conclusion and open problems

We have seen:

- PQE is **#P-hard** for all homomorphism-closed queries except safe UCQs
- PQE is **in PTIME** for MSO on bounded-treewidth graphs and intractable otherwise
- PQE behaves like **unweighted subgraph counting** for self-join-free CQs

Future directions:

- Understanding **tractable UCQs** better, especially the connection to **circuits**
- Tractable **approximation algorithms**, especially for recursive queries
- Understand **unweighted subgraph counting** for more general classes
- Extending to **arbitrary-arity data**
- Other query features: negation, inequalities, etc.
- Connections to other problems, especially **enumeration** of query results and **maintenance under updates**

Thanks for your attention!

Amarilli, Antoine, Pierre Bourhis, and Pierre Senellart (2015). “Provenance Circuits for Trees and Treelike Instances”. In: *ICALP*.

— (2016). “Tractable Lineages on Treelike Instances: Limits and Extensions”. In: *PODS*.

Amarilli, Antoine and Ismail Ilkan Ceylan (2020). “A Dichotomy for Homomorphism-Closed Queries on Probabilistic Graphs”. In: *ICDT*.

Amarilli, Antoine and Benny Kimelfeld (2020). “Uniform Reliability of Self-Join-Free Conjunctive Queries”. Preprint: https://arxiv.org/abs/1908.07093.

Chekuri, Chandra and Julia Chuzhoy (2014). “Polynomial bounds for the grid-minor theorem”. In: *STOC*. DOI: 10.1145/2591796.2591813. URL: http://doi.acm.org/10.1145/2591796.2591813.

Courcelle, Bruno (1990). “The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs”. In: *Inf. Comput.* 85.1. DOI: `10.1016/0890-5401(90)90043-H`. URL: `http://dx.doi.org/10.1016/0890-5401(90)90043-H`.

Dalvi, Nilesh and Dan Suciu (2007). “The dichotomy of conjunctive queries on probabilistic structures”. In: *Proc. PODS*.

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

Fink, Robert and Dan Olteanu (2016). “Dichotomies for queries with negation in probabilistic databases”. In: 41.1, 4:1–4:47.

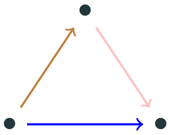
Ganian, Robert, Petr Hlineny, Alexander Langer, Jan Obdrzalek, Peter Rossmanith, and Somnath Sikdar (2014). “Lower bounds on the complexity of MSO1 model-checking”. In: *JCSS* 1.80.

Jung, Jean Christoph and Carsten Lutz (2012). "Ontology-based access to probabilistic data with OWL QL". In: *Proceedings of the 11th International Conference on The Semantic Web - Volume Part I*, pp. 182–197.

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.

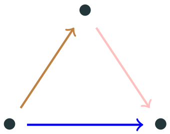
Why can we always find tight patterns?

- Unbounded queries have **arbitrarily large** minimal models
- Take a large minimal model D and **disconnect its edges**:

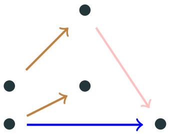


Why can we always find tight patterns?

- Unbounded queries have arbitrarily large minimal models
- Take a large minimal model D and disconnect its edges:

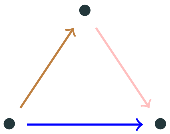


to

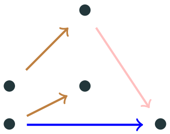


Why can we always find tight patterns?

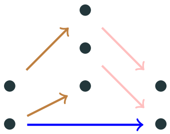
- Unbounded queries have **arbitrarily large** minimal models
- Take a large minimal model D and **disconnect its edges**:



to

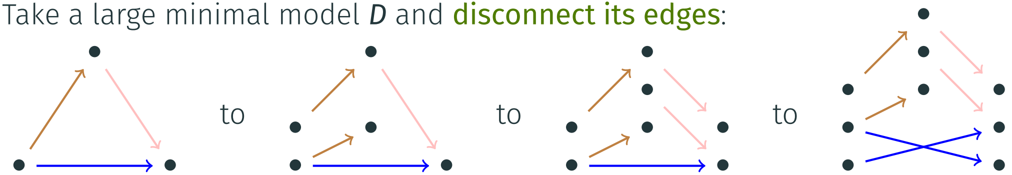


to



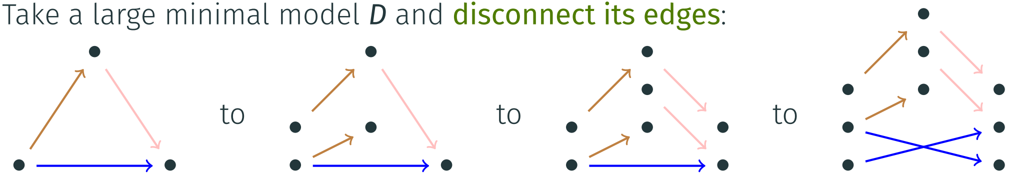
Why can we always find tight patterns?

- Unbounded queries have **arbitrarily large** minimal models
- Take a large minimal model *D* and **disconnect its edges**:



Why can we always find tight patterns?

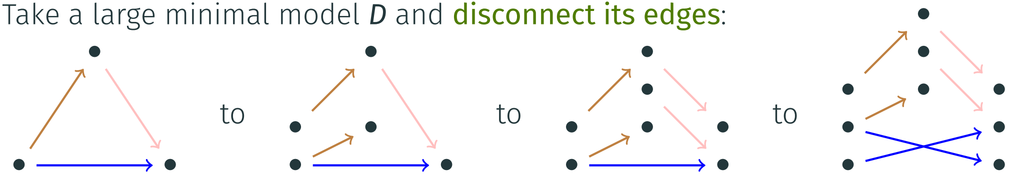
- Unbounded queries have arbitrarily large minimal models
- Take a large minimal model D and disconnect its edges:



- If Q becomes false at one step, then we have found a tight pattern

Why can we always find tight patterns?

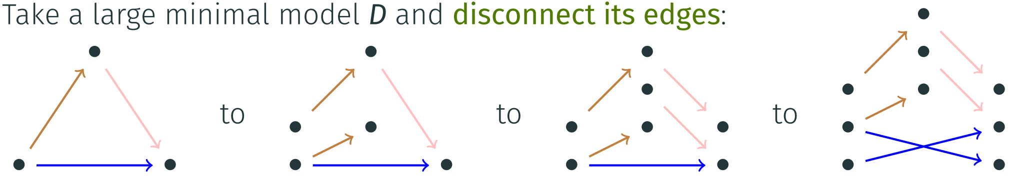
- Unbounded queries have **arbitrarily large** minimal models
- Take a large minimal model D and **disconnect its edges**:



- If Q becomes false at one step, then we have found a **tight pattern**
- Otherwise, we have found a **contradiction**:
  - The disconnection process **terminates**

Why can we always find tight patterns?

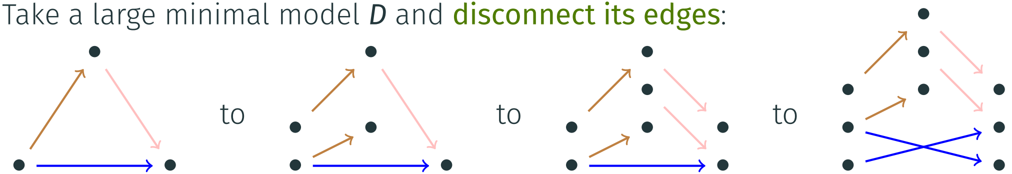
- Unbounded queries have **arbitrarily large** minimal models
- Take a large minimal model D and **disconnect its edges**:



- If Q becomes false at one step, then we have found a **tight pattern**
- Otherwise, we have found a **contradiction**:
  - The disconnection process **terminates**
  - At the end of the process, we obtain a **star** D'

Why can we always find tight patterns?

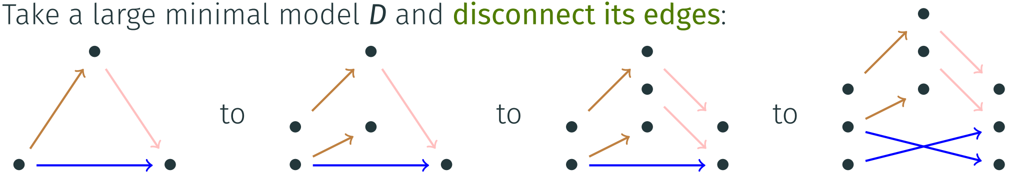
- Unbounded queries have **arbitrarily large** minimal models
- Take a large minimal model D and **disconnect its edges**:



- If Q becomes false at one step, then we have found a **tight pattern**
- Otherwise, we have found a **contradiction**:
  - The disconnection process **terminates**
  - At the end of the process, we obtain a **star** D'
  - It is **homomorphically equivalent** to a constant-sized D'' satisfying Q

Why can we always find tight patterns?

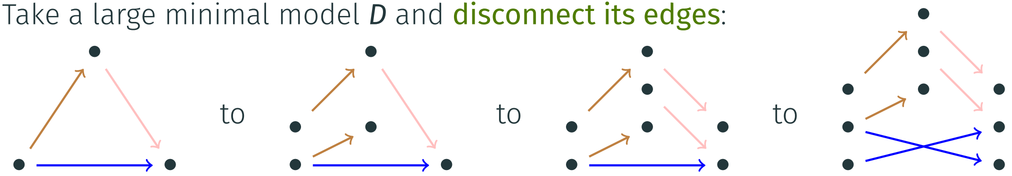
- Unbounded queries have **arbitrarily large** minimal models
- Take a large minimal model D and **disconnect its edges**:



- If Q becomes false at one step, then we have found a **tight pattern**
- Otherwise, we have found a **contradiction**:
  - The disconnection process **terminates**
  - At the end of the process, we obtain a **star** D'
  - It is **homomorphically equivalent** to a constant-sized D'' satisfying Q
  - D'' has a **homomorphism** back to D

Why can we always find tight patterns?

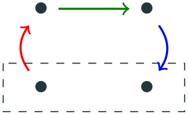
- Unbounded queries have **arbitrarily large** minimal models
- Take a large minimal model D and **disconnect its edges**:



- If Q becomes false at one step, then we have found a **tight pattern**
- Otherwise, we have found a **contradiction**:
  - The disconnection process **terminates**
  - At the end of the process, we obtain a **star** D'
  - It is **homomorphically equivalent** to a constant-sized D'' satisfying Q
  - D'' has a **homomorphism** back to D
  - This contradicts the **minimality** of the large D

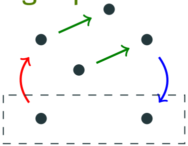
Rescuing the proof

We know that we have a **tight pattern**:



satisfies Q

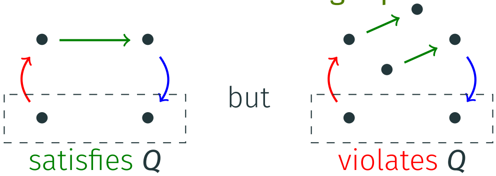
but



violates Q

Rescuing the proof

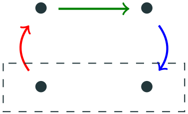
We know that we have a **tight pattern**:



Consider its **iterates**

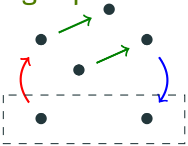
Rescuing the proof

We know that we have a **tight pattern**:



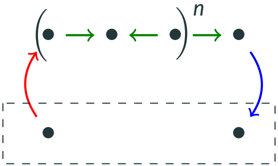
satisfies Q

but



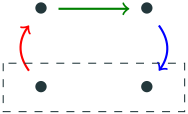
violates Q

Consider its **iterates** for each $n \in \mathbb{N}$:



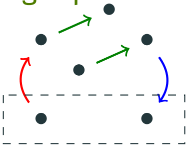
Rescuing the proof

We know that we have a **tight pattern**:



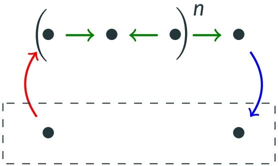
satisfies Q

but



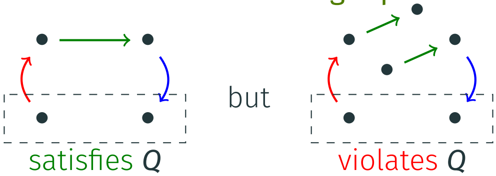
violates Q

Consider its **iterates** for each $n \in \mathbb{N}$:

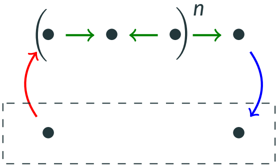


Rescuing the proof

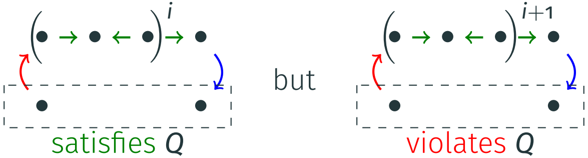
We know that we have a **tight pattern**:



Consider its **iterates** for each $n \in \mathbb{N}$:

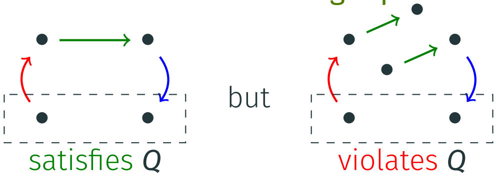


Case 1: some iterate **violates** the query:

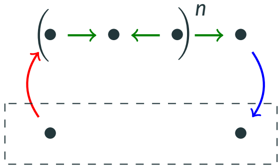


Rescuing the proof

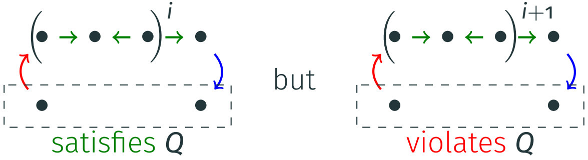
We know that we have a **tight pattern**:



Consider its **iterates** for each $n \in \mathbb{N}$:



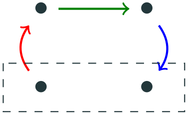
Case 1: some iterate **violates** the query:



$\rightarrow$ Reduce from $\mathrm{PQE}(Q_0)$ as we explained

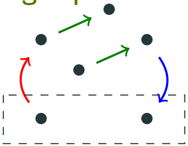
Rescuing the proof

We know that we have a **tight pattern**:



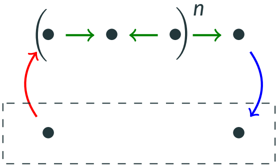
satisfies Q

but

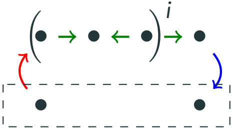


violates Q

Consider its **iterates** for each $n \in \mathbb{N}$:

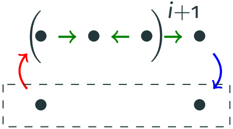


Case 1: some iterate **violates** the query:



satisfies Q

but



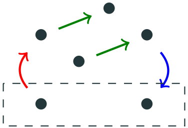
violates Q

$\rightarrow$ Reduce from $\mathrm{PQE}(Q_0)$ as we explained

Case 2: all iterates **satisfy** the query:

satisfies Q for all $n \in \mathbb{N}$

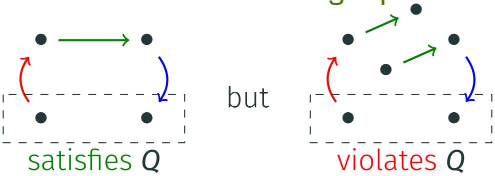
but



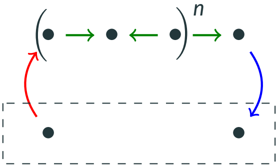
violates Q

Rescuing the proof

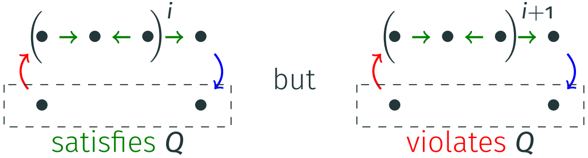
We know that we have a **tight pattern**:



Consider its **iterates** for each $n \in \mathbb{N}$:

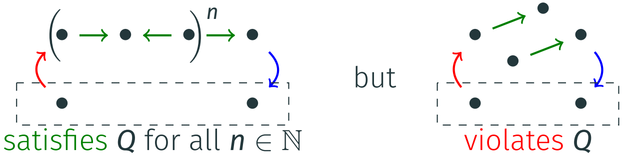


Case 1: some iterate **violates** the query:



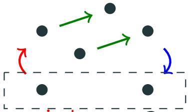
$\rightarrow$ Reduce from $\mathrm{PQE}(Q_0)$ as we explained

Case 2: all iterates **satisfy** the query:



$\rightarrow$ Call this an **iterable pattern**

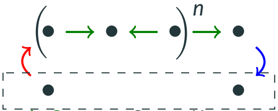
Using iterable patterns to show hardness of PQE

We have an **iterable pattern**:  but 

satisfies Q for all $n \in \mathbb{N}$ | violates Q

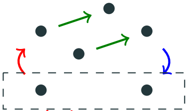
Using iterable patterns to show hardness of PQE

We have an **iterable pattern**:



satisfies Q for all $n \in \mathbb{N}$

but



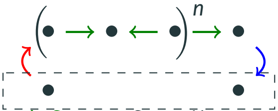
violates Q

Idea: reduce from the **#P-hard** problem **source-to-target connectivity**:

- Input: **undirected graph** with a **source** s and **target** t, all edges have probability $1/2$
- Output: what is the **probability** that the source and target are **connected**?

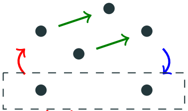
Using iterable patterns to show hardness of PQE

We have an **iterable pattern**:



satisfies Q for all $n \in \mathbb{N}$

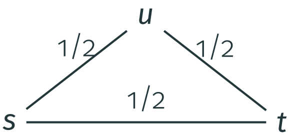
but



violates Q

Idea: reduce from the **#P-hard** problem **source-to-target connectivity**:

- Input: **undirected graph** with a **source** s and **target** t, all edges have probability $1/2$
- Output: what is the **probability** that the source and target are **connected**?

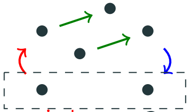


Using iterable patterns to show hardness of PQE

We have an **iterable pattern**:

satisfies Q for all $n \in \mathbb{N}$

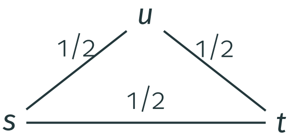
but



violates Q

Idea: reduce from the **#P-hard** problem **source-to-target connectivity**:

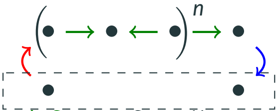
- Input: **undirected graph** with a **source** s and **target** t, all edges have probability $\mathbf{1/2}$
- Output: what is the **probability** that the source and target are **connected**?



is coded as

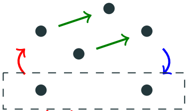
Using iterable patterns to show hardness of PQE

We have an **iterable pattern**:



satisfies Q for all $n \in \mathbb{N}$

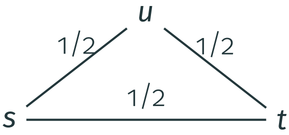
but



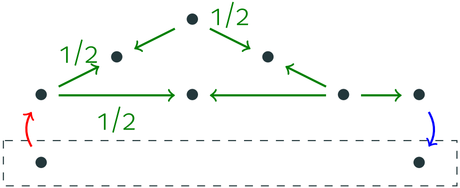
violates Q

Idea: reduce from the **#P-hard** problem **source-to-target connectivity**:

- Input: **undirected graph** with a **source** s and **target** t, all edges have probability $1/2$
- Output: what is the **probability** that the source and target are **connected**?

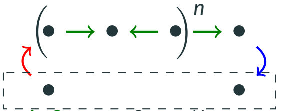


is coded as



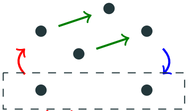
Using iterable patterns to show hardness of PQE

We have an **iterable pattern**:



satisfies Q for all $n \in \mathbb{N}$

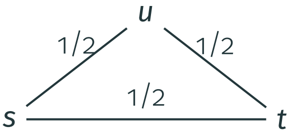
but



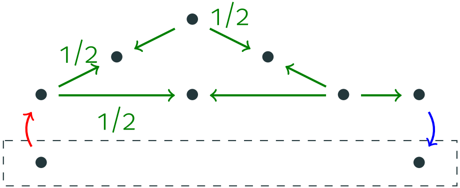
violates Q

Idea: reduce from the **#P-hard** problem **source-to-target connectivity**:

- Input: **undirected graph** with a **source** s and **target** t, all edges have probability $1/2$
- Output: what is the **probability** that the source and target are **connected**?



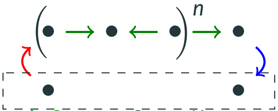
is coded as



Idea: There is a **path connecting s and t** in a possible world of the graph at the left iff the query Q is **satisfied** in the corresponding possible world of the TID at the right

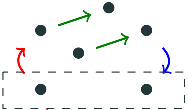
Using iterable patterns to show hardness of PQE

We have an **iterable pattern**:



satisfies Q for all $n \in \mathbb{N}$

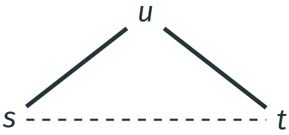
but



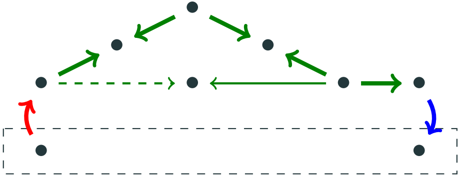
violates Q

Idea: reduce from the **#P-hard** problem **source-to-target connectivity**:

- Input: **undirected graph** with a **source** s and **target** t, all edges have probability $1/2$
- Output: what is the **probability** that the source and target are **connected**?



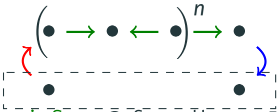
is coded as



Idea: There is a **path connecting s and t** in a possible world of the graph at the left iff the query Q is **satisfied** in the corresponding possible world of the TID at the right

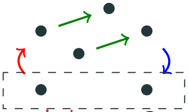
Using iterable patterns to show hardness of PQE

We have an **iterable pattern**:



satisfies Q for all $n \in \mathbb{N}$

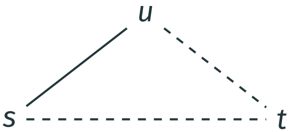
but



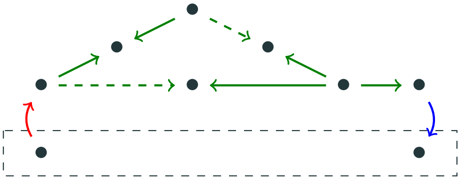
violates Q

Idea: reduce from the **#P-hard** problem **source-to-target connectivity**:

- Input: **undirected graph** with a **source** s and **target** t, all edges have probability $1/2$
- Output: what is the **probability** that the source and target are **connected**?

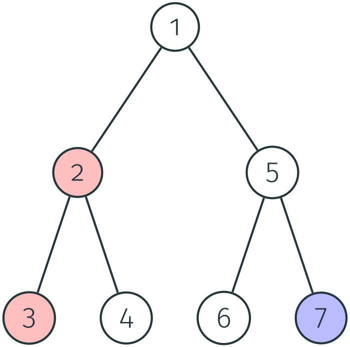


is coded as

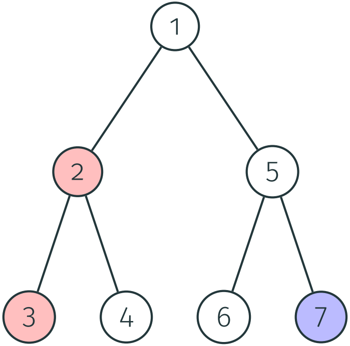


Idea: There is a **path connecting s and t** in a possible world of the graph at the left iff the query Q is **satisfied** in the corresponding possible world of the TID at the right

Uncertain trees: capturing how the query result depends on the choices

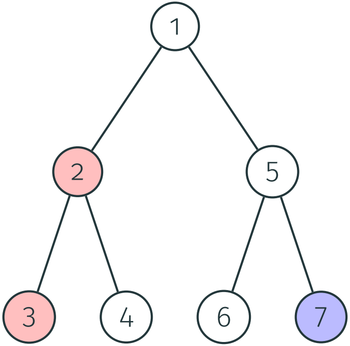


Uncertain trees: capturing how the query result depends on the choices



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

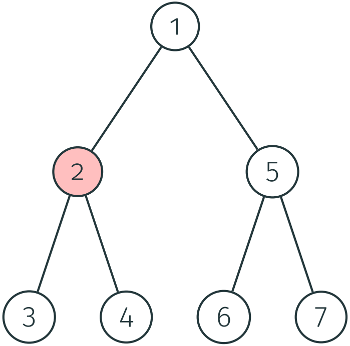
Uncertain trees: capturing how the query result depends on the choices



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

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

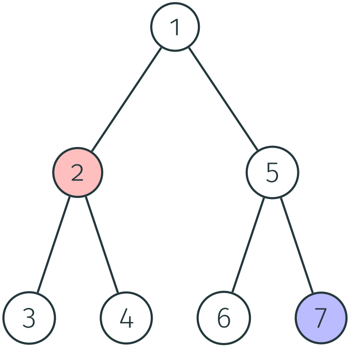
Uncertain trees: capturing how the query result depends on the choices



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

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

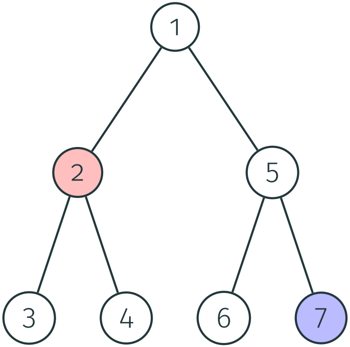
Uncertain trees: capturing how the query result depends on the choices



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

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

Uncertain trees: capturing how the query result depends on the choices

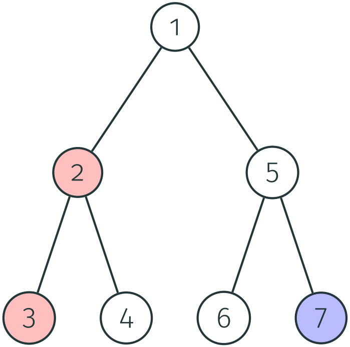


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

Uncertain trees: capturing how the query result depends on the choices



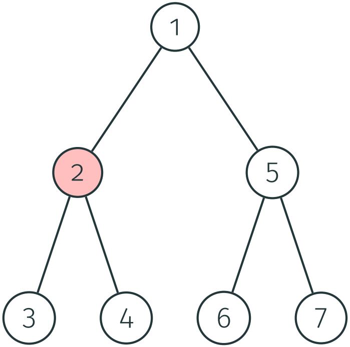
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

The query ***Q*** returns **YES**

Uncertain trees: capturing how the query result depends on the choices



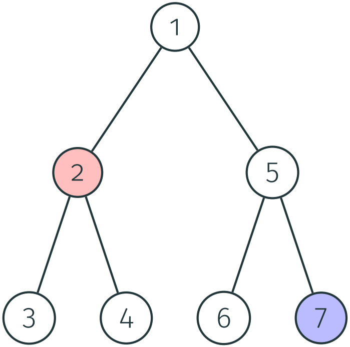
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

The query ***Q*** returns **NO**

Uncertain trees: capturing how the query result depends on the choices



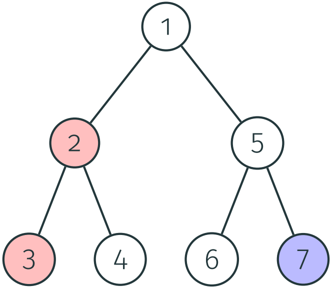
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

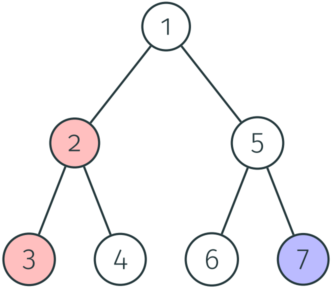
The query Q returns **YES**

Example: Provenance circuit



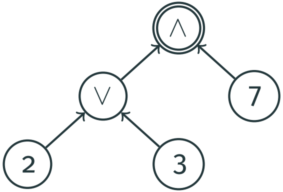
Query: *Is there both a pink and a blue node?*

Example: Provenance circuit

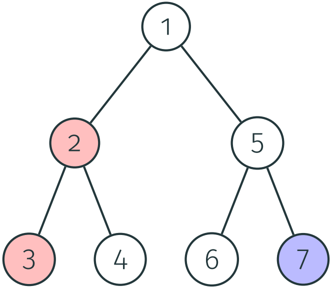


Query: *Is there both a pink and a blue node?*

Provenance circuit:

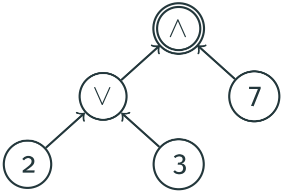


Example: Provenance circuit



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

Provenance circuit:



Formal definition of provenance circuits:

- 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

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

Theorem

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

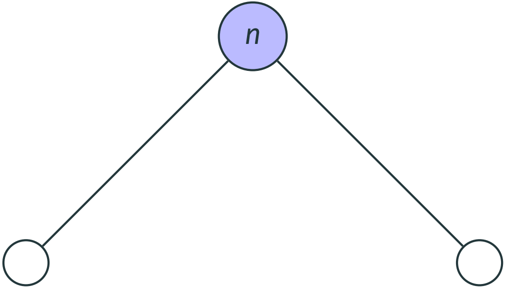
- **Alphabet:** 
- **Automaton:** *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

Provenance circuits on trees

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:**

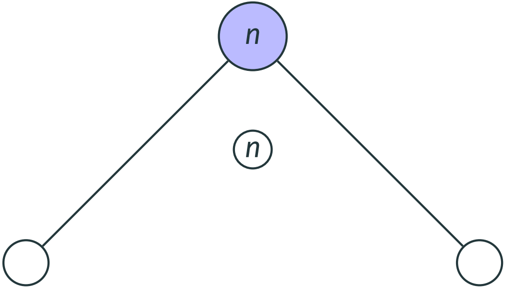


Provenance circuits on trees

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:**

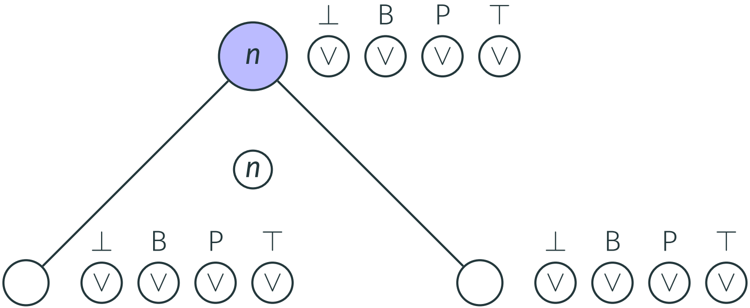


Provenance circuits on trees

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:**

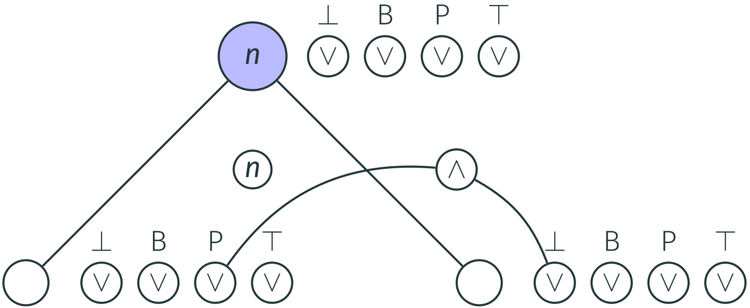


Provenance circuits on trees

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:**

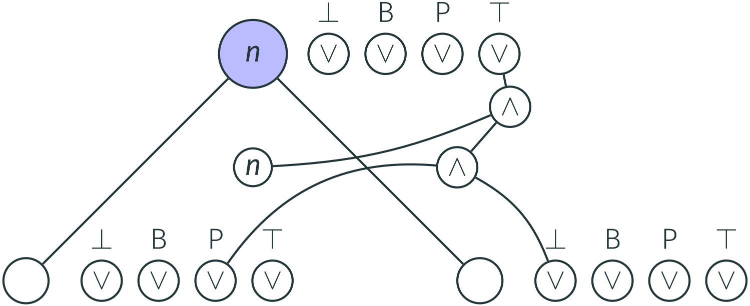


Provenance circuits on trees

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:**

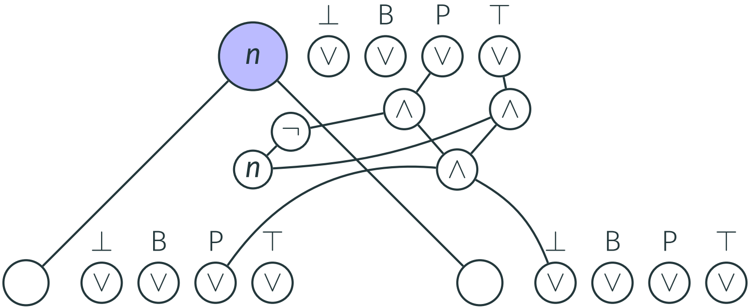


Provenance circuits on trees

Theorem

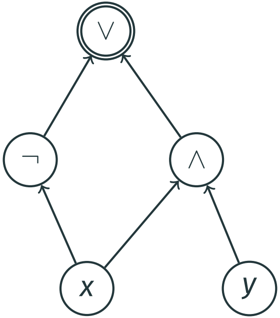
For any bottom-up ***tree automaton*** A *and input* ***tree*** T*,*
we can build a Boolean ***provenance circuit*** *of* A *on* T *in* $O(|A| \times |T|)$

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



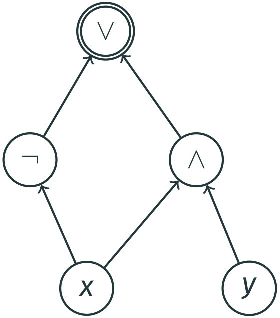
Computing the probability of the circuit

- We now have a circuit and a probability P for each variable (= tree node)



Computing the probability of the circuit

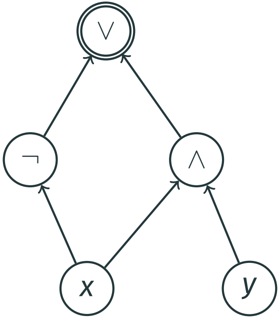
- We now have a circuit and a probability P for each variable (= tree node)



- $P(x) = 40\%$
- $P(y) = 50\%$

Computing the probability of the circuit

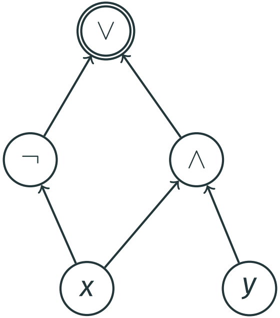
- We now have a circuit and a probability P for each variable (= tree node)
- Each variable x is true independently with probability $P(x)$



- $P(x) = 40\%$
- $P(y) = 50\%$

Computing the probability of the circuit

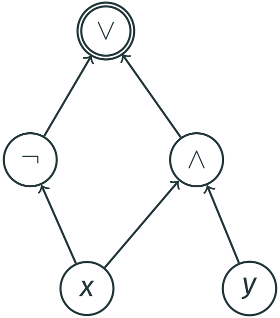
- We now have a circuit and a probability P for each variable (= tree node)
- Each variable x is true independently with probability $P(x)$
- What is the probability that the circuit evaluates to true?



- $P(x) = 40\%$
- $P(y) = 50\%$

Computing the probability of the circuit

- We now have a **circuit** and a **probability** P for each variable (= tree node)
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?

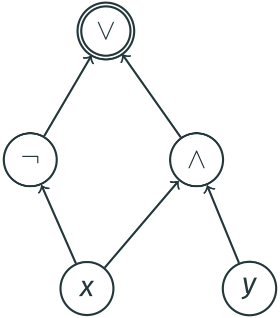


- In general, **#P-hard** (harder than SAT)

- $P(x) = 40\%$
- $P(y) = 50\%$

Computing the probability of the circuit

- We now have a circuit and a probability P for each variable (= tree node)
- Each variable x is true independently with probability $P(x)$
- What is the probability that the circuit evaluates to true?

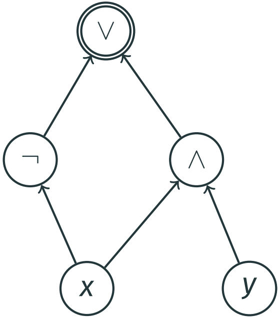


- In general, #P-hard (harder than SAT)
- Here it's easy:

- $P(x) = 40\%$
- $P(y) = 50\%$

Computing the probability of the circuit

- We now have a circuit and a probability P for each variable (= tree node)
- Each variable x is true independently with probability $P(x)$
- What is the probability that the circuit evaluates to true?

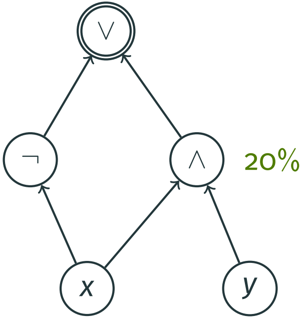


- $P(x) = 40\%$
- $P(y) = 50\%$

- In general, #P-hard (harder than SAT)
- Here it's easy:
  - The inputs to the $\wedge$-gate are independent

Computing the probability of the circuit

- We now have a circuit and a probability P for each variable (= tree node)
- Each variable x is true independently with probability $P(x)$
- What is the probability that the circuit evaluates to true?

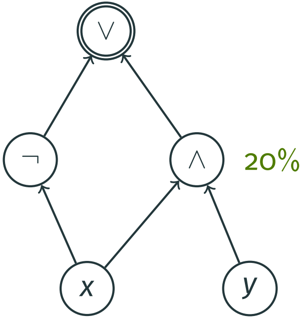


- $P(x) = 40\%$
- $P(y) = 50\%$

- In general, #P-hard (harder than SAT)
- Here it's easy:
  - The inputs to the $\wedge$-gate are independent

Computing the probability of the circuit

- We now have a circuit and a probability P for each variable (= tree node)
- Each variable x is true independently with probability $P(x)$
- What is the probability that the circuit evaluates to true?

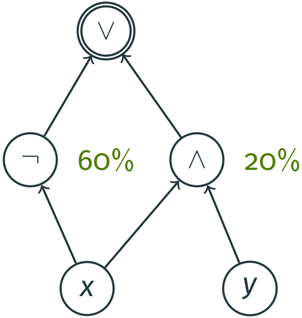


- $P(x) = 40\%$
- $P(y) = 50\%$

- In general, #P-hard (harder than SAT)
- Here it's easy:
  - The inputs to the $\wedge$-gate are independent
  - The $\neg$-gate has probability $1 - P(\text{input})$

Computing the probability of the circuit

- We now have a circuit and a probability P for each variable (= tree node)
- Each variable x is true independently with probability $P(x)$
- What is the probability that the circuit evaluates to true?

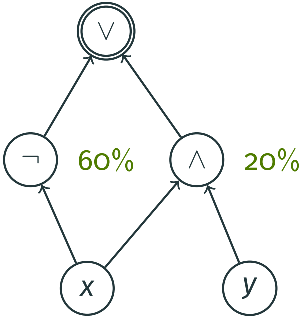


- $P(x) = 40\%$
- $P(y) = 50\%$

- In general, #P-hard (harder than SAT)
- Here it's easy:
  - The inputs to the $\wedge$-gate are independent
  - The $\neg$-gate has probability $1 - P(\text{input})$

Computing the probability of the circuit

- We now have a **circuit** and a **probability** P for each variable (= tree node)
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?

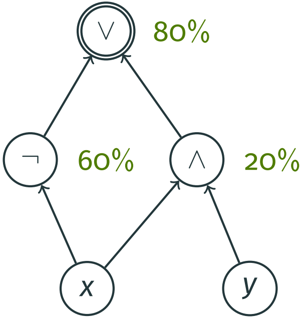


- $P(x) = 40\%$
- $P(y) = 50\%$

- In general, **#P-hard** (harder than SAT)
- Here it's **easy**:
  - The inputs to the $\wedge$**-gate** are **independent**
  - The $\neg$**-gate** has probability $1 - P(\text{input})$
  - The $\vee$**-gate** has **mutually exclusive** inputs

Computing the probability of the circuit

- We now have a **circuit** and a **probability** P for each variable (= tree node)
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?

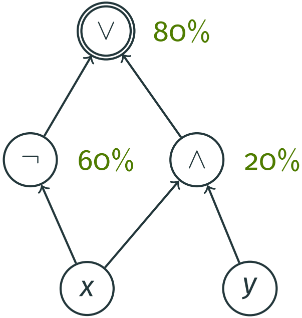


- $P(x) = 40\%$
- $P(y) = 50\%$

- In general, **#P-hard** (harder than SAT)
- Here it's **easy**:
  - The inputs to the $\wedge$**-gate** are **independent**
  - The $\neg$**-gate** has probability $1 - P(\text{input})$
  - The $\vee$**-gate** has **mutually exclusive** inputs

Computing the probability of the circuit

- We now have a **circuit** and a **probability** P for each variable (= tree node)
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?



- $P(x) = 40\%$
- $P(y) = 50\%$

- In general, **#P-hard** (harder than SAT)
- Here it's **easy**:
  - The inputs to the **$\wedge$-gate** are **independent**
  - The **$\neg$-gate** has probability $1 - P(\text{input})$
  - The **$\vee$-gate** has **mutually exclusive** inputs

$\rightarrow$ The circuit that we constructed falls in a **restricted class** satisfying such conditions

d-DNNFs

Lemma

For unambiguous automata, the provenance circuit that we compute is a d-DNNF

d-DNNF requirements

d-DNNFs

Lemma

For unambiguous automata, the provenance circuit that we compute is a d-DNNF

d-DNNF requirements

- $\neg$ gates only have **variables** as inputs

d-DNNFs

Lemma

For unambiguous automata, the provenance circuit that we compute is a d-DNNF

d-DNNF requirements

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

d-DNNFs

Lemma

For unambiguous automata, the provenance circuit that we compute is a d-DNNF

d-DNNF requirements

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

d-DNNFs

Lemma

For unambiguous automata, the provenance circuit that we compute is a d-DNNF

d-DNNF requirements

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

… make probability computation **easy**!

d-DNNFs

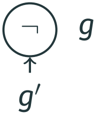
Lemma

For unambiguous automata, the provenance circuit that we compute is a d-DNNF

d-DNNF requirements

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

... make probability computation **easy**!



d-DNNFs

Lemma

For unambiguous automata, the provenance circuit that we compute is a d-DNNF

d-DNNF requirements

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

... make probability computation **easy**!

$$\begin{tikzcd} \neg \; g \\ g' \arrow[u] \end{tikzcd}$$

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

d-DNNFs

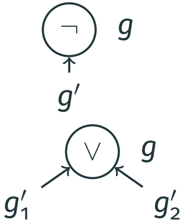
Lemma

For unambiguous automata, the provenance circuit that we compute is a d-DNNF

d-DNNF requirements

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

... make probability computation **easy**!



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

d-DNNFs

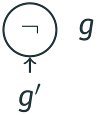
Lemma

For unambiguous automata, the provenance circuit that we compute is a d-DNNF

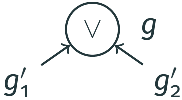
d-DNNF requirements

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

... make probability computation **easy**!



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



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

d-DNNFs

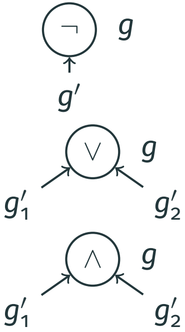
Lemma

For unambiguous automata, the provenance circuit that we compute is a d-DNNF

d-DNNF requirements

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

… make probability computation **easy**!



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

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

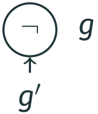
d-DNNFs

Lemma

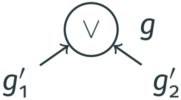
For unambiguous automata, the provenance circuit that we compute is a d-DNNF

d-DNNF requirements ... make probability computation easy!

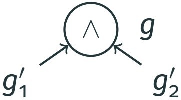
- $\neg$ gates only have **variables** as inputs
- $\vee$ gates always have **mutually exclusive** inputs
- $\wedge$ gates are all on **independent** inputs



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



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



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

d-DNNFs

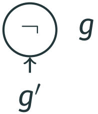
Lemma

For unambiguous automata, the provenance circuit that we compute is a d-DNNF

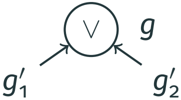
d-DNNF requirements

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

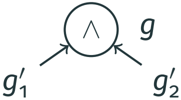
… make probability computation **easy**!



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



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



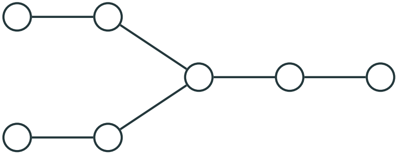
$$P(g) := P(g'_1) \times P(g'_2)$$

$\rightarrow$ Connections to other circuit classes in the field of **knowledge compilation**

Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

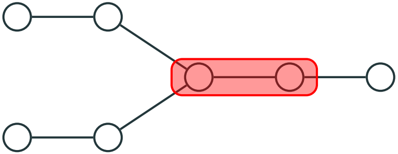
Treewidth by example:



Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

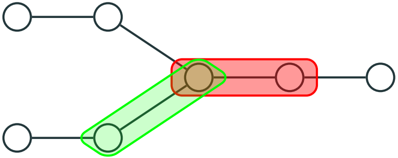
Treewidth by example:



Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

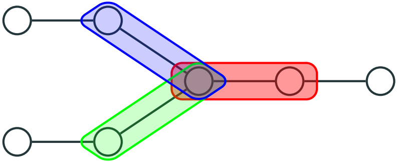
Treewidth by example:



Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

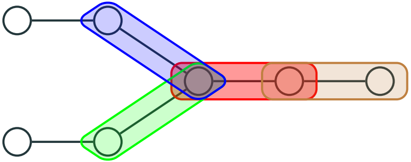
Treewidth by example:



Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

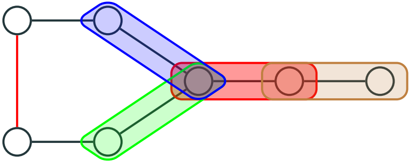
Treewidth by example:



Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

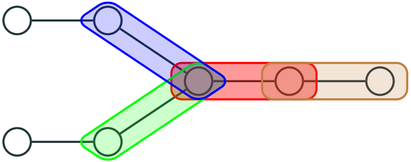
Treewidth by example:

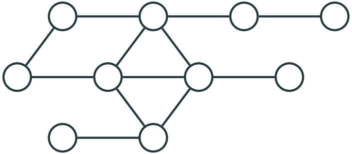


Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:

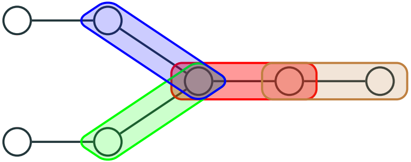


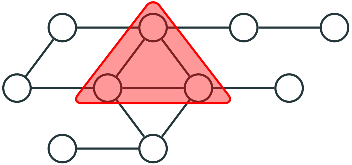


Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:

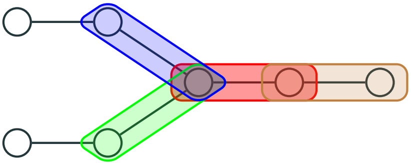


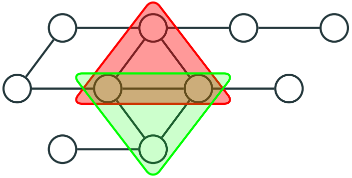


Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:

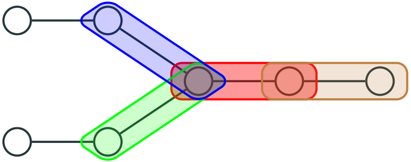


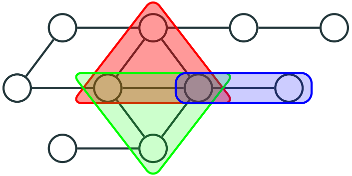


Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:

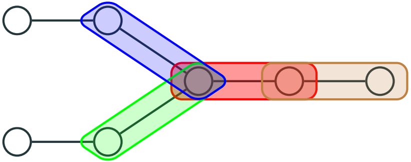


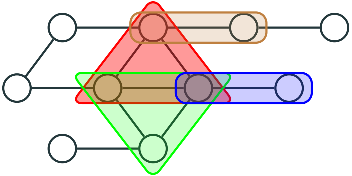


Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:

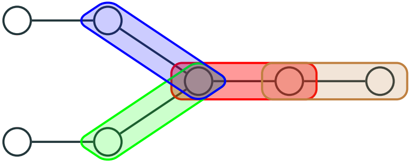


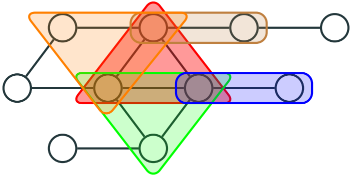


Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:

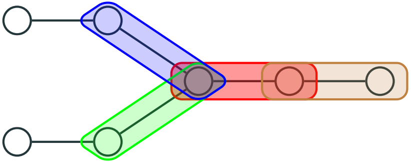


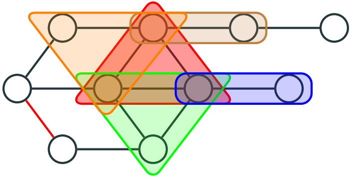


Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:

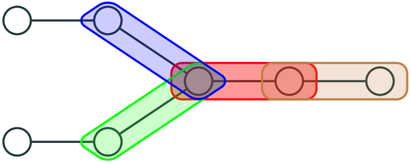


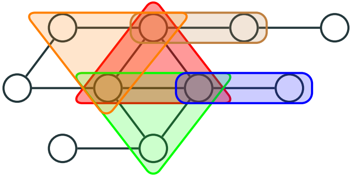


Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:

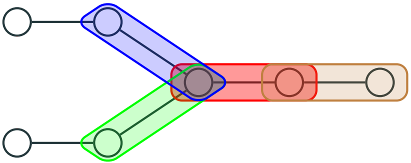


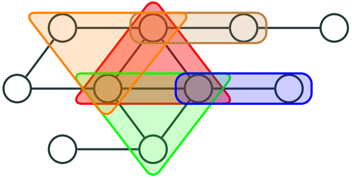


Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:



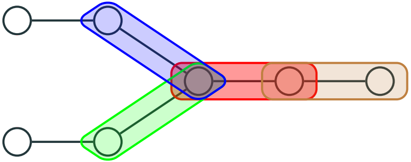


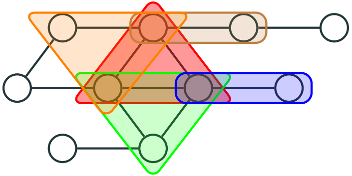
- **Trees** have treewidth 1
- **Cycles** have treewidth 2
- **k-cliques** and **$(k-1)$-grids** have treewidth $k-1$

Treewidth

We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:





- **Trees** have treewidth **1**
- **Cycles** have treewidth **2**
- **k-cliques** and **$(k-1)$-grids** have treewidth $k-1$

$\rightarrow$ **Treelike**: the **treewidth** is bounded by a **constant**

Why is this a dichotomy? Where's the lower bound?

Theorem (Amarilli, Bourhis, and Senellart 2016)

For any set of edge colors, there is a first-order query **Q** *such that for any constructible unbounded-treewidth family* $\mathcal{I}$ *of probabilistic graphs, the PQE problem for* **Q** *and* $\mathcal{I}$ *is #P-hard under RP reductions*

- **Family:** an infinite set of graphs allowed as input (with arbitrary probabilities) so in particular **closed under subgraphs**

Why is this a dichotomy? Where's the lower bound?

Theorem (Amarilli, Bourhis, and Senellart 2016)

For any set of edge colors, there is a first-order query Q such that for any constructible unbounded-treewidth family $\mathcal{I}$ of probabilistic graphs, the PQE problem for Q and $\mathcal{I}$ is #P-hard under RP reductions

- **Family:** an infinite set of graphs allowed as input (with arbitrary probabilities) so in particular **closed under subgraphs**
- **Unbounded-treewidth**: for all $k \in \mathbb{N}$, there is $I_k \in \mathcal{I}$ of treewidth $\geq k$

Why is this a dichotomy? Where's the lower bound?

Theorem (Amarilli, Bourhis, and Senellart 2016)

For any set of edge colors, there is a first-order query Q such that for any constructible unbounded-treewidth family $\mathcal{I}$ of probabilistic graphs, the PQE problem for Q and $\mathcal{I}$ is #P-hard under RP reductions

- **Family:** an infinite set of graphs allowed as input (with arbitrary probabilities) so in particular **closed under subgraphs**
- **Unbounded-treewidth**: for all $k \in \mathbb{N}$, there is $I_k \in \mathcal{I}$ of treewidth $\geq k$
- **Constructible:** given k, we can **compute** such an instance I_k in PTIME

Why is this a dichotomy? Where's the lower bound?

Theorem (Amarilli, Bourhis, and Senellart 2016)

For any set of edge colors, there is a first-order query $\mathbf{Q}$ such that for any constructible unbounded-treewidth family $\mathcal{I}$ of probabilistic graphs, the PQE problem for $\mathbf{Q}$ and $\mathcal{I}$ is #P-hard under RP reductions

- **Family:** an infinite set of graphs allowed as input (with arbitrary probabilities) so in particular **closed under subgraphs**
- **Unbounded-treewidth**: for all $k \in \mathbb{N}$, there is $I_k \in \mathcal{I}$ of treewidth $\geq k$
- **Constructible:** given k, we can **compute** such an instance I_k in PTIME
- **Under RP reductions:** reduce in PTIME with high probability

Why is this a dichotomy? Where's the lower bound?

Theorem (Amarilli, Bourhis, and Senellart 2016)

For any set of edge colors, there is a first-order query $\mathbf{Q}$ such that for any constructible unbounded-treewidth family $\mathcal{I}$ of probabilistic graphs, the PQE problem for $\mathbf{Q}$ and $\mathcal{I}$ is #P-hard under RP reductions

- **Family:** an infinite set of graphs allowed as input (with arbitrary probabilities) so in particular **closed under subgraphs**
- **Unbounded-treewidth**: for all $k \in \mathbb{N}$, there is $I_k \in \mathcal{I}$ of treewidth $\geq k$
- **Constructible:** given k, we can **compute** such an instance I_k in PTIME
- **Under RP reductions:** reduce in PTIME with high probability

$\rightarrow$ This result does **not** generalize to arity-two!

Why is this a dichotomy? Where's the lower bound?

Theorem (Amarilli, Bourhis, and Senellart 2016)

For any set of edge colors, there is a first-order query $\mathbf{Q}$ such that for any constructible unbounded-treewidth family $\mathcal{I}$ of probabilistic graphs, the PQE problem for $\mathbf{Q}$ and $\mathcal{I}$ is #P-hard under RP reductions

- **Family:** an infinite set of graphs allowed as input (with arbitrary probabilities) so in particular **closed under subgraphs**
- **Unbounded-treewidth**: for all $k \in \mathbb{N}$, there is $I_k \in \mathcal{I}$ of treewidth $\geq k$
- **Constructible:** given k, we can **compute** such an instance I_k in PTIME
- **Under RP reductions:** reduce in PTIME with high probability

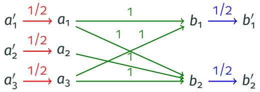
$\rightarrow$ This result does **not** generalize to arity-two!

$\rightarrow$ Proof idea: **extract wall graphs as topological minors** (Chekuri and Chuzhoy 2014) and adapt a technique of Ganian, Hlineny, Langer, Obdrzalek, Rossmanith, and

Proof technique

Hard part: show hardness for (variants of) the query Q: $x \longrightarrow y \longrightarrow z \longrightarrow w$

We reduce from $\mathrm{PQE}(Q)$, on **probabilistic graphs** G of the following form:

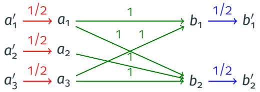


Task: count the number X of **red-blue edge subsets** that **violate** Q

Proof technique

Hard part: show hardness for (variants of) the query Q: $x \longrightarrow y \longrightarrow z \longrightarrow w$

We reduce from $\mathrm{PQE}(Q)$, on **probabilistic graphs** G of the following form:



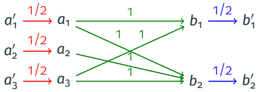
Task: count the number X of **red-blue edge subsets** that **violate** Q

- Split the **subsets** on some **parameter** e.g., the number of nodes: $X = X_1 + \cdots + X_k$

Proof technique

Hard part: show hardness for (variants of) the query Q: $x \longrightarrow y \longrightarrow z \longrightarrow w$

We reduce from $\mathrm{PQE}(Q)$, on **probabilistic graphs** G of the following form:



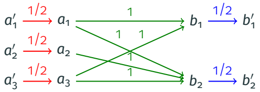
Task: count the number X of **red-blue edge subsets** that **violate** Q

- Split the **subsets** on some **parameter** e.g., the number of nodes: $X = X_1 + \cdots + X_k$
- Create unweighted copies of G modified with some **parameterized gadgets**
  - $\rightarrow$ Call the **oracle** for $\mathrm{SC(Q)}$ on each to get answers $N_1, \ldots, N_k$

Proof technique

Hard part: show hardness for (variants of) the query Q: $x \xrightarrow{} y \xrightarrow{} z \xrightarrow{} w$

We reduce from $\mathrm{PQE}(Q)$, on **probabilistic graphs** G of the following form:



Task: count the number X of **red-blue edge subsets** that **violate** Q

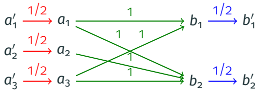
- Split the **subsets** on some **parameter** e.g., the number of nodes: $X = X_1 + \cdots + X_k$
- Create unweighted copies of G modified with some **parameterized gadgets**
  $\rightarrow$ Call the **oracle** for $\mathrm{SC}(\mathrm{Q})$ on each to get answers $N_1, \ldots, N_k$
- Show that each N_i is a **linear function** of $X_1, \ldots, X_k$, so:

$$\begin{pmatrix} N_1 \\ \vdots \\ N_k \end{pmatrix} = \begin{pmatrix} \alpha_{1,1} & \cdots & \alpha_{1,k} \\ \vdots & \ddots & \vdots \\ \alpha_{k,1} & \cdots & \alpha_{k,k} \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ \vdots \\ X_k \end{pmatrix}$$

Proof technique

Hard part: show hardness for (variants of) the query Q: $x \longrightarrow y \longrightarrow z \longrightarrow w$

We reduce from $\mathrm{PQE}(Q)$, on **probabilistic graphs** G of the following form:



Task: count the number X of **red-blue edge subsets** that **violate** Q

- Split the **subsets** on some **parameter** e.g., the number of nodes: $X = X_1 + \cdots + X_k$
- Create unweighted copies of G modified with some **parameterized gadgets**
  - $\rightarrow$ Call the **oracle** for $\mathrm{SC}(\mathrm{Q})$ on each to get answers $N_1, \ldots, N_k$
- Show that each N_i is a **linear function** of $X_1, \ldots, X_k$, so:

$$\begin{pmatrix} N_1 \\ \vdots \\ N_k \end{pmatrix} = \begin{pmatrix} \alpha_{1,1} & \cdots & \alpha_{1,k} \\ \vdots & \ddots & \vdots \\ \alpha_{k,1} & \cdots & \alpha_{k,k} \end{pmatrix} \cdot \begin{pmatrix} X_1 \\ \vdots \\ X_k \end{pmatrix}$$

- Show **invertibility** of this matrix to recover the X_i from the N_i

Advertisement: TCS4F and "No free view? No review!"

Are you concerned about how academic research in theoretical computer science is contributing to the climate crisis?

If so, sign the TCS4F pledge! (Theoretical Computer Scientists 4 Future)

`www.tcs4f.org`

(with Thomas Schwentick, Thomas Colcombet, Hugo Férée)

TCS4F
THEORETICAL
COMPUTER
SCIENTISTS
FOR FUTURE

Advertisement: TCS4F and “No free view? No review!”

Are you concerned about how academic research in theoretical computer science is contributing to the climate crisis?

If so, **sign the TCS4F pledge**! (Theoretical Computer Scientists 4 Future)

`www.tcs4f.org`

(with Thomas Schwentick, Thomas Colcombet, Hugo Férée)

TCS4F
THEORETICAL
COMPUTER
SCIENTISTS
FOR FUTURE

NO FREE VIEW?
NO REVIEW!

Are you tired of doing reviewing work for conferences and journals that do not publish their research online?

If so, **sign the pledge “No free view? No review!”**

`www.nofreeviewnoreview.org`

(with Antonin Delpeuch)