Probabilistic Databases: Width-Based Approaches

Antoine Amarilli

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

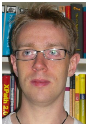
*OK, PQE is **intractable** for essentially all queries. What now?*

- 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 (A., Bourhis, Senellart, 2015, 2016)

*Fix a bound $k \in \mathbb{N}$ and fix a Boolean **monadic second-order** query Q. Then $\mathrm{PQE}(Q)$ is in **PTIME** on input TID instances of **treewidth** $\leq k$*



Going back to more restricted instances

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

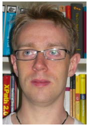
- We could restrict the **structure** of instances: instead of arbitrary graphs, focus on:
  - probabilistic **words**
  - probabilistic **trees**
  - probabilistic graphs with **bounded treewidth**
- In the non-probabilistic case, this ensures tractability for **complex queries**

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

Theorem (A., Bourhis, Senellart, 2015, 2016)

*Fix a bound $k \in \mathbb{N}$ and fix a Boolean **monadic second-order** query Q. Then $\mathrm{PQE}(Q)$ is in **PTIME** on input TID instances of **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)*



Non-probabilistic query evaluation on words

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

Non-probabilistic query evaluation on words

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

Query Q: a **sentence** (yes/no question) in **monadic second-order logic** (MSO)

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

Non-probabilistic query evaluation on words

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

Query Q: a **sentence** (yes/no question) in **monadic second-order logic** (MSO)

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

Result: TRUE/FALSE indicating if the word w satisfies the query Q

Non-probabilistic query evaluation on words

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

Query Q: a **sentence** (yes/no question) in **monadic second-order logic** (MSO)

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

Result: TRUE/FALSE indicating if the word w satisfies the query Q

Computational complexity as a function of w
(the query Q is **fixed**)

Monadic second-order logic (MSO)

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

Monadic second-order logic (MSO)

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

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

Monadic second-order logic (MSO)

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

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

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

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

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

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

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

Word automata

Translate the query Q to a deterministic word automaton

Alphabet: 

w: 

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

Word automata

Translate the query Q to a deterministic word automaton

Alphabet: 

w: 

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

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

Word automata

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

Alphabet: 

w: 

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

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

Word automata

Translate the query Q to a deterministic word automaton

Alphabet:  w:  Q: $\exists x\, y\; P(x) \wedge P(y)$

- States: $\{\perp, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: $\perp$ P B

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

Alphabet: 

w: 

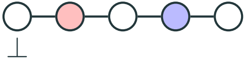
Q: $\exists x\, y\ P_{\circ}(x) \wedge P_{\circ}(y)$

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

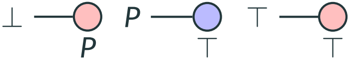
Word automata

Translate the query Q to a deterministic word automaton

Alphabet: 

w: 

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

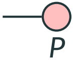
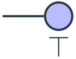
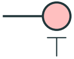
- States: $\{\perp, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function:  $\perp$  P  B
- Transitions (examples): 

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

Alphabet: 

w: 

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

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

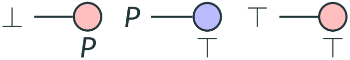
Translate the query Q to a **deterministic word automaton**

Alphabet: 

w: 

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

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

- **Transitions** (examples): 

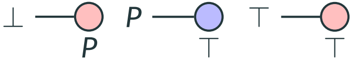
Word automata

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

Alphabet: 

w: 

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

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

Word automata

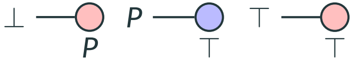
Translate the query Q to a **deterministic word automaton**

Alphabet: 

w: 

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

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

- **Transitions** (examples): 

Word automata

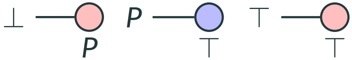
Translate the query Q to a **deterministic word automaton**

Alphabet: 

w: 

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

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

- **Transitions** (examples): 

Theorem (Büchi, 1960)

*MSO and **word automata** have the same **expressive power** on words*

Word automata

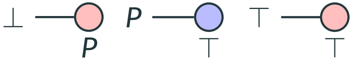
Translate the query Q to a deterministic word automaton

Alphabet: 

w: 

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

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

- Transitions (examples): 

Theorem (Büchi, 1960)

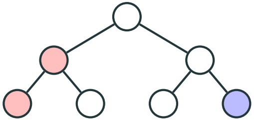
MSO and word automata have the same expressive power on words

Corollary

Query evaluation of MSO on words is in linear time (in data complexity)

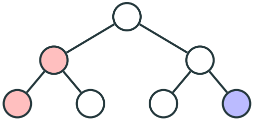
Non-probabilistic query evaluation on trees

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



Non-probabilistic query evaluation on trees

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



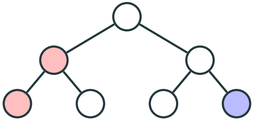
Query Q: in monadic second-order logic (MSO)

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

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

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

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



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"

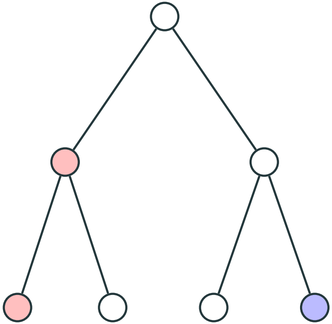
"Is there both a pink and a blue node?"

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

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

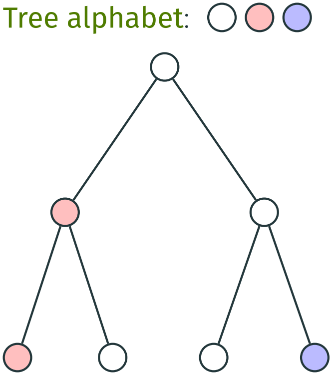
Tree automata

Tree alphabet:



Tree automata

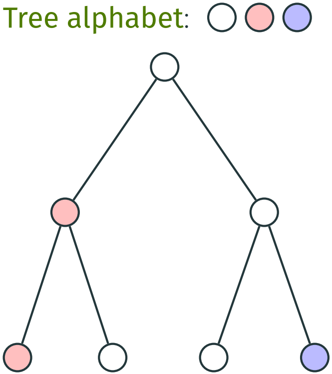
Tree alphabet:



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

Tree automata

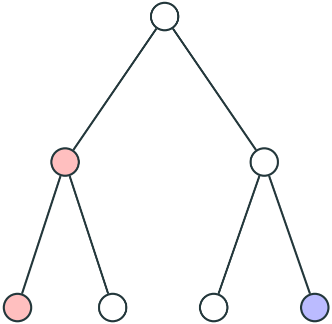
Tree alphabet:



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

Tree automata

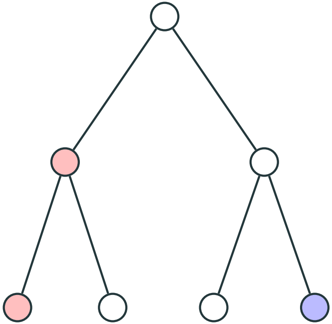
Tree alphabet: 



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

Tree automata

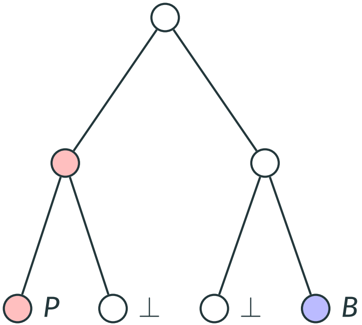
Tree alphabet:



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

Tree automata

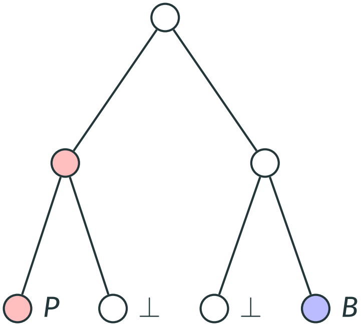
Tree alphabet:



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

Tree automata

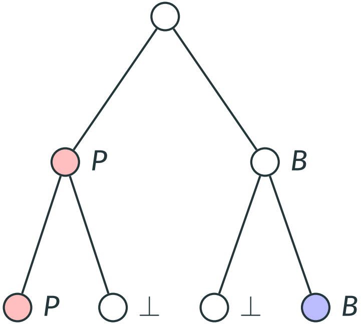
Tree alphabet:



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

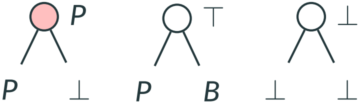
Tree automata

Tree alphabet: 



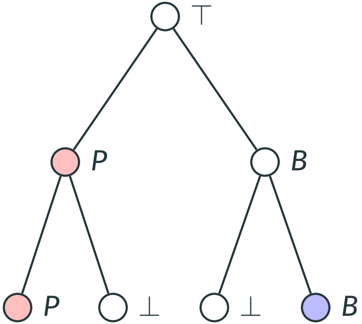
- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** 

- **Transitions** (examples):



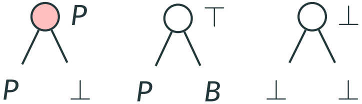
Tree automata

Tree alphabet: 



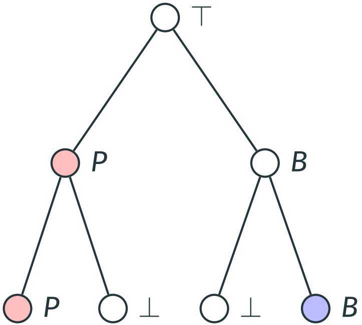
- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** 

- **Transitions** (examples):



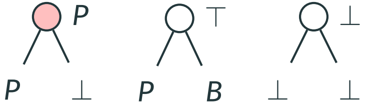
Tree automata

Tree alphabet: 



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

- **Transitions** (examples):



Theorem ([Thatcher and Wright, 1968])

MSO and tree automata have the same expressive power on trees

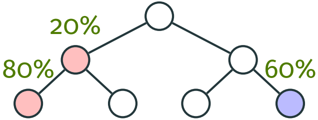
Probabilistic query evaluation on trees

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:

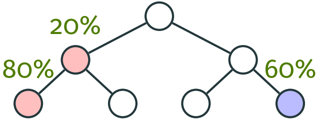
Database: a tree T where each node has a probability of keeping its color (vs taking the default color ◯)



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** ◯)



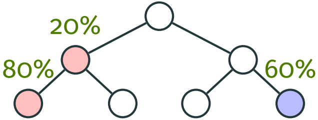
Query Q: in monadic second-order logic (MSO)

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

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** ◯)



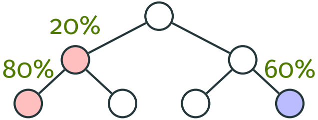
Query Q: in monadic second-order logic (MSO)

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

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** $\bigcirc$)

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

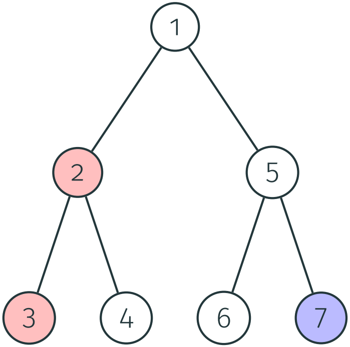
$\exists x\, y\; P_{\bigcirc}(x) \land P_{\bigcirc}(y)$

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

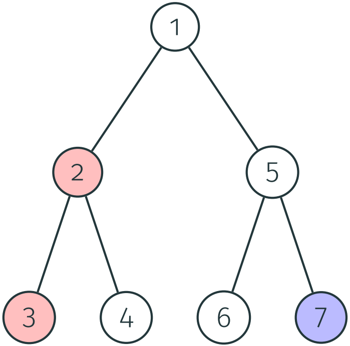
Theorem

*For any fixed **MSO query** Q, the problem $\mathrm{PQE}(Q)$ on trees is in **linear time** assuming constant-time arithmetics*

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

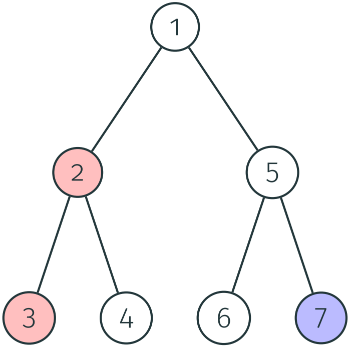


Uncertain trees: capturing how the query result depends on the choices



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

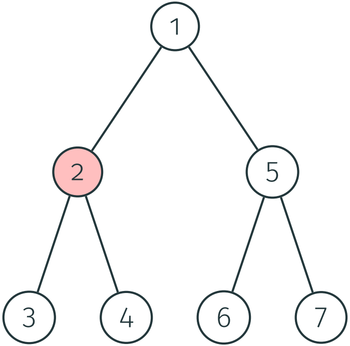
Uncertain trees: capturing how the query result depends on the choices



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

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

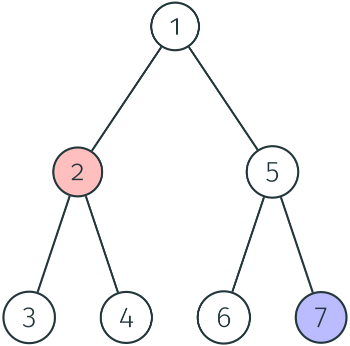
Uncertain trees: capturing how the query result depends on the choices



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

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

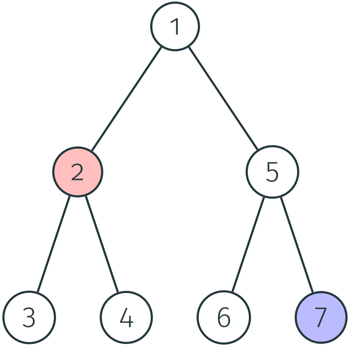
Uncertain trees: capturing how the query result depends on the choices



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

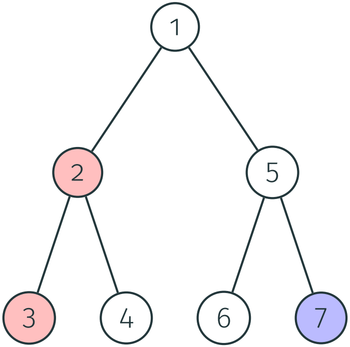


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

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



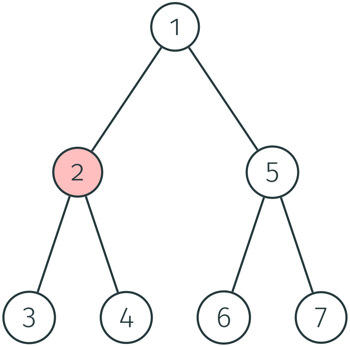
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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



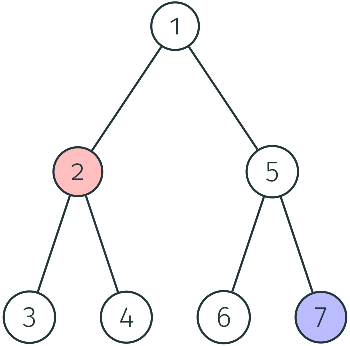
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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



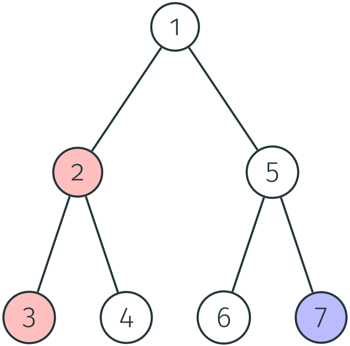
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

The query *Q* returns YES

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

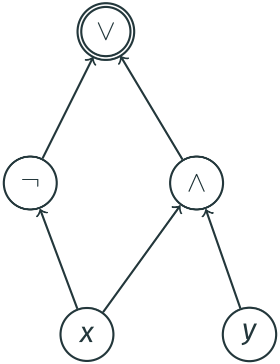


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

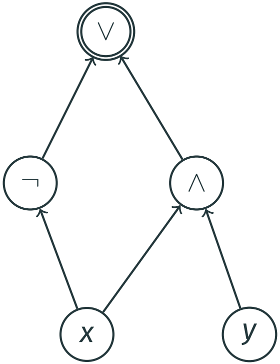
$\rightarrow$ This is a so-called **Boolean provenance circuit** on the "color facts" of the tree nodes!

Boolean circuit



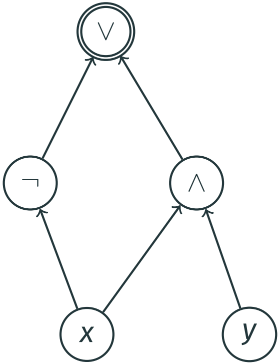
- Directed acyclic graph of gates

Boolean circuit



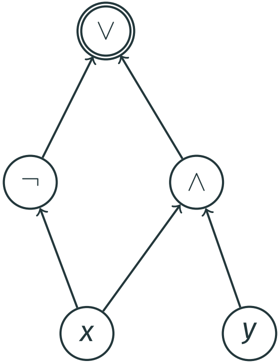
- Directed acyclic graph of **gates**
- **Output** gate: 

Boolean circuit



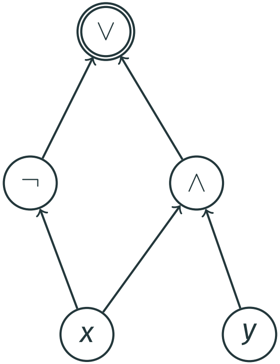
- Directed acyclic graph of **gates**
- **Output** gate: 
- **Variable** gates: 

Boolean circuit



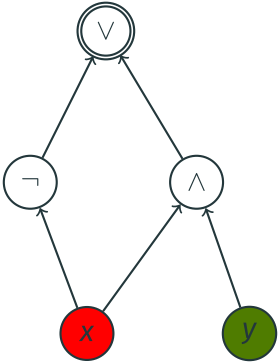
- Directed acyclic graph of **gates**
- **Output** gate:
- **Variable** gates: x
- **Internal** gates: $\vee$ $\wedge$ $\neg$

Boolean circuit



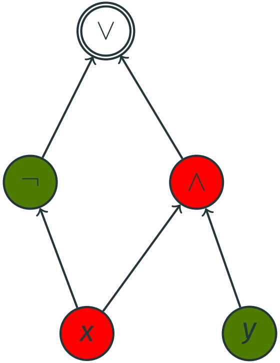
- Directed acyclic graph of **gates**
- **Output** gate:
- **Variable** gates: x
- **Internal** gates: $\lor$ $\land$ $\neg$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$...

Boolean circuit



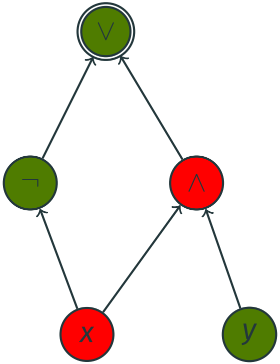
- Directed acyclic graph of **gates**
- **Output** gate:
- **Variable** gates: x
- **Internal** gates: $\vee$ $\wedge$ $\neg$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$...

Boolean circuit



- Directed acyclic graph of **gates**
- **Output** gate:
- **Variable** gates: x
- **Internal** gates: $\lor$ $\land$ $\neg$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$...

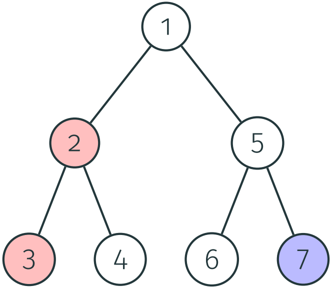
Boolean circuit



- Directed acyclic graph of **gates**
- **Output** gate:
- **Variable** gates: x
- **Internal** gates: $\vee$ $\wedge$ $\neg$
- **Valuation**: function from variables to $\{0, 1\}$

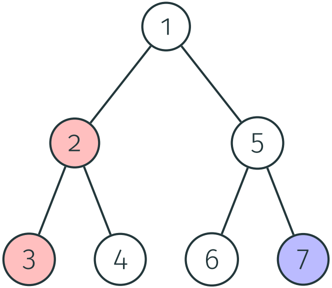
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$... mapped to 1

Example: Provenance circuit



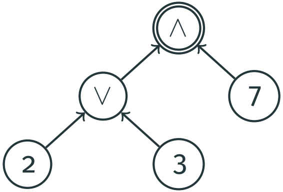
Query: *Is there both a pink and a blue node?*

Example: Provenance circuit

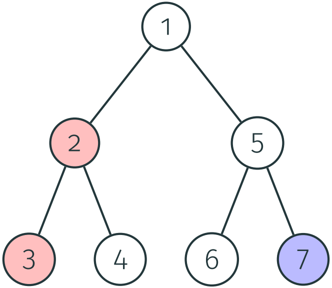


Query: *Is there both a pink and a blue node?*

Provenance circuit:

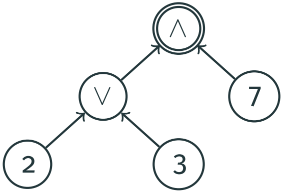


Example: Provenance circuit



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

Building provenance circuits on trees

Theorem

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

Building provenance circuits on trees

Theorem

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

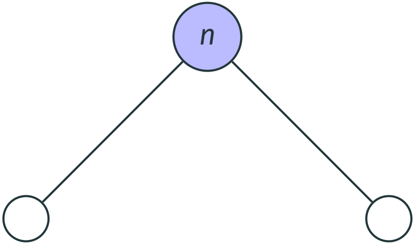
- **Alphabet:**
- **Automaton:** *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

Building provenance circuits on trees

Theorem

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

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

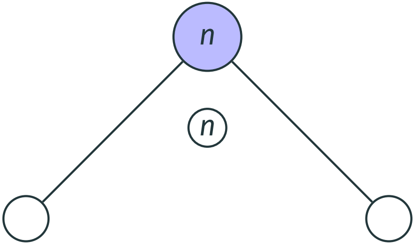


Building provenance circuits on trees

Theorem

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

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

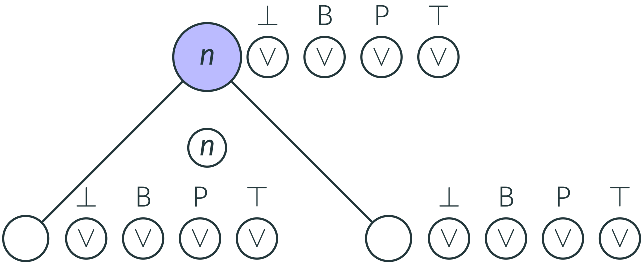


Building provenance circuits on trees

Theorem

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

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

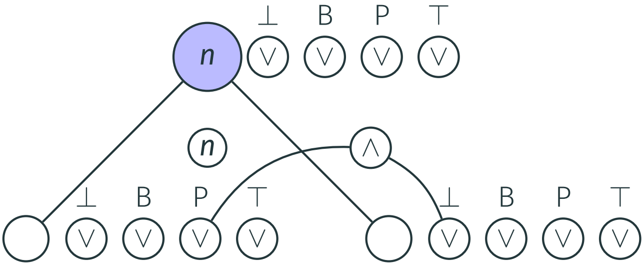


Building provenance circuits on trees

Theorem

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

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

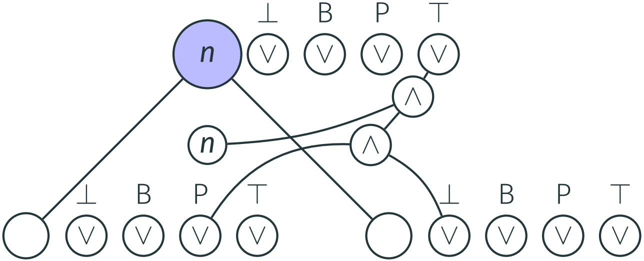


Building provenance circuits on trees

Theorem

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

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

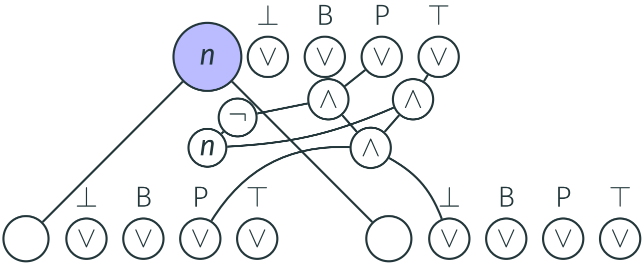


Building provenance circuits on trees

Theorem

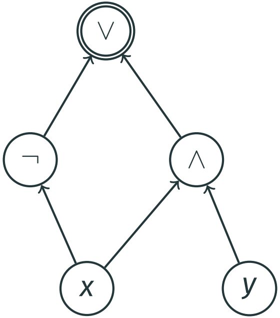
For any bottom-up ***tree automaton*** A *and input* ***tree*** T*,*
we can build a ***provenance circuit*** *of* A *on* T *in* $O(|A| \times |T|)$

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



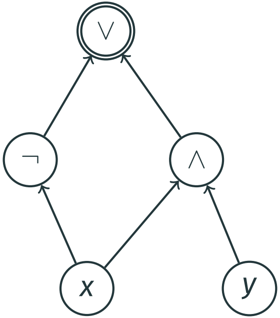
Computing the probability of a circuit

- We are given a circuit and a probability P for each variable



Computing the probability of a circuit

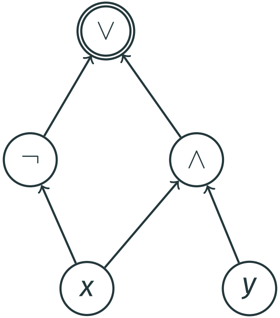
- We are given a circuit and a probability P for each variable



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

Computing the probability of a circuit

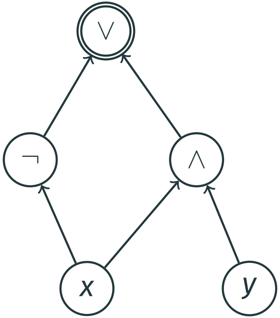
- We are given a **circuit** and a **probability** P for each variable
- Each variable x is true **independently** with probability $P(x)$



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

Computing the probability of a circuit

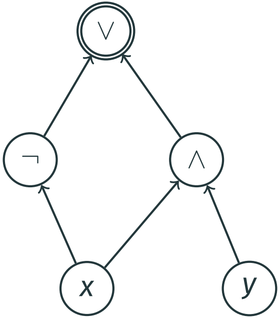
- We are given a circuit and a probability P for each variable
- Each variable x is true independently with probability $P(x)$
- What is the probability that the circuit evaluates to true?



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

Computing the probability of a circuit

- We are given a **circuit** and a **probability** P for each variable
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?

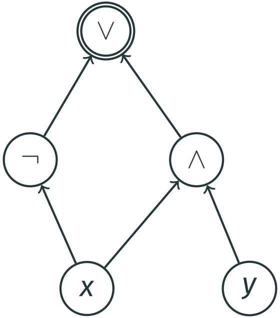


- $P(x) = 40\%$
- $P(y) = 50\%$

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

Computing the probability of a circuit

- We are given a **circuit** and a **probability** P for each variable
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?

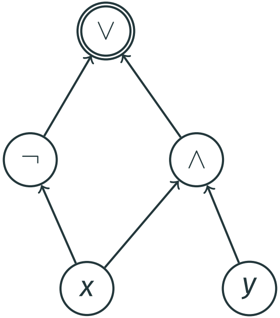


- $P(x) = 40\%$
- $P(y) = 50\%$

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

Computing the probability of a circuit

- We are given a **circuit** and a **probability** P for each variable
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?

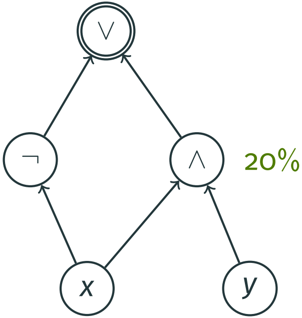


- $P(x) = 40\%$
- $P(y) = 50\%$

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

Computing the probability of a circuit

- We are given a **circuit** and a **probability** P for each variable
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?

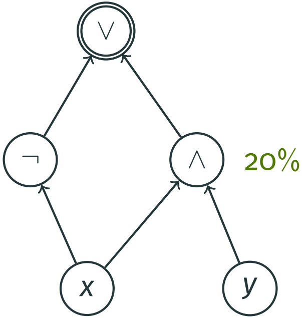


- $P(x) = 40\%$
- $P(y) = 50\%$

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

Computing the probability of a circuit

- We are given a **circuit** and a **probability** P for each variable
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?

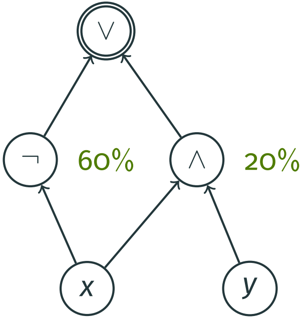


- $P(x) = 40\%$
- $P(y) = 50\%$

- 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 a circuit

- We are given a **circuit** and a **probability** P for each variable
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?

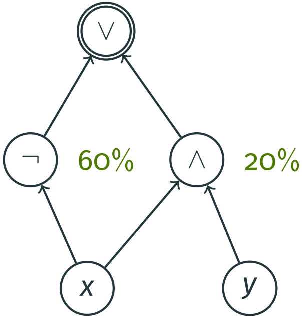


- $P(x) = 40\%$
- $P(y) = 50\%$

- 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 a circuit

- We are given a **circuit** and a **probability** P for each variable
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?

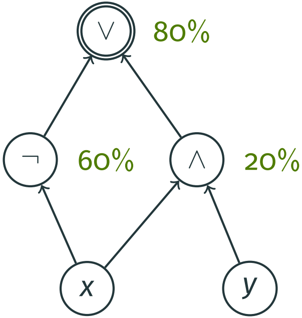


- $P(x) = 40\%$
- $P(y) = 50\%$

- 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 a circuit

- We are given a **circuit** and a **probability** P for each variable
- Each variable x is true **independently** with probability $P(x)$
- What is the probability that the circuit **evaluates to true**?

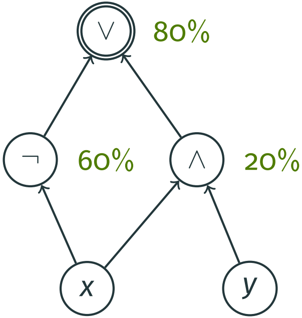


- $P(x) = 40\%$
- $P(y) = 50\%$

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

Computing the probability of a circuit

- We are given a **circuit** and a **probability** P for each variable
- 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 **$\land$-gate** are **independent**
  - The **$\neg$-gate** has probability **$1 - P(\text{input})$**
  - The **$\lor$-gate** has **mutually exclusive** inputs
- Let's focus on a **restricted class** of circuits that satisfies these conditions

d-DNNFs

The circuit is a d-DNNF...

d-DNNFs

The circuit is a d-DNNF...

- $\neg$ gates only have variables as inputs

d-DNNFs

The circuit is a **d-DNNF**...

- ¬ gates only have **variables** as inputs
- ∨ gates always have **mutually exclusive** inputs

d-DNNFs

The circuit is a d-DNNF...

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

d-DNNFs

The circuit is a d-DNNF…

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

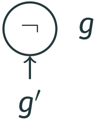
… so probability computation is easy!

d-DNNFs

The circuit is a d-DNNF…

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

… so probability computation is easy!

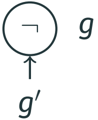


d-DNNFs

The circuit is a d-DNNF...

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

... so probability computation is easy!



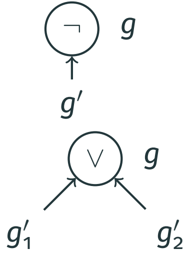
$$P(g) := 1 - P(g')$$

d-DNNFs

The circuit is a d-DNNF...

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

... so probability computation is **easy**!



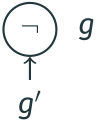
$$P(g) := 1 - P(g')$$

d-DNNFs

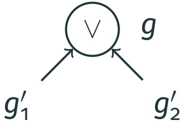
The circuit is a d-DNNF...

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

... so probability computation is easy!



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



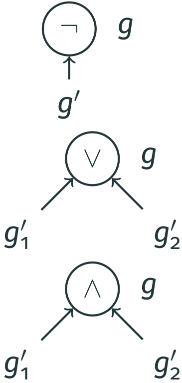
$$P(g) := P(g'_1) + P(g'_2)$$

d-DNNFs

The circuit is a d-DNNF…

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

… so probability computation is **easy**!



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

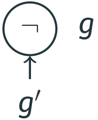
$$P(g) := P(g'_1) + P(g'_2)$$

d-DNNFs

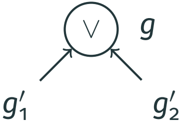
The circuit is a d-DNNF…

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

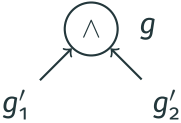
… so probability computation is **easy**!



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



$$P(g) := P(g_1') + P(g_2')$$



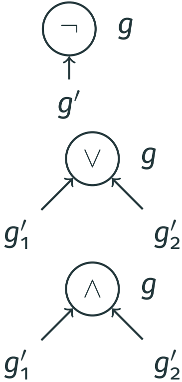
$$P(g) := P(g_1') \times P(g_2')$$

d-DNNFs

The circuit is a d-DNNF…

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

… so probability computation is easy!



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

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

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

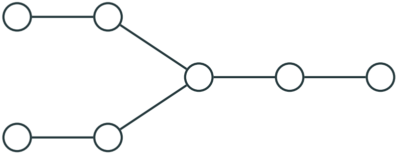
Lemma

The provenance circuit computed in our construction is a d-DNNF

Treewidth

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

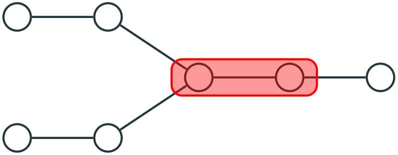
Treewidth by example:



Treewidth

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

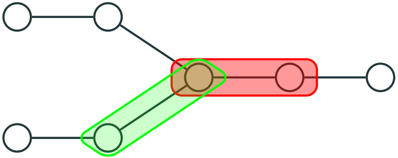
Treewidth by example:



Treewidth

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

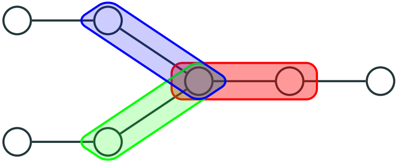
Treewidth by example:



Treewidth

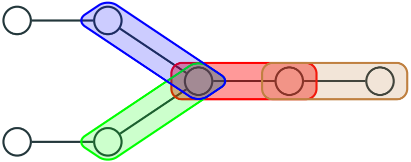
We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:



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

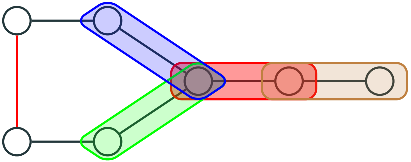
Treewidth by example:



Treewidth

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

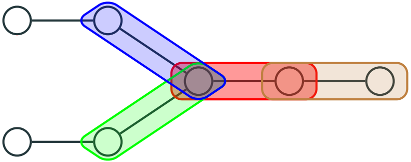
Treewidth by example:

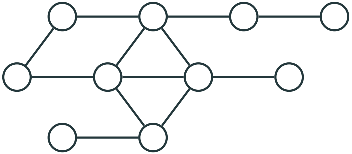


Treewidth

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

Treewidth by example:

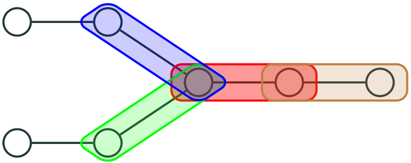


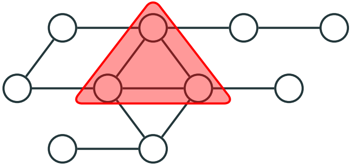


Treewidth

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

Treewidth by example:

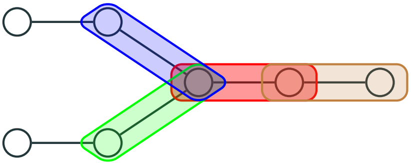


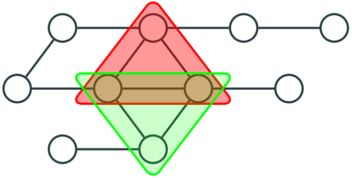


Treewidth

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

Treewidth by example:

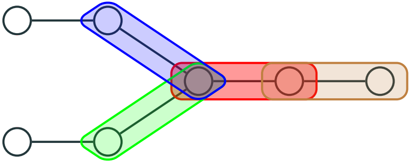


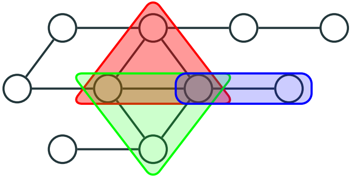


Treewidth

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

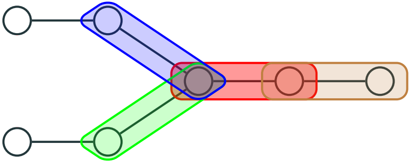
Treewidth by example:

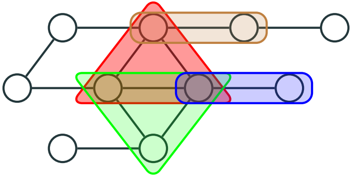




We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:

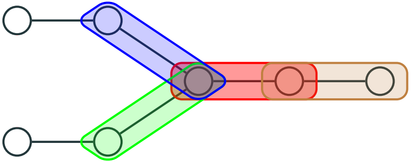


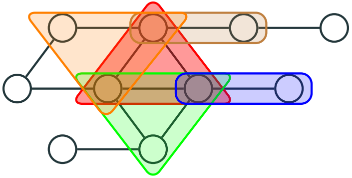


Treewidth

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

Treewidth by example:

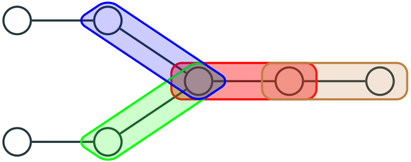


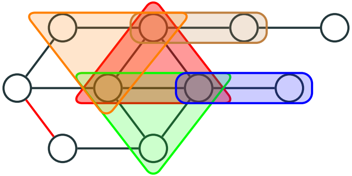


Treewidth

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

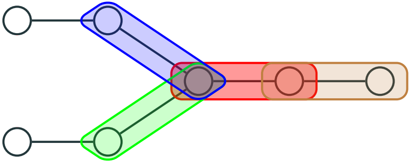
Treewidth by example:

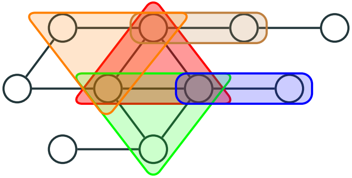




We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

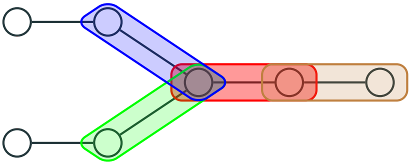
Treewidth by example:

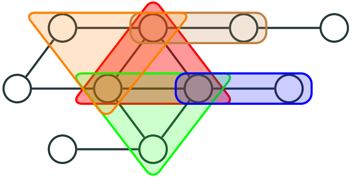




We have shown tractability of PQE on **trees**; let us extend to **bounded treewidth**

Treewidth by example:



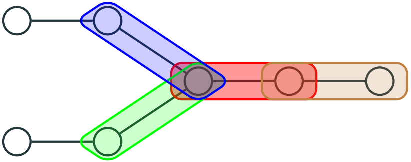


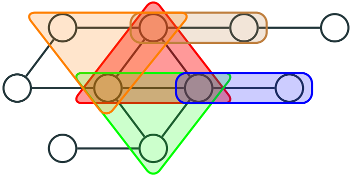
- **Trees** have treewidth 1
- **Cycles** have treewidth 2
- **k-cliques** and **$(k-1)$-grids** have treewidth $k-1$

Treewidth

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

Treewidth by example:



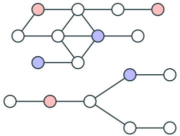


- **Trees** have treewidth **1**
- **Cycles** have treewidth **2**
- **k-cliques** and **$(k-1)$-grids** have treewidth $k-1$

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

Courcelle's theorem and extension to PQE

Treelike **data**

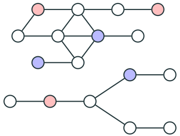


MSO query

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

Courcelle's theorem and extension to PQE

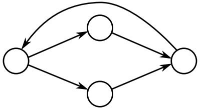
Treelike **data**



MSO query

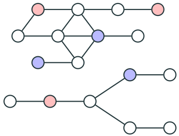
$$\exists x\, y \quad P_{\circ}(x) \wedge P_{\circ}(y)$$

Tree **automaton**



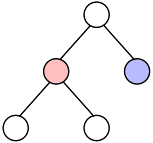
Courcelle's theorem and extension to PQE

Treelike **data**



linear →

Tree **encoding**

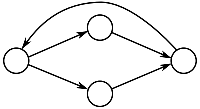


MSO query

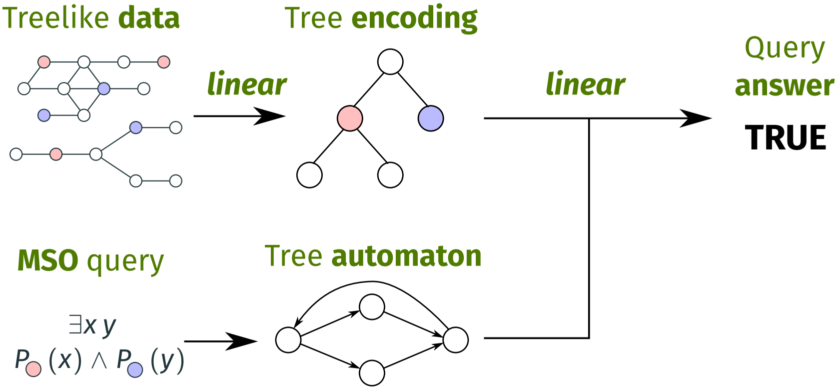
$\exists x\, y$

$P_{\circ}(x) \wedge P_{\circ}(y)$ →

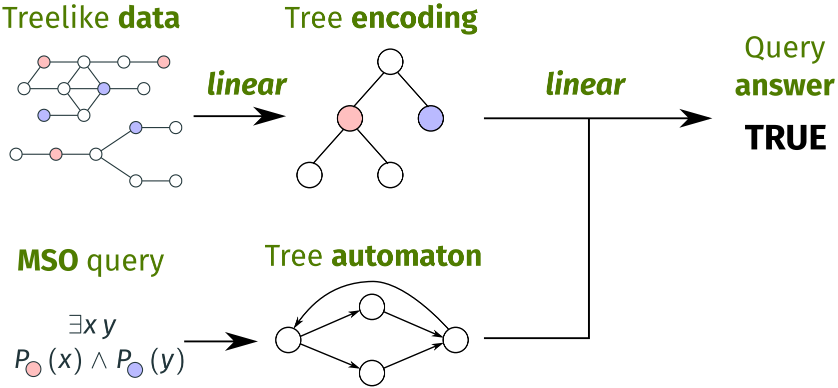
Tree **automaton**



Courcelle's theorem and extension to PQE



Courcelle's theorem and extension to PQE

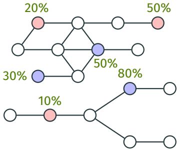


Theorem ([Courcelle, 1990])

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

Courcelle's theorem and extension to PQE

Probabilistic treelike **data**

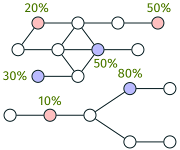


MSO query

$$\exists x\, y \quad P_{\text{red}}(x) \land P_{\text{blue}}(y)$$

Courcelle's theorem and extension to PQE

Probabilistic treelike **data**

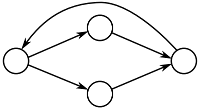


MSO query

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

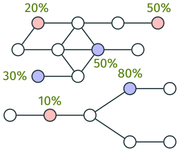
$\longrightarrow$

Tree **automaton**



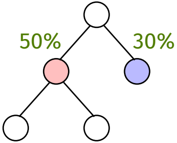
Courcelle's theorem and extension to PQE

Probabilistic treelike **data**



linear

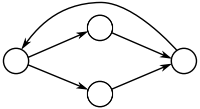
Probabilistic tree **encoding**



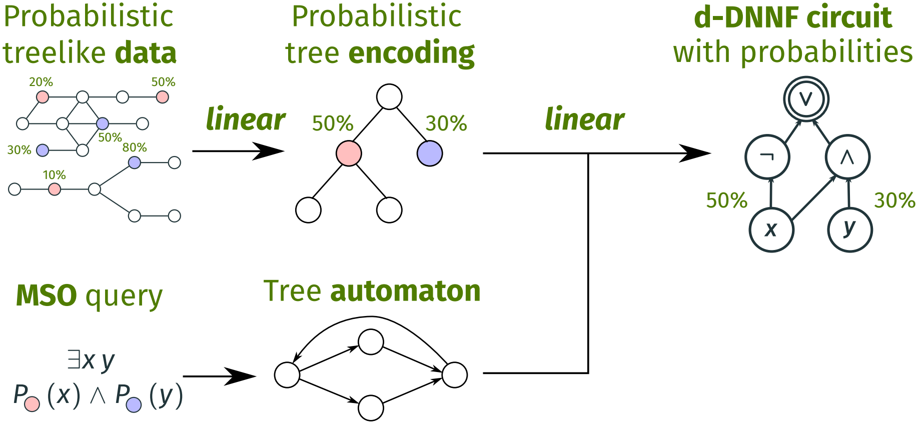
MSO query

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

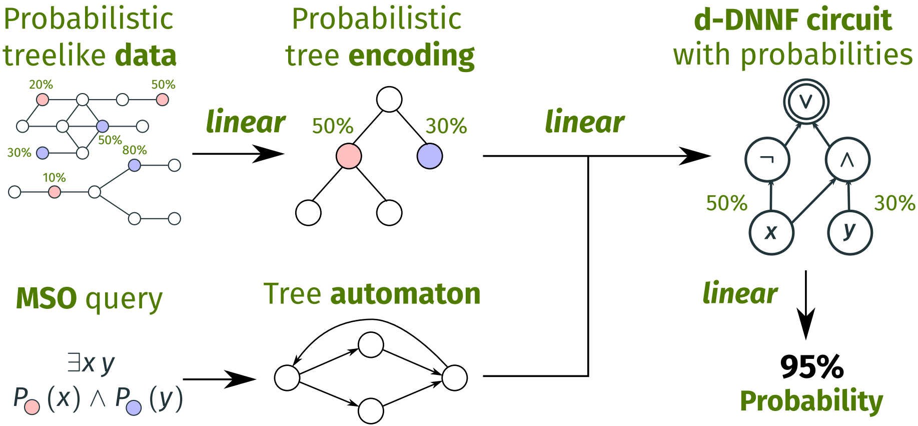
Tree **automaton**



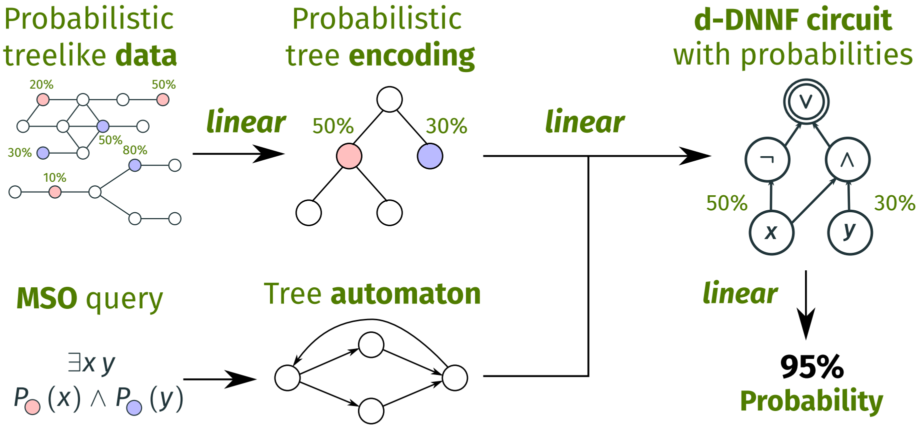
Courcelle's theorem and extension to PQE



Courcelle's theorem and extension to PQE



Courcelle's theorem and extension to PQE



Theorem (A., Bourhis, Senellart, 2015, 2016)

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

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

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

Theorem (A., Bourhis, Senellart, 2016)

For any arity-two signature, 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*

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

Theorem (A., Bourhis, Senellart, 2016)

For any arity-two signature, 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 (A., Bourhis, Senellart, 2016)

For any arity-two signature, 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 (A., Bourhis, Senellart, 2016)

For any arity-two signature, 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 (A., Bourhis, Senellart, 2016)

For any arity-two signature, 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
- **Under RP reductions:** reduce in PTIME with high probability

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

Theorem (A., Bourhis, Senellart, 2016)

*For any arity-two signature, 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
- **Under RP reductions:** reduce in PTIME with high probability

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

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

Theorem (A., Bourhis, Senellart, 2016)

For any arity-two signature, 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 higher-arity!

$\rightarrow$ Proof idea: **extract wall graphs as topological minors** ([Chekuri and Chuzhoy, 2014]) and use them for a lower bound

References i

Amarilli, A., Bourhis, P., and Senellart, P. (2015).
Provenance circuits for trees and treelike instances.
In *ICALP*.

Amarilli, A., Bourhis, P., and Senellart, P. (2016).
Tractable lineages on treelike instances: Limits and extensions.
In *PODS*.

Chekuri, C. and Chuzhoy, J. (2014).
Polynomial bounds for the grid-minor theorem.
In *STOC*.

References ii

Courcelle, B. (1990).
The monadic second-order logic of graphs. I. Recognizable sets of finite graphs.
Inf. Comput., 85(1).

Thatcher, J. W. and Wright, J. B. (1968).
Generalized finite automata theory with an application to a decision problem of second-order logic.
Mathematical systems theory, 2(1).