Provenance Circuits for Trees and Treelike Instances

Antoine Amarilli[1], Pierre Bourhis[2], Pierre Senellart[1,3]

[1]Télécom ParisTech

[2]CNRS-LIFL

[3]National University of Singapore

July 10th, 2015

General idea

- We consider a query and a relational instance
- Often it is not sufficient to merely evaluate the query:
  - $\rightarrow$ We need quantitative information
  - $\rightarrow$ We need the link from the output to the input data

General idea

- We consider a query and a relational instance
- Often it is not sufficient to merely evaluate the query:
  - $\rightarrow$ We need quantitative information
  - $\rightarrow$ We need the link from the output to the input data

$\rightarrow$ Compute query provenance!

Example 1: security for a conjunctive query

- Consider the conjunctive query: $\exists xyz\ R(x, y) \wedge R(y, z)$
- Consider the relational instance below:

R	
a	b
b	c
d	e
e	d
f	f

Example 1: security for a conjunctive query

- Consider the conjunctive query: $\exists xyz\ R(x, y) \wedge R(y, z)$
- Consider the relational instance below:

R	
a	b
b	c
d	e
e	d
f	f

- Result: true

Example 1: security for a conjunctive query

- Consider the conjunctive query: $\exists xyz\ R(x, y) \wedge R(y, z)$
- Consider the relational instance below:

	R	
a	b	Public
b	c	Secret
d	e	Confidential
e	d	Confidential
f	f	Top secret

- Result: true
- Add security annotations: Public, Confidential, Secret, Top secret, Never available

Example 1: security for a conjunctive query

- Consider the conjunctive query: $\exists xyz\ R(x, y) \wedge R(y, z)$
- Consider the relational instance below:

	R	
a	b	Public
b	c	Secret
d	e	Confidential
e	d	Confidential
f	f	Top secret

- Result: true
- Add security annotations: Public, Confidential, Secret, Top secret, Never available
- What is the minimal security clearance required?

Example 1: security for a conjunctive query

- Consider the conjunctive query: $\exists xyz\ R(x,y) \land R(y,z)$
- Consider the relational instance below:

R		
a	b	Public
b	c	Secret
d	e	Confidential
e	d	Confidential
f	f	Top secret

- Result: true
- Add security annotations: Public, Confidential, Secret, Top secret, Never available
- What is the minimal security clearance required?

Example 1: security for a conjunctive query

- Consider the conjunctive query: $\exists xyz\, R(x,y) \wedge R(y,z)$
- Consider the relational instance below:

R		
a	b	Public
b	c	Secret
d	e	Confidential
e	d	Confidential
f	f	Top secret

- Result: true
- Add security annotations: Public, Confidential, Secret, Top secret, Never available
- What is the minimal security clearance required?

Example 1: security for a conjunctive query

- Consider the conjunctive query: $\exists xyz\, R(x, y) \wedge R(y, z)$
- Consider the relational instance below:

R		
a	b	Public
b	c	Secret
d	e	Confidential
e	d	Confidential
f	f	Top secret

- Result: true
- Add security annotations: Public, Confidential, Secret, Top secret, Never available
- What is the minimal security clearance required?

Example 1: security for a conjunctive query

- Consider the conjunctive query: $\exists xyz\, R(x, y) \wedge R(y, z)$
- Consider the relational instance below:

R		
a	b	Public
b	c	Secret
d	e	Confidential
e	d	Confidential
f	f	Top secret

- Result: true
- Add security annotations: Public, Confidential, Secret, Top secret, Never available
- What is the minimal security clearance required?

$\rightarrow$ Result: Confidential

Example 2: bag queries

Consider again: $\exists xyz\, R(x,y) \wedge R(y,z)$.

R	
a	b
b	c
d	e
e	d
f	f

Example 2: bag queries

Consider again: $\exists xyz\ R(x, y) \land R(y, z)$.

R	
a	b
b	c
d	e
e	d
f	f

- Result: true

Example 2: bag queries

Consider again: $\exists xyz\ R(x,y) \wedge R(y,z)$.

	R	
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Result: true
- Add multiplicity annotations

Example 2: bag queries

Consider again: $\exists xyz\, R(x, y) \wedge R(y, z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Result: true
- Add multiplicity annotations
- How many query matches?

Example 2: bag queries

Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Result: true
- Add multiplicity annotations
- How many query matches?

$\rightarrow$ Result: 1

Example 2: bag queries

Consider again: $\exists xyz\ R(x,y) \wedge R(y,z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Result: true
- Add multiplicity annotations
- How many query matches?

$\rightarrow$ Result: $1 + 1$

Example 2: bag queries

Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Result: true
- Add multiplicity annotations
- How many query matches?

→ Result: $1 + 1 + 1$

Example 2: bag queries

Consider again: $\exists xyz\ R(x,y) \wedge R(y,z)$.

	R	
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Result: true
- Add multiplicity annotations
- How many query matches?

$\rightarrow$ Result: $1+1+1+1$

Example 2: bag queries

Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	1
b	c	1
d	e	1
e	d	1
f	f	1

- Result: true
- Add multiplicity annotations
- How many query matches?

→ Result: $1 + 1 + 1 + 1 = 4$

Example 3: uncertain facts

Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.

R	
a	b
b	c
d	e
e	d
f	f

Example 3: uncertain facts

Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.

R	
a	b
b	c
d	e
e	d
f	f

- Result: true

Example 3: uncertain facts

Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Result: true
- Assume facts are uncertain, give them atomic annotations

Example 3: uncertain facts

Consider again: $\exists xyz\, R(x, y) \wedge R(y, z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Result: true
- Assume facts are uncertain, give them atomic annotations
- For which subinstances does the query hold?

Example 3: uncertain facts

Consider again: $\exists xyz\ R(x, y) \land R(y, z)$.

	R	
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Result: true
- Assume facts are uncertain, give them atomic annotations
- For which subinstances does the query hold?

$\rightarrow$ Result: $(f_1 \land f_2)$

Example 3: uncertain facts

Consider again: $\exists xyz\ R(x,y) \wedge R(y,z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Result: true
- Assume facts are uncertain, give them atomic annotations
- For which subinstances does the query hold?

$\rightarrow$ Result: $(f_1 \wedge f_2) \vee (f_3 \wedge f_4)$

Example 3: uncertain facts

Consider again: $\exists xyz\ R(x,y) \wedge R(y,z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Result: true
- Assume facts are uncertain, give them atomic annotations
- For which subinstances does the query hold?

$\rightarrow$ Result: $(f_1 \wedge f_2) \vee (f_3 \wedge f_4)$

Example 3: uncertain facts

Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Result: true
- Assume facts are uncertain, give them atomic annotations
- For which subinstances does the query hold?

→ Result: $(f_1 \wedge f_2) \vee (f_3 \wedge f_4) \vee f_5$

Example 3: uncertain facts

Consider again: $\exists xyz\, R(x, y) \wedge R(y, z)$.

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

- Result: true
- Assume facts are uncertain, give them atomic annotations
- For which subinstances does the query hold?

→ Result: $(f_1 \wedge f_2) \vee (f_3 \wedge f_4) \vee f_5$

Example 4: the universal semiring $\mathbb{N}[X]$

- Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Annotate input facts with atomic annotations $X = f_1, \ldots, f_n$
- Most general semiring: $\mathbb{N}[X]$ of polynomials on X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

Example 4: the universal semiring $\mathbb{N}[X]$

- Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Annotate input facts with atomic annotations $X = f_1, \ldots, f_n$
- Most general semiring: $\mathbb{N}[X]$ of polynomials on X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

→ Result:

Example 4: the universal semiring $\mathbb{N}[X]$

- Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Annotate input facts with atomic annotations $X = f_1, \ldots, f_n$
- Most general semiring: $\mathbb{N}[X]$ of polynomials on X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

→ Result:

Example 4: the universal semiring $\mathbb{N}[X]$

- Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Annotate input facts with atomic annotations $X = f_1, \ldots, f_n$
- Most general semiring: $\mathbb{N}[X]$ of polynomials on X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

→ Result: $(f_1 \otimes f_2)$

Example 4: the universal semiring $\mathbb{N}[X]$

- Consider again: $\exists xyz\ R(x, y) \land R(y, z)$.
- Annotate input facts with atomic annotations $X = f_1, \ldots, f_n$
- Most general semiring: $\mathbb{N}[X]$ of polynomials on X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Result: $(f_1 \otimes f_2)$

Example 4: the universal semiring $\mathbb{N}[X]$

- Consider again: $\exists xyz\ R(x, y) \land R(y, z)$.
- Annotate input facts with atomic annotations $X = f_1, \ldots, f_n$
- Most general semiring: $\mathbb{N}[X]$ of polynomials on X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Result: $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4)$

Example 4: the universal semiring $\mathbb{N}[X]$

- Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Annotate input facts with atomic annotations $X = f_1, \ldots, f_n$
- Most general semiring: $\mathbb{N}[X]$ of polynomials on X

	R	
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Result: $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4)$

Example 4: the universal semiring $\mathbb{N}[X]$

- Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Annotate input facts with atomic annotations $X = f_1, \ldots, f_n$
- Most general semiring: $\mathbb{N}[X]$ of polynomials on X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Result: $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4) \oplus (f_4 \otimes f_3)$

Example 4: the universal semiring $\mathbb{N}[X]$

- Consider again: $\exists xyz\ R(x, y) \land R(y, z)$.
- Annotate input facts with atomic annotations $X = f_1, \ldots, f_n$
- Most general semiring: $\mathbb{N}[X]$ of polynomials on X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

→ Result: $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4) \oplus (f_4 \otimes f_3)$

Example 4: the universal semiring $\mathbb{N}[X]$

- Consider again: $\exists xyz\ R(x, y) \land R(y, z)$.
- Annotate input facts with atomic annotations $X = f_1, \ldots, f_n$
- Most general semiring: $\mathbb{N}[X]$ of polynomials on X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

→ Result: $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4) \oplus (f_4 \otimes f_3) \oplus (f_5 \otimes f_5)$

Example 4: the universal semiring $\mathbb{N}[X]$

- Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.
- Annotate input facts with atomic annotations $X = f_1, \ldots, f_n$
- Most general semiring: $\mathbb{N}[X]$ of polynomials on X

R		
a	b	f_1
b	c	f_2
d	e	f_3
e	d	f_4
f	f	f_5

$\rightarrow$ Result: $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4) \oplus (f_4 \otimes f_3) \oplus (f_5 \otimes f_5)$

Specialization and homomorphisms

- These examples are captured by commutative semirings:
  - security semiring $(K, \text{min}, \text{max}, \text{Public}, \text{Never available})$
  - bag semiring $(\mathbb{N}, +, \times, 0, 1)$
  - Boolean semiring $(\text{PosBool}[X], \vee, \wedge, \mathfrak{f}, \mathfrak{t})$
  - universal semiring $(\mathbb{N}[X], +, \times, 0, 1)$

Specialization and homomorphisms

- These examples are captured by commutative semirings:
  - security semiring $(K, \min, \max, \text{Public}, \text{Never available})$
  - bag semiring $(\mathbb{N}, +, \times, 0, 1)$
  - Boolean semiring $(\text{PosBool}[X], \vee, \wedge, \mathfrak{f}, \mathfrak{t})$
  - universal semiring $(\mathbb{N}[X], +, \times, 0, 1)$
- $\mathbb{N}[X]$ is the universal semiring:
  - The provenance for $\mathbb{N}[X]$ can be specialized to any $K[X]$
  - By commutation with homomorphisms, atomic annotations in X can be replaced by their value in K

Specialization and homomorphisms

- These examples are captured by commutative semirings:
  - security semiring $(K, \min, \max, \text{Public}, \text{Never available})$
  - bag semiring $(\mathbb{N}, +, \times, 0, 1)$
  - Boolean semiring $(\text{PosBool}[X], \vee, \wedge, \mathfrak{f}, \mathfrak{t})$
  - universal semiring $(\mathbb{N}[X], +, \times, 0, 1)$
- $\mathbb{N}[X]$ is the universal semiring:
  - The provenance for $\mathbb{N}[X]$ can be specialized to any $K[X]$
  - By commutation with homomorphisms, atomic annotations in X can be replaced by their value in K

→ Computing $\mathbb{N}[X]$ provenance subsumes all tasks

→ It can be done in PTIME data complexity for CQs

Provenance and probability

- Probabilistic query evaluation:
  - Fixed CQ q, and input TID instance:

R		
a	b	0.6
b	c	0.9

Provenance and probability

- Probabilistic query evaluation:
  - Fixed CQ q, and input TID instance:

R		
a	b	0.6
b	c	0.9

$\rightarrow$ Computing the probability of the $\mathrm{PosBool}[X]$-provenance

Provenance and probability

- Probabilistic query evaluation:
  - Fixed CQ q, and input TID instance:

R		
a	b	0.6
b	c	0.9

$\rightarrow$ Computing the probability of the $\mathrm{PosBool}[X]$-provenance

$\rightarrow$ #P-hard in data complexity

Trees and treelike instances

- Idea: restrict the instances to trees and treelike instances
  - Tree decomposition of an instance: cover all facts

Trees and treelike instances

- Idea: restrict the instances to trees and treelike instances
  - Tree decomposition of an instance: cover all facts
  - Treewidth: minimal width (bag size) of a decomposition

Trees and treelike instances

- Idea: restrict the instances to trees and treelike instances
  - Tree decomposition of an instance: cover all facts
  - Treewidth: minimal width (bag size) of a decomposition
    - Trees have treewidth 1
    - Cycles have treewidth 2
    - k-cliques and k-grids have treewidth $k-1$

Trees and treelike instances

- Idea: restrict the instances to trees and treelike instances
  - Tree decomposition of an instance: cover all facts
  - Treewidth: minimal width (bag size) of a decomposition
    - Trees have treewidth 1
    - Cycles have treewidth 2
    - k-cliques and k-grids have treewidth $k - 1$
  - Treelike: the treewidth is bounded by a constant

Problem statement

- Many tasks have tractable data complexity on treelike instances:
  - MSO query evaluation is linear [Courcelle, 1990]
  - MSO result counting is linear [Arnborg et al., 1991]
  - Probability evaluation is linear for trees [Cohen et al., 2009]
  - (MSO covers relational algebra, UCQs, monadic Datalog...)

Problem statement

- Many tasks have tractable data complexity on treelike instances:
  - MSO query evaluation is linear [Courcelle, 1990]
  - MSO result counting is linear [Arnborg et al., 1991]
  - Probability evaluation is linear for trees [Cohen et al., 2009]
  - (MSO covers relational algebra, UCQs, monadic Datalog...)

→ Can we define provenance in this setting?

Problem statement

- Many tasks have tractable data complexity on treelike instances:
  - MSO query evaluation is linear [Courcelle, 1990]
  - MSO result counting is linear [Arnborg et al., 1991]
  - Probability evaluation is linear for trees [Cohen et al., 2009]
  - (MSO covers relational algebra, UCQs, monadic Datalog...)

→ Can we define provenance in this setting?

→ Can we compute it efficiently?

Problem statement

- Many tasks have tractable data complexity on treelike instances:
  - MSO query evaluation is linear [Courcelle, 1990]
  - MSO result counting is linear [Arnborg et al., 1991]
  - Probability evaluation is linear for trees [Cohen et al., 2009]
  - (MSO covers relational algebra, UCQs, monadic Datalog...)

→ Can we define provenance in this setting?

→ Can we compute it efficiently?

→ Can we generalize the above results?

Table of contents

General idea

- Bool[X]-provenance on trees and treelike instances
- The world of trees:
  - Query: MSO on trees
- The world of treelike instances:
  - Query: MSO on the instance

  → Reduces to trees [Courcelle, 1990]

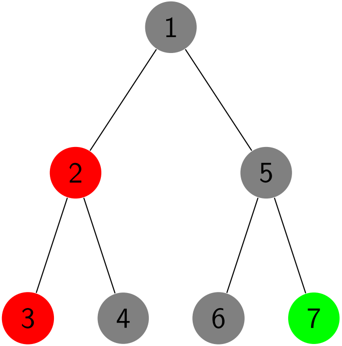
General idea

- $\mathrm{Bool}[X]$-provenance on trees and treelike instances
- The world of trees:
  - Query: MSO on trees
- The world of treelike instances:
  - Query: MSO on the instance

  → Reduces to trees [Courcelle, 1990]

→ Start with $\mathrm{Bool}[X]$-provenance for queries on trees

Uncertain trees



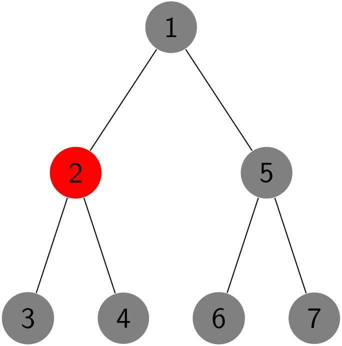
A valuation of a tree decides whether to keep or discard node labels.

Example query:
"Is there both a red and a green node?"

Valuation: $\{1, 2, 3, 4, 5, 6, 7\}$

The query is true

Uncertain trees



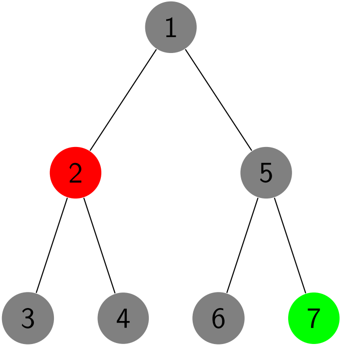
A valuation of a tree decides whether to keep or discard node labels.

Example query:
"Is there both a red and a green node?"

Valuation: $\{1, 2, 5, 6\}$

The query is false

Uncertain trees



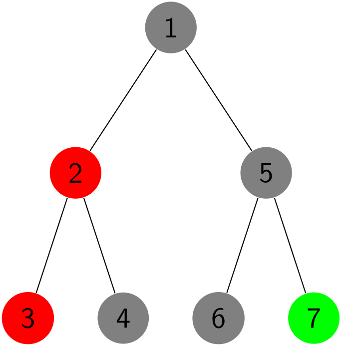
A valuation of a tree decides whether to keep or discard node labels.

Example query:
"Is there both a red and a green node?"

Valuation: $\{2, 7\}$

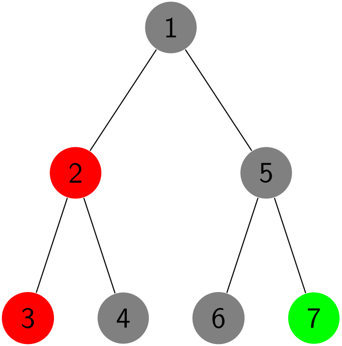
The query is true

Provenance formulae and circuits



- Which valuations satisfy the query?

Provenance formulae and circuits



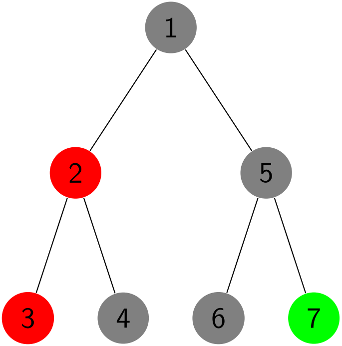
- Which valuations satisfy the query?

→ Provenance formula of a query q on an uncertain tree T:

- Boolean formula ϕ
- on variables $x_1 \dots x_7$

→ $\nu(T)$ satisfies q iff $\nu(\phi)$ is true

Provenance formulae and circuits



- Which valuations satisfy the query?

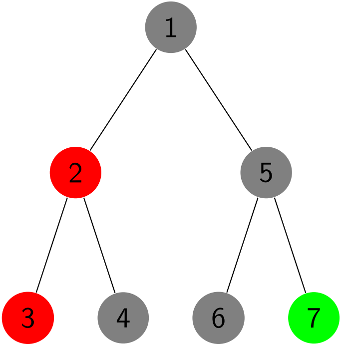
→ Provenance formula of a query q on an uncertain tree T:
  - Boolean formula ϕ
  - on variables $x_1 \dots x_7$

  → $\nu(T)$ satisfies q iff $\nu(\phi)$ is true

- Provenance circuit of q on T [Deutch et al., 2014]
  - Boolean circuit C
  - with input gates $g_1 \dots g_7$

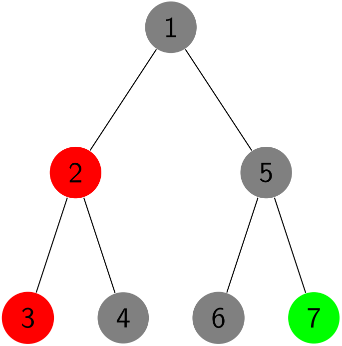
  → $\nu(T)$ satisfies q iff $\nu(C)$ is true

Example



Is there both a red and a green node?

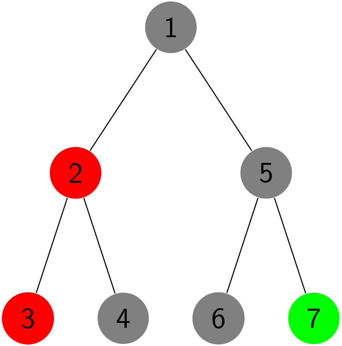
Example



Is there both a red and a green node?

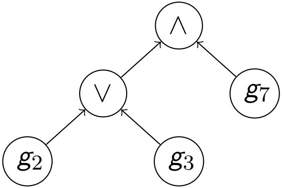
- Provenance formula: $(x_2 \vee x_3) \wedge x_7$

Example



Is there both a red and a green node?

- Provenance formula: $(x_2 \vee x_3) \wedge x_7$
- Provenance circuit:



Our main result on trees

Theorem

For any fixed MSO query q (first order + quantify on sets),
for any input tree T,
we can build a $\mathrm{Bool}[X]$ provenance circuit of q on T
in linear time in T.

Our main result on trees

Theorem

For any fixed MSO query q (first order + quantify on sets),
for any input tree T,
we can build a Bool[X] *provenance circuit of q on T*
in linear time in T.

→ Key ideas:

- Compile q to a tree automaton [Thatcher and Wright, 1968]
- Write the possible transitions of the automaton on T

Our main result on trees

Theorem

For any fixed MSO query q (first order + quantify on sets), for any input tree T, we can build a $\mathrm{Bool}[X]$ provenance circuit of q on T in linear time in T.

→ Key ideas:

- Compile q to a tree automaton [Thatcher and Wright, 1968]
- Write the possible transitions of the automaton on T

Corollary

If tree nodes have a probability of being independently kept, we can compute the query probability in linear time.

Treelike instances

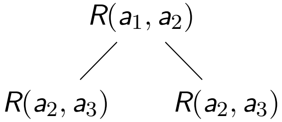
- Tree encodings: represent treelike instances as trees
- MSO query on an instance → MSO query on the tree encoding

Treelike instances

- Tree encodings: represent treelike instances as trees
- MSO query on an instance $\rightarrow$ MSO query on the tree encoding
- Uncertain instance: each fact can be present or absent

$\rightarrow$ Possible subinstances are possible valuations of the encoding

R	
a	b
b	c
b	d

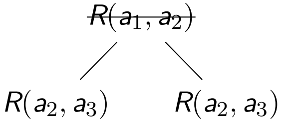


Treelike instances

- Tree encodings: represent treelike instances as trees
- MSO query on an instance → MSO query on the tree encoding
- Uncertain instance: each fact can be present or absent

→ Possible subinstances are possible valuations of the encoding

R	
~~a~~	~~b~~
b	c
b	d

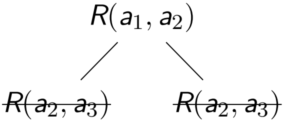


Treelike instances

- Tree encodings: represent treelike instances as trees
- MSO query on an instance $\rightarrow$ MSO query on the tree encoding
- Uncertain instance: each fact can be present or absent

$\rightarrow$ Possible subinstances are possible valuations of the encoding

R	
a	b
~~b~~	~~c~~
~~b~~	~~d~~

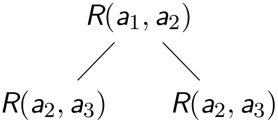


Treelike instances

- Tree encodings: represent treelike instances as trees
- MSO query on an instance → MSO query on the tree encoding
- Uncertain instance: each fact can be present or absent

→ Possible subinstances are possible valuations of the encoding

R	
a	b
b	c
b	d



Our result and consequences

Theorem

*For any fixed MSO query q and $k \in \mathbb{N}$,
for any input instance I of treewidth $\leq k$,
we can build in linear time a $\mathrm{Bool}[X]$ provenance circuit of q on I.*

Our result and consequences

Theorem

For any fixed MSO query q and $k \in \mathbb{N}$,
for any input instance I of treewidth $\leq k$,
we can build in linear time a $\mathrm{Bool}[X]$ provenance circuit of q on I.

Corollary

MSO queries have linear data complexity on treelike TID instances.

Our result and consequences

Theorem

For any fixed MSO query q and $k \in \mathbb{N}$,
for any input instance I of treewidth $\leq k$,
we can build in linear time a $\mathrm{Bool}[X]$ provenance circuit of q on I.

Corollary

MSO queries have linear data complexity on treelike TID instances.

Corollary

MSO counting has linear time complexity (already known).

Table of contents

First problem: non-monotone queries

- We want to move from $\mathrm{Bool}[X]$ to $\mathbb{N}[X]$
- Semirings and negation don't mix [Amsterdamer et al., 2011]
- Our previous construction builds circuits with NOT-gates

First problem: non-monotone queries

- We want to move from $\mathrm{Bool}[X]$ to $\mathbb{N}[X]$
- Semirings and negation don't mix [Amsterdamer et al., 2011]
- Our previous construction builds circuits with NOT-gates

$\rightarrow$ q monotone if $I \models q$ implies $I' \models q$ for all $I' \supseteq I$

First problem: non-monotone queries

- We want to move from $\mathrm{Bool}[X]$ to $\mathbb{N}[X]$
- Semirings and negation don't mix [Amsterdamer et al., 2011]
- Our previous construction builds circuits with NOT-gates

→ q monotone if $I \models q$ implies $I' \models q$ for all $I' \supseteq I$

→ Provenance circuits for monotone queries can be monotone

Second problem: intrinsic definition

- Boolean provenance has an intrinsic definition:
  "Characterize which subinstances satisfy the query"
  - $\rightarrow$ Independent from how the query is written
  - $\rightarrow$ Independent from its encoding on trees
- $\mathbb{N}[X]$-provenance was defined operationally
  - $\rightarrow$ Depends on how the query is written

Second problem: intrinsic definition

- Boolean provenance has an intrinsic definition:
  "Characterize which subinstances satisfy the query"
  - $\rightarrow$ Independent from how the query is written
  - $\rightarrow$ Independent from its encoding on trees
- $\mathbb{N}[X]$-provenance was defined operationally
  - $\rightarrow$ Depends on how the query is written

$\rightarrow$ We restrict to (Boolean) UCQs from now on

Provenance of a Boolean CQ

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Query: $q : \exists xy\ R(x, y) \wedge R(y, x)$

Provenance of a Boolean CQ

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Query: $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance:

Provenance of a Boolean CQ

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Query: $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance:
  $(x_1 \otimes x_1)$

Provenance of a Boolean CQ

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Query: $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance:
  $(x_1 \otimes x_1)$

Provenance of a Boolean CQ

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Query: $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance:
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3)$

Provenance of a Boolean CQ

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Query: $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance:
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3)$

Provenance of a Boolean CQ

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Query: $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance:
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3) \oplus (x_3 \otimes x_2)$

Provenance of a Boolean CQ

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Query: $q : \exists xy \; R(x, y) \land R(y, x)$
- Provenance:
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3) \oplus (x_3 \otimes x_2)$
  aka $x_1^2 + 2x_2x_3$

Provenance of a Boolean CQ

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Query: $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance:
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3) \oplus (x_3 \otimes x_2)$
  aka $x_1^2 + 2x_2x_3$
- Definition:
  - Sum over query matches
  - Multiply over matched facts

Provenance of a Boolean CQ

R		
a	a	x_1
b	c	x_2
c	b	x_3

- Query: $q : \exists xy\ R(x, y) \wedge R(y, x)$
- Provenance:
  $(x_1 \otimes x_1) \oplus (x_2 \otimes x_3) \oplus (x_3 \otimes x_2)$
  aka $x_1^2 + 2x_2x_3$
- Definition:
  - Sum over query matches
  - Multiply over matched facts

How is $\mathbb{N}[X]$ more expressive than $\mathrm{PosBool}[X]$?

→ Coefficients: counting multiple derivations

→ Exponents: using facts multiple times

Our result for $\mathbb{N}[X]$-provenance circuits

Theorem

For any fixed UCQ q and $k \in \mathbb{N}$,
for any input instance I of treewidth $\leq k$,
we can build in linear time a $\mathbb{N}[X]$ provenance circuit of q on I.

Our result for $\mathbb{N}[X]$-provenance circuits

Theorem

*For any fixed UCQ q and $k \in \mathbb{N}$,
for any input instance I of treewidth $\leq k$,
we can build in linear time a $\mathbb{N}[X]$ provenance circuit of q on I.*

$\rightarrow$ What fails for MSO/Datalog?

- Unbounded maximal multiplicity
- Logical definition of fact multiplicity?

Table of contents

Summary

- Result:
  - → Linear time provenance circuit computation on trees and treelike instances:
    - for MSO, $\mathrm{Bool}[X]$
    - for monotone MSO, $\mathrm{PosBool}[X]$
    - for UCQ, $\mathbb{N}[X]$
  - → cheaper than on arbitrary instances (linear vs PTIME)
  - → not more expensive than query evaluation

Summary

- Result:
  - → Linear time provenance circuit computation on trees and treelike instances:
    - for MSO, $\mathrm{Bool}[X]$
    - for monotone MSO, $\mathrm{PosBool}[X]$
    - for UCQ, $\mathbb{N}[X]$
  - → cheaper than on arbitrary instances (linear vs PTIME)
  - → not more expensive than query evaluation
- Techniques:
  - Creative provenance representations (arithmetic circuits)
  - Intrinsic definitions of provenance (rather than operational)
  - Extending provenance to MSO ($\mathrm{PosBool}[X]$ only for now)

Summary

- Result:
  - → Linear time provenance circuit computation on trees and treelike instances:
    - for MSO, $\mathrm{Bool}[X]$
    - for monotone MSO, $\mathrm{PosBool}[X]$
    - for UCQ, $\mathbb{N}[X]$
  - → cheaper than on arbitrary instances (linear vs PTIME)
  - → not more expensive than query evaluation
- Techniques:
  - Creative provenance representations (arithmetic circuits)
  - Intrinsic definitions of provenance (rather than operational)
  - Extending provenance to MSO ($\mathrm{PosBool}[X]$ only for now)
- Applications:
  - → Capture existing results (decouple symbolic and numerical computation)
  - → Extend to new applications (probabilities)

Future work

- Extend $\mathbb{N}[X]$ beyond UCQs (e.g., formal series, multiplicities)
- Monadic Datalog? [Gottlob et al., 2010]
- Other applications? aggregation, enumeration?
- Experiments for efficient probabilistic query evaluation
- Query-specific tree decompositions?

Future work

- Extend $\mathbb{N}[X]$ beyond UCQs (e.g., formal series, multiplicities)
- Monadic Datalog? [Gottlob et al., 2010]
- Other applications? aggregation, enumeration?
- Experiments for efficient probabilistic query evaluation
- Query-specific tree decompositions?

Thanks for your attention!

References I

Amsterdamer, Y., Deutch, D., and Tannen, V. (2011).
On the limitations of provenance for queries with difference.
In *TaPP*.

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

Chaudhuri, S. and Vardi, M. Y. (1992).
On the equivalence of recursive and nonrecursive Datalog programs.
In *PODS*.

Cohen, S., Kimelfeld, B., and Sagiv, Y. (2009).
Running tree automata on probabilistic XML.
In *PODS*.

References II

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

- Deutch, D., Milo, T., Roy, S., and Tannen, V. (2014).
  Circuits for datalog provenance.
  In *ICDT*.

- Gottlob, G., Pichler, R., and Wei, F. (2010).
  Monadic datalog over finite structures of bounded treewidth.
  TOCL, 12(1):3.

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

References III

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):57–81.

Semiring provenance [Green et al., 2007]

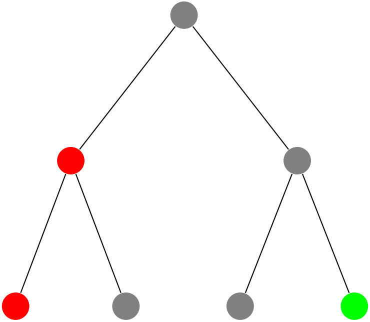
- Semiring $(K, \oplus, \otimes, 0, 1)$
  - $(K, \oplus)$ commutative monoid with identity 0
  - $(K, \otimes)$ commutative monoid with identity 1
  - $\otimes$ distributes over $\oplus$
  - 0 absorptive for $\otimes$

Semiring provenance [Green et al., 2007]

- Semiring $(K, \oplus, \otimes, 0, 1)$
  - $(K, \oplus)$ commutative monoid with identity 0
  - $(K, \otimes)$ commutative monoid with identity 1
  - $\otimes$ distributes over $\oplus$
  - 0 absorptive for $\otimes$
- Idea: Maintain annotations on tuples while evaluating:
  - Union: annotation is the sum of union tuples
  - Select: select as usual
  - Project: annotation is the sum of projected tuples
  - Product: annotation is the product

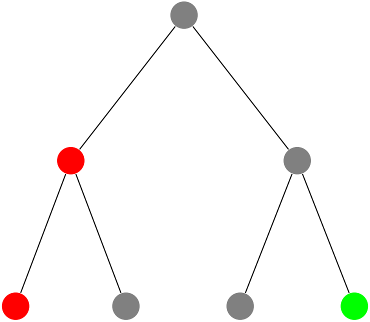
Tree automata

Tree alphabet:



Tree automata

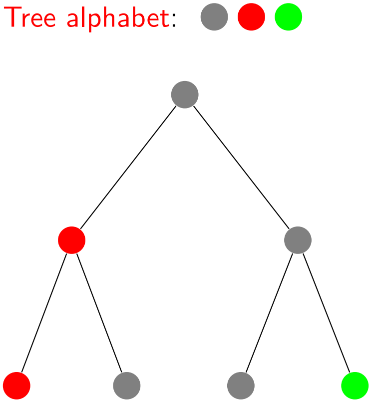
Tree alphabet: 



- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"

Tree automata

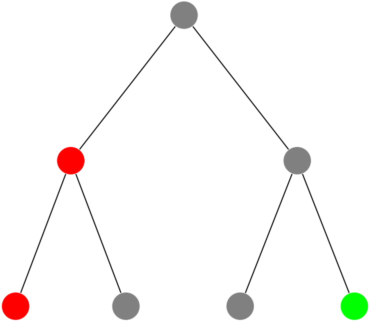
Tree alphabet:



- bNTA: bottom-up nondeterministic tree automaton
- “Is there both a red and green node?”
- States: $\{\bot, G, R, \top\}$

Tree automata

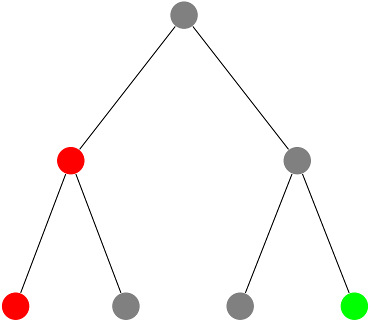
Tree alphabet:



- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$

Tree automata

Tree alphabet: 

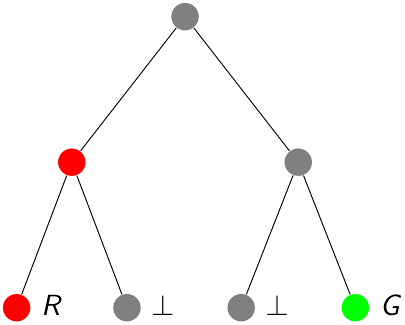


- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

 $\bot$  R  G

Tree automata

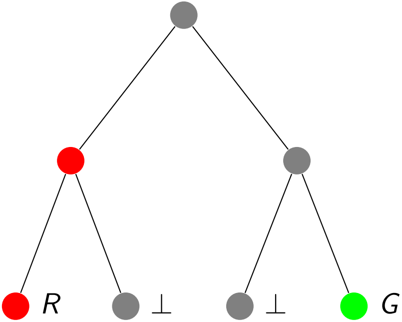
Tree alphabet: 



- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

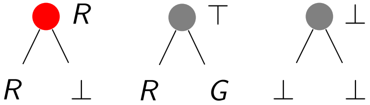
Tree automata

Tree alphabet: 



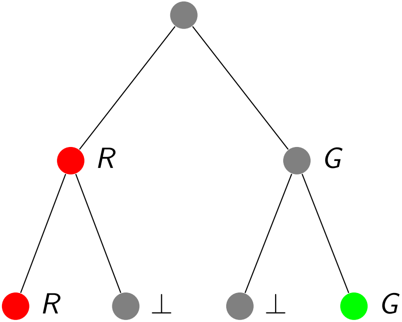
- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

- Transitions (examples):



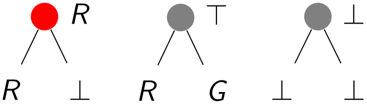
Tree automata

Tree alphabet: 



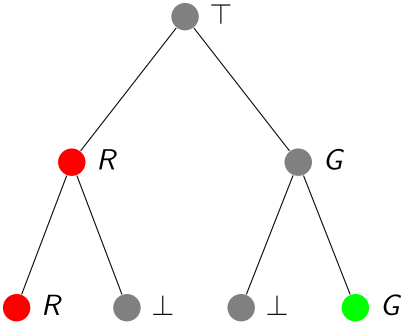
- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

- Transitions (examples):



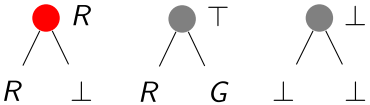
Tree automata

Tree alphabet: 



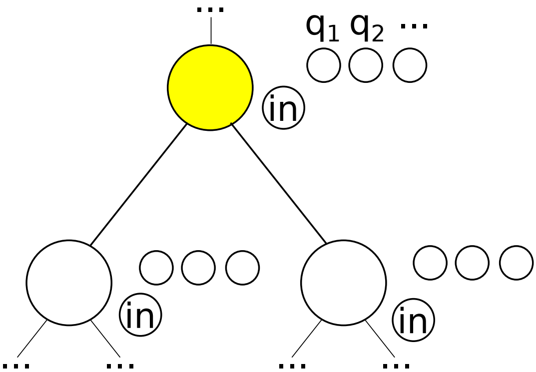
- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

- Transitions (examples):



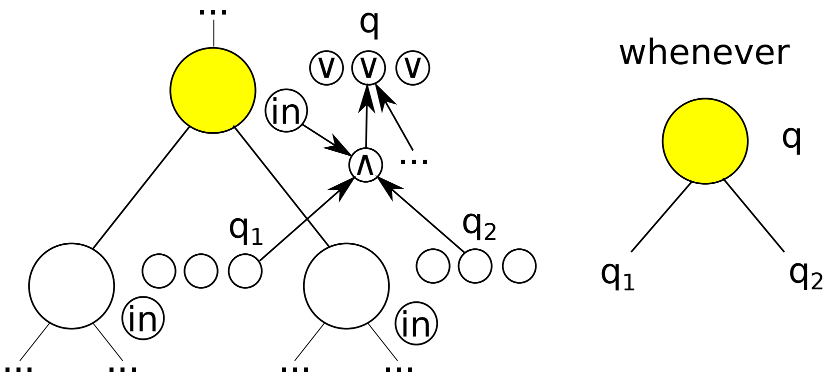
Constructing the provenance circuit

→ Construct a Boolean provenance circuit bottom-up



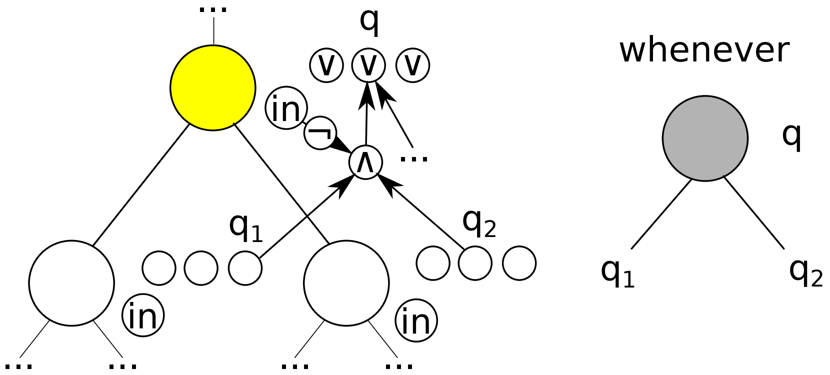
Constructing the provenance circuit

→ Construct a Boolean provenance circuit bottom-up



Constructing the provenance circuit

→ Construct a Boolean provenance circuit bottom-up



Encoding treelike instances [Chaudhuri and Vardi, 1992]

Instance:

N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e

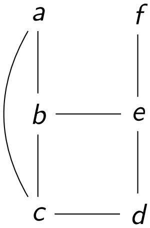
Encoding treelike instances [Chaudhuri and Vardi, 1992]

Instance:

N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e

Gaifman graph:



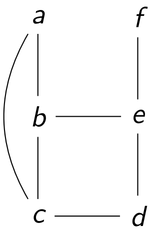
Encoding treelike instances [Chaudhuri and Vardi, 1992]

Instance:

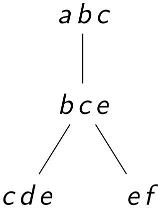
N	
a	*b*
b	*c*
c	*d*
d	*e*
e	*f*

S	
a	*c*
b	*e*

Gaifman graph:



Tree decomp.:



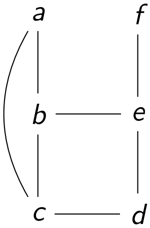
Encoding treelike instances [Chaudhuri and Vardi, 1992]

Instance:

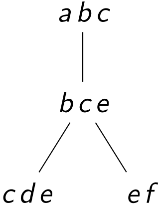
N	
a	*b*
b	*c*
c	*d*
d	*e*
e	*f*

S	
a	*c*
b	*e*

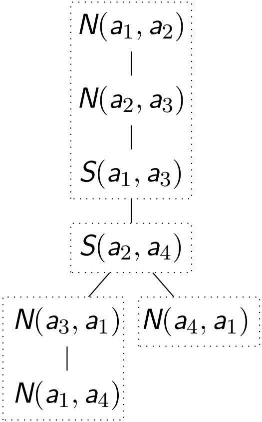
Gaifman graph:



Tree decomp.:



Tree encoding:



Example: block-independent disjoint (BID) instances

name	**city**	**iso**	*p*
pods	melbourne	au	0.8
pods	sydney	au	0.2
icalp	tokyo	jp	0.1
icalp	kyoto	jp	0.9

Example: block-independent disjoint (BID) instances

name	**city**	**iso**	*p*
pods	melbourne	au	0.8
pods	sydney	au	0.2
icalp	tokyo	jp	0.1
icalp	kyoto	jp	0.9

- Evaluating a fixed CQ is #P-hard in general

Example: block-independent disjoint (BID) instances

name	**city**	**iso**	*p*
pods	melbourne	au	0.8
pods	sydney	au	0.2
icalp	tokyo	jp	0.1
icalp	kyoto	jp	0.9

- Evaluating a fixed CQ is #P-hard in general

→ For a treelike instance, linear time!

Supporting coefficients

- In the world of trees
  - The same valuation can be accepted multiple times
  - $\rightarrow$ Number of accepting runs of the bNTA
- In the world of treelike instances
  - The same match can be the image of multiple homomorphisms

Supporting coefficients

- In the world of trees
  - The same valuation can be accepted multiple times
  - → Number of accepting runs of the bNTA
- In the world of treelike instances
  - The same match can be the image of multiple homomorphisms

→ Add assignment facts to represent possible assignments

→ Encode to a bNTA that guesses them

Supporting exponents

- In the world of trees
  - The same fact can be used multiple times
  - Annotate nodes with a multiplicity
  - The bNTA is monotone for that multiplicity
  - Use each input gate as many times as we read its fact
- In the world of treelike instances
  - The same fact can be the image of multiple atoms
  - Maximal multiplicity is query-dependent but instance-independent

Supporting exponents

- In the world of trees
  - The same fact can be used multiple times
  - Annotate nodes with a multiplicity
  - The bNTA is monotone for that multiplicity
  - Use each input gate as many times as we read its fact
- In the world of treelike instances
  - The same fact can be the image of multiple atoms
  - Maximal multiplicity is query-dependent but instance-independent

→ Encodes CQs to bNTAs that read multiplicities

  - Consider all possible CQ self-homomorphisms
  - Count the multiplicities of identical atoms
  - Rewrite relations to add multiplicities
  - Usual compilation on the modified signature