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

April 8th, 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: $\exists xyz\ R(x, y) \wedge R(y, z)$.

R	
a	b
b	c
d	e
e	d
f	f

Example 1: security for a conjunctive query

Consider the conjunctive query: $\exists xyz\ R(x,y) \wedge R(y,z)$.

R	
a	b
b	c
d	e
e	d
f	f

- Result: true

Example 1: security for a conjunctive query

Consider the conjunctive query: $\exists xyz\ R(x,y) \land R(y,z)$.

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

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

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

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

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

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?

→ 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?

→ 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?

→ 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?

$\rightarrow$ 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?

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

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: $(f_1 \wedge f_2)$

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

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

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?

$\rightarrow$ 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: $\exists xyz\ R(x, y) \land R(y, z)$.
- 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:

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:

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: $\exists xyz\ R(x, y) \wedge R(y, z)$.
- 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: $(f_1 \otimes f_2)$

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

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

→ 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: $\exists xyz\ R(x, y) \wedge R(y, z)$.
- 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: $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4) \oplus (f_4 \otimes f_3)$

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

- Consider again: $\exists xyz\ R(x, y) \wedge R(y, z)$.
- 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: $(f_1 \otimes f_2) \oplus (f_3 \otimes f_4) \oplus (f_4 \otimes f_3) \oplus (f_5 \otimes f_5)$

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

- All these examples can be captured using semirings:
  - security semiring $(K, \min, \max, \text{Public}, \text{Never available})$
  - bag semiring $(\mathbb{N}, +, \times, 0, 1)$
  - Boolean semiring $(\mathrm{PosBool}[X], \vee, \wedge, \mathfrak{f}, \mathfrak{t})$
  - universal semiring $(\mathbb{N}[X], +, \times, 0, 1)$

Specialization and homomorphisms

- All these examples can be captured using 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

- All these examples can be captured using semirings:
  - security semiring $(K, \min, \max, \text{Public}, \text{Never available})$
  - bag semiring $(\mathbb{N}, +, \times, 0, 1)$
  - Boolean semiring $(\mathrm{PosBool}[X], \vee, \wedge, \mathfrak{f}, \mathfrak{t})$
  - 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:

R		
a	b	0.6
b	c	0.9

Provenance and probability

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

R		
a	b	0.6
b	c	0.9

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

$\rightarrow$ #P-hard

Provenance and probability

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

R		
a	b	0.6
b	c	0.9

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

→ #P-hard

→ Use the provenance (here, $\mathrm{PosBool}[X]$)

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 et al., 2001]
  - 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 et al., 2001]
  - 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 explain this tractability with provenance?

- Idea: queries on treelike instances have treelike provenance?

→ Can we extend tractability to more quantitative tasks?

→ Can we define and compute provenance for MSO?

Table of contents

General idea

- $\mathrm{PosBool}[X]$-provenance on trees and treelike instances
- The world of trees:
  - Query: MSO on trees
- The world of treelike instances:
  - Query: MSO on the instance
  - $\rightarrow$ Reduces to trees [Courcelle et al., 2001]

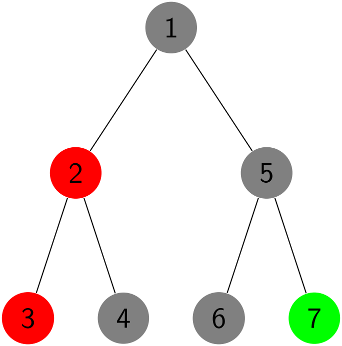
General idea

- $\mathrm{PosBool}[X]$-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 et al., 2001]

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

Uncertain trees



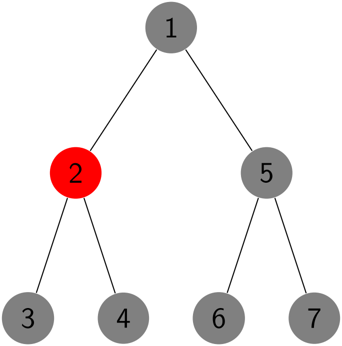
Query: "Is there both a red and green node?"

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

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

The query is true

Uncertain trees



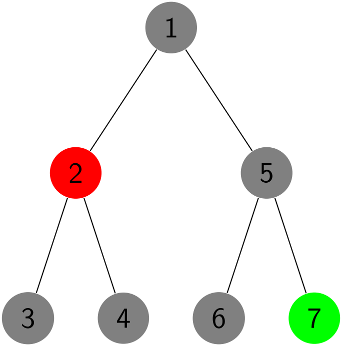
Query: "Is there both a red and green node?"

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

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

The query is false

Uncertain trees



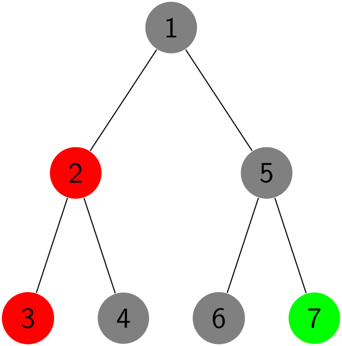
Query: "Is there both a red and green node?"

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

Keep: $\{2, 7\}$

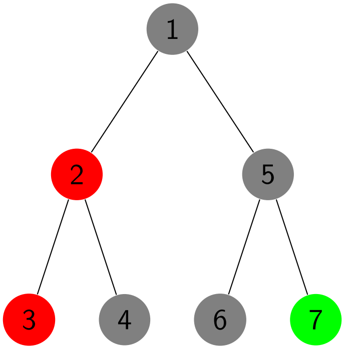
The query is true

Provenance circuits



- $X = \{g_1, g_2, g_3, g_4, g_5, g_6, g_7\}$
- $\mathrm{PosBool}[X]$-provenance of a query q on tree T:
  - monotone Boolean formula ϕ
  - on variables X
  - → $\nu(T)$ satisfies q iff $\nu(\phi)$ is true

Provenance circuits

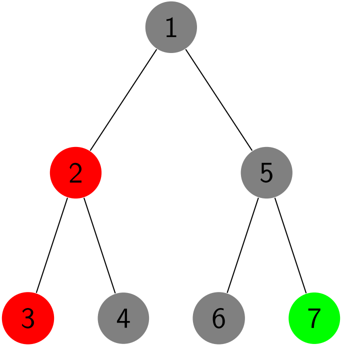


- $X = \{g_1, g_2, g_3, g_4, g_5, g_6, g_7\}$
- PosBool[X]-provenance of a query q on tree T:
  - monotone Boolean formula ϕ
  - on variables X

  → $\nu(T)$ satisfies q iff $\nu(\phi)$ is true
- Represent as a circuit [Deutch et al., 2014]
  - monotone Boolean circuit C
  - with input gates X

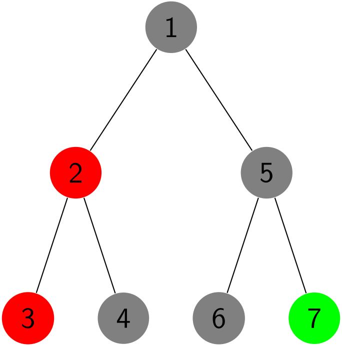
  → $\nu(T)$ satisfies q iff $\nu(C)$ is true (output gate)

Example



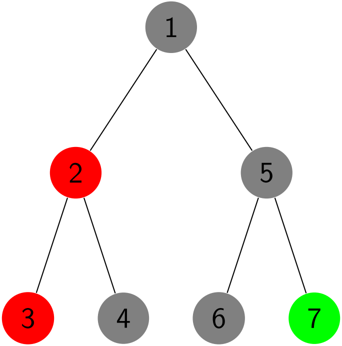
- Query: is there both a red and a green node?

Example

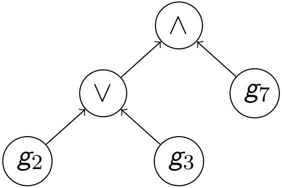


- Query: is there both a red and a green node?
- PosBool[X]-provenance: $(g_2 \vee g_3) \wedge g_7$

Example



- Query: is there both a red and a green node?
- PosBool[X]-provenance: $(g_2 \vee g_3) \wedge g_7$
- PosBool[X] provenance circuit:



Our results on trees

- A $\mathrm{PosBool}[X]$ provenance circuit of a MSO query q on a tree:
  - $\rightarrow$ can be computed in linear time in the tree for a fixed query
  - $\rightarrow$ has treewidth only dependent on the query
  - $\rightarrow$ is actually a $\mathrm{Bool}[X]$-circuit (more soon)

Our results on trees

- A $\mathrm{PosBool}[X]$ provenance circuit of a MSO query q on a tree:
  - → can be computed in linear time in the tree for a fixed query
  - → has treewidth only dependent on the query
  - → is actually a $\mathrm{Bool}[X]$-circuit (more soon)

→ Let's extend this to treelike instances!

Treelike instances

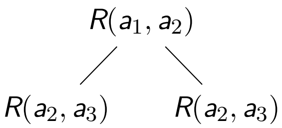
- Tree encodings: represent treelike instances as trees
- MSO queries on the instance $\rightarrow$ MSO queries on the tree encoding

Treelike instances

- Tree encodings: represent treelike instances as trees
- MSO queries on the instance → MSO queries 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

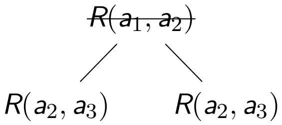


Treelike instances

- Tree encodings: represent treelike instances as trees
- MSO queries on the instance → MSO queries 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*

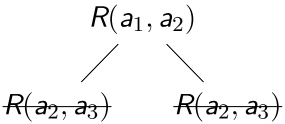


Treelike instances

- Tree encodings: represent treelike instances as trees
- MSO queries on the instance → MSO queries 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~~

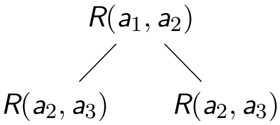


Treelike instances

- Tree encodings: represent treelike instances as trees
- MSO queries on the instance → MSO queries 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

- Compute a $\mathrm{Bool}[X]$-provenance circuit for a fixed MSO query on a treelike instance in linear time in the instance

Our result and consequences

- Compute a $\mathrm{Bool}[X]$-provenance circuit for a fixed MSO query on a treelike instance in linear time in the instance

→ Linear time data complexity for MSO probabilistic query evaluation on treelike instances
(assuming unit-cost arithmetics)

→ Covers many known probabilistic data models

Our result and consequences

- Compute a Bool[X]-provenance circuit for a fixed MSO query on a treelike instance in linear time in the instance

→ Linear time data complexity for MSO probabilistic query evaluation on treelike instances (assuming unit-cost arithmetics)

→ Covers many known probabilistic data models

- We can reduce counting to probabilistic evaluation

→ Re-proves that MSO counting has linear-time data complexity

Table of contents

First problem: non-monotone queries

- We want to generalize from $\mathrm{PosBool}[X]$ to $\mathbb{N}[X]$
- Semirings have bad support for negation [Amsterdamer et al., 2011]
- Our previous construction uses negation

First problem: non-monotone queries

- We want to generalize from $\mathrm{PosBool}[X]$ to $\mathbb{N}[X]$
- Semirings have bad support for negation [Amsterdamer et al., 2011]
- Our previous construction uses negation

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

First problem: non-monotone queries

- We want to generalize from $\mathrm{PosBool}[X]$ to $\mathbb{N}[X]$
- Semirings have bad support for negation [Amsterdamer et al., 2011]
- Our previous construction uses negation

→ 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"
  - → Independent from how the query is written
  - → Independent from its encoding on trees
- $\mathbb{N}[X]$-provenance was defined operationally
  - → Depends on how the query is written

Second problem: intrinsic definition

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

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

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

We can compute in linear time data complexity a $\mathbb{N}[X]$ provenance circuit (arithmetic circuit) for UCQs.

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

We can compute in linear time data complexity a $\mathbb{N}[X]$ provenance circuit (arithmetic circuit) for UCQs.

→ 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/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 counting or query evaluation

Summary

- Result:
  - → Linear time provenance circuit computation on trees/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 counting or 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)
  - Provenance-preserving encoding of queries

Summary

- Result:
  - → Linear time provenance circuit computation on trees/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 counting or 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)
  - Provenance-preserving encoding of queries
- Applications:
  - → Capture counting results (decouple symbolic and numerical computation)
  - → Extend to new applications (probabilities)

Future work

- Monadic Datalog [Gottlob et al., 2010] to avoid high combined complexity
- A neater approach for counting and probabilities
- Extend $\mathbb{N}[X]$ beyond CQs (e.g., formal series, multiplicities)
- Other applications? aggregation, enumeration?

Future work

- Monadic Datalog [Gottlob et al., 2010] to avoid high combined complexity
- A neater approach for counting and probabilities
- Extend $\mathbb{N}[X]$ beyond CQs (e.g., formal series, multiplicities)
- Other applications? aggregation, enumeration?

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., Makowsky, J. A., and Rotics, U. (2001).
On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic.
Discrete Applied Mathematics, 108(1-2):23–52.

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*.

Semiring provenance [Green et al., 2007]

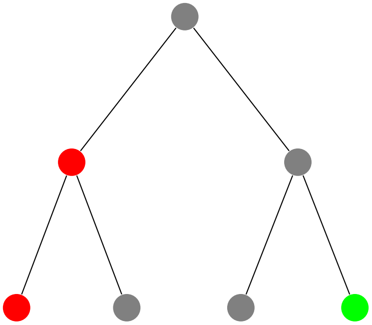
- Semiring $(K, \oplus, \otimes, 0, 1)$
  - $(K, \oplus)$ commutative monoid with identity 0
  - $(K, \otimes)$ commutative monoid with identity 1
  - $\otimes$ distributes over $\oplus$
  - 0 absorptive for $\otimes$

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

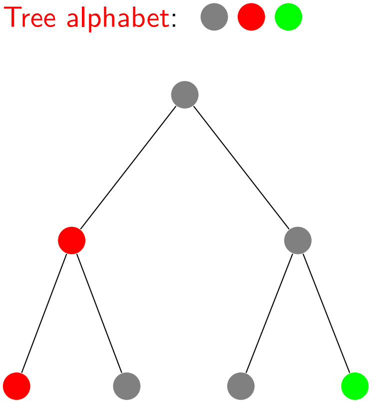
Tree automata

Tree alphabet: 



Tree automata

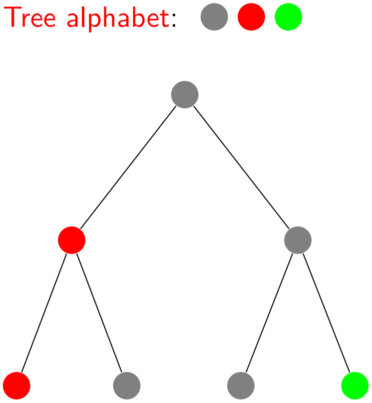
Tree alphabet:



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

Tree automata

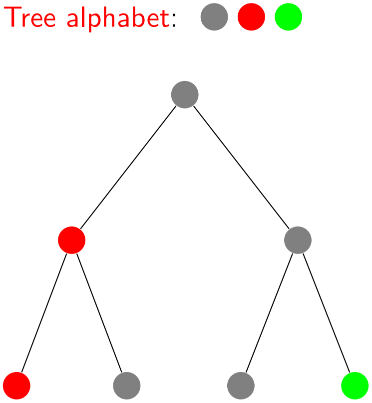
Tree alphabet:



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

Tree automata

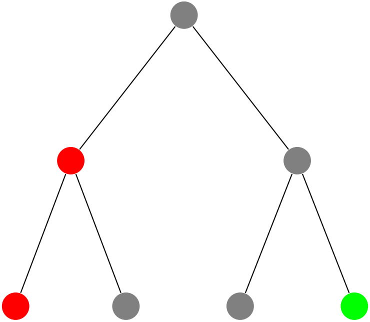
Tree alphabet:



- 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:

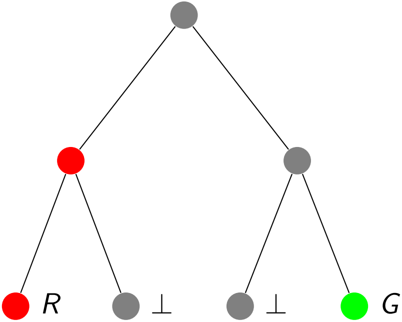


- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

  $\bot$ R G

Tree automata

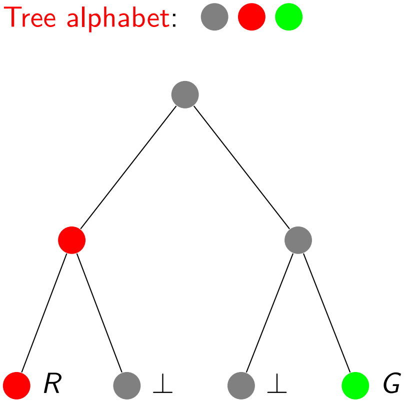
Tree alphabet: 



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

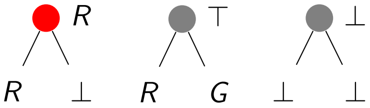
Tree automata

Tree alphabet:



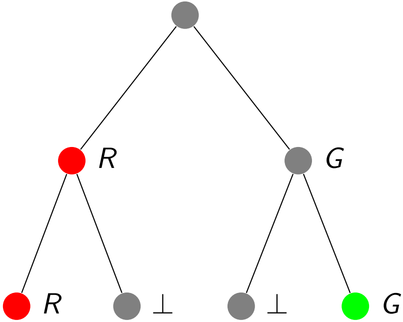
- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

- Transitions (examples):



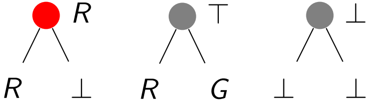
Tree automata

Tree alphabet: 



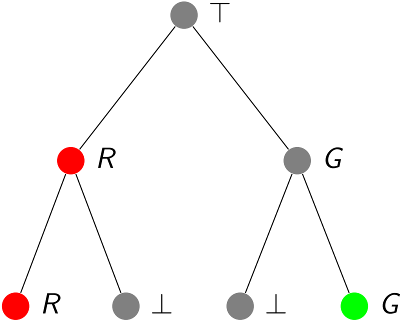
- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

- Transitions (examples):



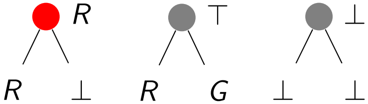
Tree automata

Tree alphabet: 



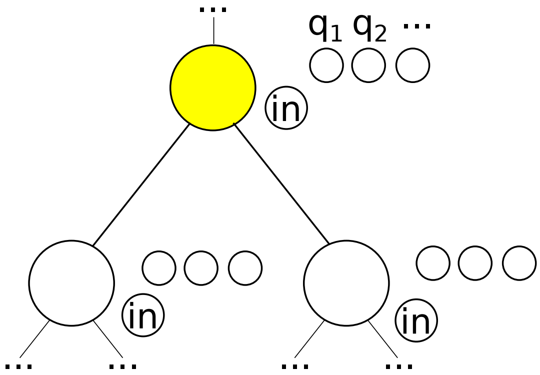
- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

- Transitions (examples):



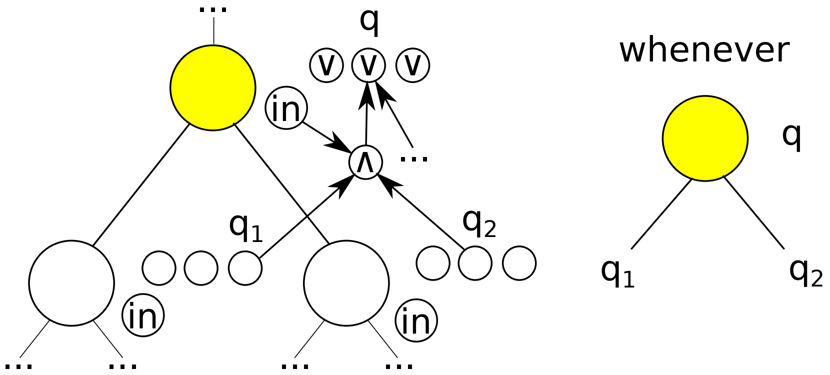
Constructing the provenance circuit

→ Construct a Boolean provenance circuit bottom-up



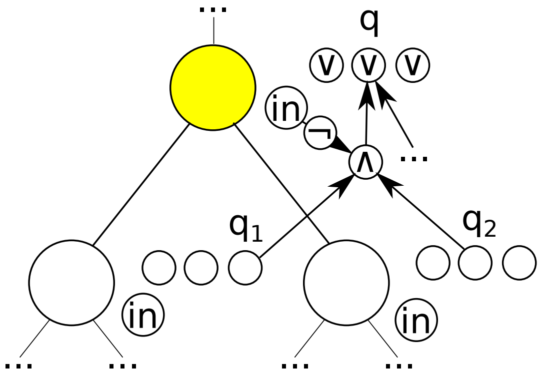
Constructing the provenance circuit

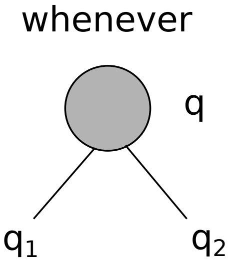
→ Construct a Boolean provenance circuit bottom-up



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

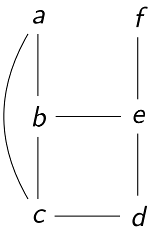
Encoding treelike instances [Chaudhuri and Vardi, 1992]

Instance:

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

S	
a	*c*
b	*e*

Gaifman graph:



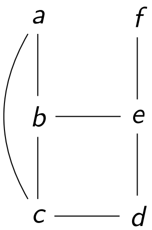
Encoding treelike instances [Chaudhuri and Vardi, 1992]

Instance:

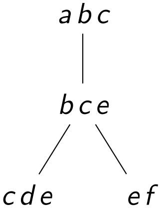
N	
a	*b*
b	*c*
c	*d*
d	*e*
e	*f*

S	
a	*c*
b	*e*

Gaifman graph:



Tree decomp.:



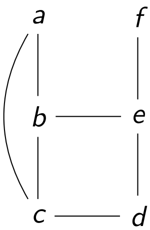
Encoding treelike instances [Chaudhuri and Vardi, 1992]

Instance:

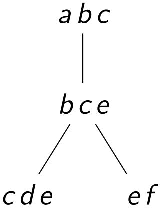
N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e

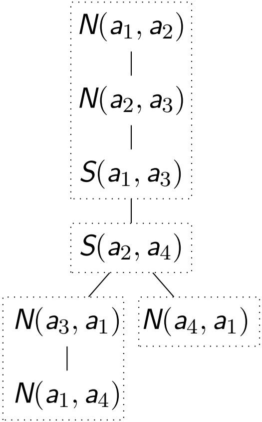
Gaifman graph:



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

<u>**name**</u>	**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

<u>**name**</u>	**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