Structurally Tractable Uncertain Data

Antoine Amarilli

Supervisor: Pierre Senellart

Expected graduation: August 2016

Télécom ParisTech, France

May 31st, 2015

Uncertain data management

Is data always complete and certain?

Uncertain data management

Is data always complete and certain?

- Unreliable sources
  - → Crowdsourcing
  - → Massive collaborations: Wikidata, etc.

Uncertain data management

Is data always complete and certain?

- Unreliable sources
  - → Crowdsourcing
  - → Massive collaborations: Wikidata, etc.
- Error-prone processing
  - → Unsupervised information extraction
  - → OCR, speech recognition, etc.

Uncertain data management

Is data always complete and certain?

- Unreliable sources
  - → Crowdsourcing
  - → Massive collaborations: Wikidata, etc.
- Error-prone processing
  - → Unsupervised information extraction
  - → OCR, speech recognition, etc.
- Outdated or stale data

Uncertain data management

Is data always complete and certain?

- Unreliable sources
  - → Crowdsourcing
  - → Massive collaborations: Wikidata, etc.
- Error-prone processing
  - → Unsupervised information extraction
  - → OCR, speech recognition, etc.
- Outdated or stale data

→ We need uncertain data management

Example model: TID

- Consider a relational instance

Date	Animal
Wed 3rd	Kangaroo
Wed 3rd	Tasmanian devil
Thu 4th	Kangaroo
Thu 4th	Tasmanian devil
Fri 5th	Kangaroo
Fri 5th	Tasmanian devil

Example model: TID

- Consider a relational instance

Date	Animal
Wed 3rd	Kangaroo
Wed 3rd	Tasmanian devil
Thu 4th	Kangaroo
Thu 4th	Tasmanian devil
Fri 5th	Kangaroo
Fri 5th	Tasmanian devil

- Add probabilities to facts

Example model: TID

- Consider a relational instance

Date	Animal	Probability
Wed 3rd	Kangaroo	5%
Wed 3rd	Tasmanian devil	0%
Thu 4th	Kangaroo	6%
Thu 4th	Tasmanian devil	2%
Fri 5th	Kangaroo	20%
Fri 5th	Tasmanian devil	15%

- Add probabilities to facts

Example model: TID

- Consider a relational instance

Date	Animal	Probability
Wed 3rd	Kangaroo	5%
Wed 3rd	Tasmanian devil	0%
Thu 4th	Kangaroo	6%
Thu 4th	Tasmanian devil	2%
Fri 5th	Kangaroo	20%
Fri 5th	Tasmanian devil	15%

- Add probabilities to facts
- Assume independence between facts

Example model: TID

- Consider a relational instance

Date	Animal	Probability
Wed 3rd	Kangaroo	5%
Wed 3rd	Tasmanian devil	0%
Thu 4th	Kangaroo	6%
Thu 4th	Tasmanian devil	2%
Fri 5th	Kangaroo	20%
Fri 5th	Tasmanian devil	15%

- Add probabilities to facts
- Assume independence between facts
  - $\rightarrow$ Semantics: a probability distribution on regular instances

Example model: TID

- Consider a relational instance

Date	Animal	Probability
Wed 3rd	Kangaroo	5%
Wed 3rd	Tasmanian devil	0%
Thu 4th	Kangaroo	6%
Thu 4th	Tasmanian devil	2%
Fri 5th	Kangaroo	20%
Fri 5th	Tasmanian devil	15%

- Add probabilities to facts
- Assume independence between facts
  - $\rightarrow$ Semantics: a probability distribution on regular instances
- What about queries? (Boolean CQs)

Example model: TID

- Consider a relational instance

Date	Animal	Probability
Wed 3rd	Kangaroo	5%
Wed 3rd	Tasmanian devil	0%
Thu 4th	Kangaroo	6%
Thu 4th	Tasmanian devil	2%
Fri 5th	Kangaroo	20%
Fri 5th	Tasmanian devil	15%

- Add probabilities to facts
- Assume independence between facts
  - → Semantics: a probability distribution on regular instances
- What about queries? (Boolean CQs)
  - → Semantics: compute the probability that the query holds

Big problem: Tractability

- Evaluate the fixed Boolean CQ: $\exists xy\ R(x)\ S(x,y)\ T(y)$
- Measure data complexity, i.e., as a function of the instance

Big problem: Tractability

- Evaluate the fixed Boolean CQ: $\exists xy\; R(x)\; S(x, y)\; T(y)$
- Measure data complexity, i.e., as a function of the instance

$\rightarrow$ #P-hard [Dalvi and Suciu, 2007] (instead of AC^0)

Big problem: Tractability

- Evaluate the fixed Boolean CQ: $\exists xy\ R(x)\ S(x,y)\ T(y)$
- Measure data complexity, i.e., as a function of the instance

$\rightarrow$ #P-hard [Dalvi and Suciu, 2007] (instead of AC^0)

Existing approaches:

- Avoid hard queries [Dalvi and Suciu, 2012]
- Use sampling to get approximate answers

The general idea

Input instances are not arbitrary!

$\rightarrow$ Impose structural restrictions on instances

$\rightarrow$ Prove fixed-parameter tractability results

This talk

- Parameter: instance treewidth
- Bound it by a constant

$\rightarrow$ MSO queries have linear data complexity [Courcelle, 1990]

This talk

- Parameter: instance treewidth
- Bound it by a constant

→ MSO queries have linear data complexity [Courcelle, 1990]

→ Also holds on TID instances (with unit cost arithmetics)
(joint work with Pierre Bourhis and Pierre Senellart)

Table of contents

1 Introduction

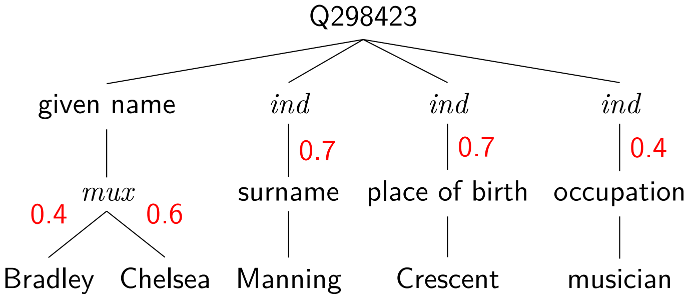
2 Trees

3 Treelike instances

4 Future work

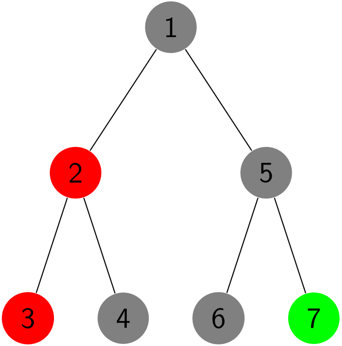
Uncertain tree example

- A possible PrXML tree, from Wikidata facts:



→ Probabilities reflect contributor trustworthiness

Formalizing uncertain trees



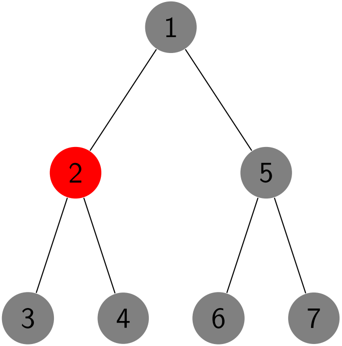
A valuation of a tree decides whether to keep or discard node labels.

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

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

The query is true

Formalizing uncertain trees



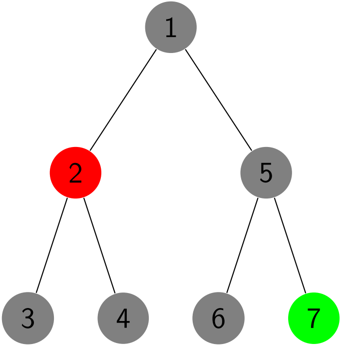
A valuation of a tree decides whether to keep or discard node labels.

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

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

The query is false

Formalizing uncertain trees



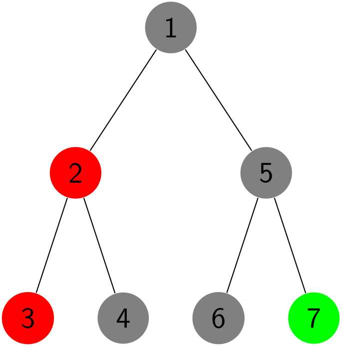
A valuation of a tree decides whether to keep or discard node labels.

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

Valuation: $\{2, 7\}$

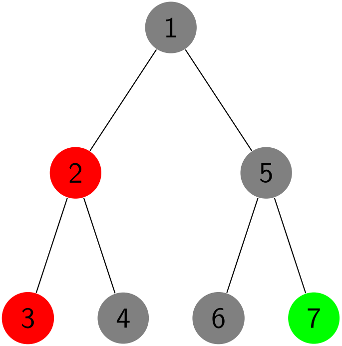
The query is true

Provenance formulae and circuits



- Which valuations satisfy the query?

Provenance formulae and circuits



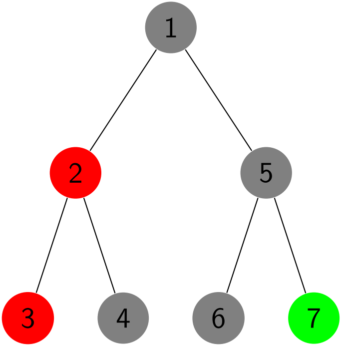
- Which valuations satisfy the query?

→ 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?

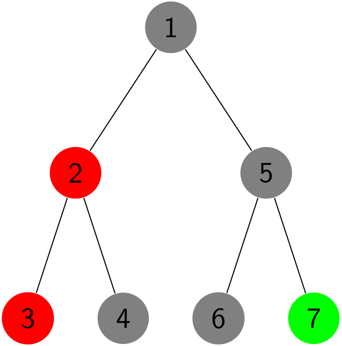
→ Provenance formula of a query q on an uncertain tree T:
  - Boolean formula ϕ
  - on variables $x_1 \dots x_7$

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

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

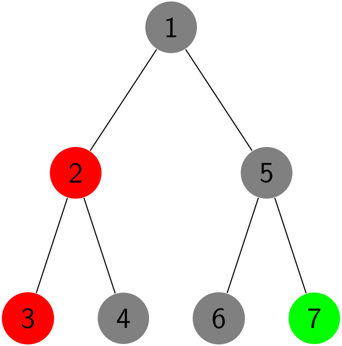
  → $\nu(T)$ satisfies q iff $\nu(C)$ is true

Example



Is there both a red and a green node?

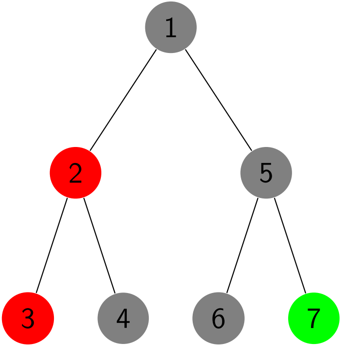
Example



Is there both a red and a green node?

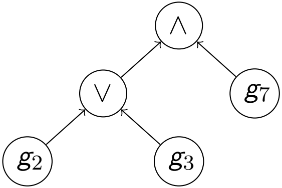
- Provenance formula: $(x_2 \lor x_3) \land x_7$

Example



Is there both a red and a green node?

- Provenance formula: $(x_2 \lor x_3) \land x_7$
- Provenance circuit:



Our main result on trees

Theorem

For any fixed MSO query q (first order + quantify on sets) we can compute a provenance circuit C for any input tree T in linear time in the input T.

Our main result on trees

Theorem

For any fixed MSO query q (first order + quantify on sets) we can compute a provenance circuit C for any input tree T in linear time in the input T.

$\rightarrow$ Key ideas:

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

Our main result on trees

Theorem

For any fixed MSO query q (first order + quantify on sets) we can compute a provenance circuit C for any input tree T in linear time in the input T.

→ Key ideas:

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

Corollary

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

Our main result on trees

Theorem

For any fixed MSO query q (first order + quantify on sets) we can compute a provenance circuit C for any input tree T in linear time in the input T.

→ Key ideas:

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

Corollary

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

→ Relates to message passing [Lauritzen and Spiegelhalter, 1988]

→ Already known [Cohen et al., 2009]

Table of contents

1 Introduction

2 Trees

3 Treelike instances

4 Future work

Treewidth intuition

Generalize from trees to treelike instances:

Treewidth intuition

Generalize from trees to treelike instances:

- Treewidth: measure on instances
  - 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

Treewidth intuition

Generalize from trees to treelike instances:

- Treewidth: measure on instances
  - 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

$\rightarrow$ Treelike instances can be encoded to trees

Treewidth formal definition

Instance:

N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e

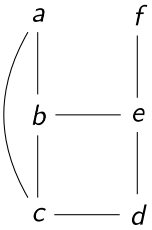
Treewidth formal definition

Instance:

N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e

Gaifman graph:



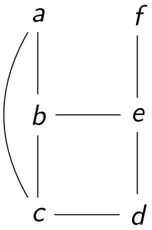
Treewidth formal definition

Instance:

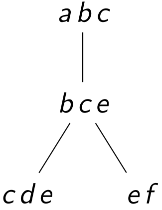
N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e

Gaifman graph:



Tree decomp.:



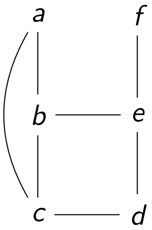
Treewidth formal definition

Instance:

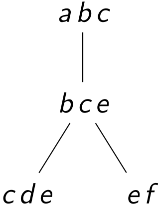
N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e

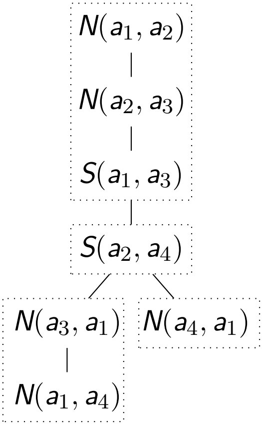
Gaifman graph:



Tree decomp.:



Tree encoding:



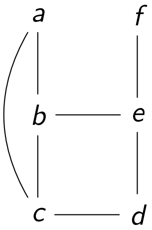
Treewidth formal definition

Instance:

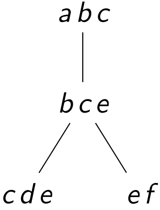
N	
a	b
b	c
c	d
d	e
e	f

S	
a	c
b	e

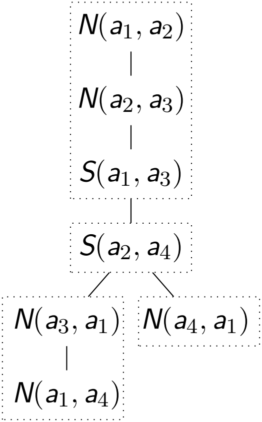
Gaifman graph:



Tree decomp.:



Tree encoding:



→ Treelike: constant bound on the maximal bag size

Our main result on treelike instances

Theorem

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

Our main result on treelike instances

Theorem

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

$\rightarrow$ Key ideas:

- Compute tree decomposition and tree encoding in linear time
- Compile q to an automaton on encodings [Flum et al., 2002]
- Use the previous construction

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

Our main result on treelike instances

Theorem

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

$\rightarrow$ Key ideas:

- Compute tree decomposition and tree encoding in linear time
- Compile q to an automaton on encodings [Flum et al., 2002]
- Use the previous construction

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

Corollary

MSO queries have linear data complexity on treelike TID instances.

Further results

- Support other models with dependencies between facts:
  - Block-independent disjoint (BID): mutually exclusive facts
  - pc-tables: events and Boolean annotations

Further results

- Support other models with dependencies between facts:
  - Block-independent disjoint (BID): mutually exclusive facts
  - pc-tables: events and Boolean annotations
- Support other tasks:
  - Counting query results encodes to probabilistic evaluation
  - General connection to semiring provenance [Green et al., 2007]

Table of contents

1 Introduction

2 Trees

3 Treelike instances

4 Future work

Extending the provenance connection

- Negation:
  - Semiring provenance usually defined for positive queries
  - Yet our provenance circuits work fine with negation

  $\rightarrow$ Relate this to provenance for queries with negation?

Extending the provenance connection

- Negation:
  - Semiring provenance usually defined for positive queries
  - Yet our provenance circuits work fine with negation
  - $\rightarrow$ Relate this to provenance for queries with negation?
- Multiplicities:
  - Our works connects to the universal semiring $\mathbb{N}[X]$...
  - ... but only for UCQs, not arbitrary MSO
  - Missing: notion of multiplicity for MSO (multisets?)

Extending the provenance connection

- Negation:
  - Semiring provenance usually defined for positive queries
  - Yet our provenance circuits work fine with negation
  - $\rightarrow$ Relate this to provenance for queries with negation?
- Multiplicities:
  - Our works connects to the universal semiring $\mathbb{N}[X]$...
  - ... but only for UCQs, not arbitrary MSO
  - Missing: notion of multiplicity for MSO (multisets?)
- Structural restrictions:
  - Are real-world instances tree-like?
  - Are there other possible restrictions?
  - Experiments?

Connect to other frameworks

- Compiling to automata has high combined complexity

→ Investigate Monadic Datalog approaches [Gottlob et al., 2010]

Connect to other frameworks

- Compiling to automata has high combined complexity

→ Investigate Monadic Datalog approaches [Gottlob et al., 2010]

- Uncertainty on facts not values

→ Connect to work on nulls [Libkin, 2014]

Connect to other frameworks

- Compiling to automata has high combined complexity

→ Investigate Monadic Datalog approaches [Gottlob et al., 2010]

- Uncertainty on facts not values

→ Connect to work on nulls [Libkin, 2014]

- What about reasoning on uncertain data and its implications?

→ Connect to tractable languages (e.g., guarded Datalog)

Connect to other frameworks

- Compiling to automata has high combined complexity

$\rightarrow$ Investigate Monadic Datalog approaches [Gottlob et al., 2010]

- Uncertainty on facts not values

$\rightarrow$ Connect to work on nulls [Libkin, 2014]

- What about reasoning on uncertain data and its implications?

$\rightarrow$ Connect to tractable languages (e.g., guarded Datalog)

- What about incorporating new evidence?

$\rightarrow$ Connect to work on conditioning [Tang et al., 2012]

Other projects and directions

- Open-world query answering (with Michael Benedikt)
  - Certainty of a Boolean CQ when completing under constraints
  - Which constraint languages are decidable?
  - What is the impact of assuming finiteness?

Other projects and directions

- Open-world query answering (with Michael Benedikt)
  - Certainty of a Boolean CQ when completing under constraints
  - Which constraint languages are decidable?
  - What is the impact of assuming finiteness?

- Uncertain ordered data
  - Bag semantics for the relational algebra
    (with M. Lamine Ba, Daniel Deutch, Pierre Senellart)
  - Interpolation schemes for partially ordered numerical values
    (with Yael Amsterdamer, Tova Milo, Pierre Senellart)

Other projects and directions

- Open-world query answering (with Michael Benedikt)
  - Certainty of a Boolean CQ when completing under constraints
  - Which constraint languages are decidable?
  - What is the impact of assuming finiteness?
- Uncertain ordered data
  - Bag semantics for the relational algebra
    (with M. Lamine Ba, Daniel Deutch, Pierre Senellart)
  - Interpolation schemes for partially ordered numerical values
    (with Yael Amsterdamer, Tova Milo, Pierre Senellart)
- Problem of instance possibility
  - On uncertain orders (labeled posets)
  - On probabilistic XML

Other projects and directions

- Open-world query answering (with Michael Benedikt)
  - Certainty of a Boolean CQ when completing under constraints
  - Which constraint languages are decidable?
  - What is the impact of assuming finiteness?
- Uncertain ordered data
  - Bag semantics for the relational algebra (with M. Lamine Ba, Daniel Deutch, Pierre Senellart)
  - Interpolation schemes for partially ordered numerical values (with Yael Amsterdamer, Tova Milo, Pierre Senellart)
- Problem of instance possibility
  - On uncertain orders (labeled posets)
  - On probabilistic XML

Thanks for your attention!

References I

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

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

Dalvi, N. and Suciu, D. (2007).
Efficient query evaluation on probabilistic databases.
VLDBJ, 16(4):523–544.

References II

Dalvi, N. and Suciu, D. (2012).
The dichotomy of probabilistic inference for unions of conjunctive queries.
JACM, 59(6):30.

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

Flum, J., Frick, M., and Grohe, M. (2002).
Query evaluation via tree-decompositions.
JACM, 49(6):716–752.

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

References III

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

Lauritzen, S. L. and Spiegelhalter, D. J. (1988).
Local computations with probabilities on graphical structures and their application to expert systems.
J. Royal Statistical Society. Series B.

Libkin, L. (2014).
Incomplete data: what went wrong, and how to fix it.
In *Proc. SIGMOD*.

Tang, R., Cheng, R., Wu, H., and Bressan, S. (2012).
A framework for conditioning uncertain relational data.
In *Database and Expert Systems Applications*. Springer.

References IV

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]

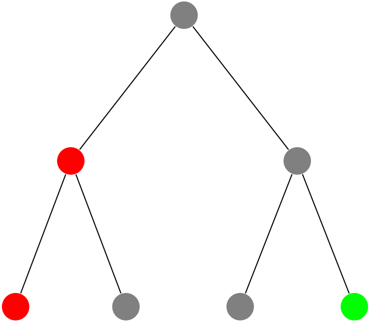
- Semiring $(K, \oplus, \otimes, 0, 1)$
  - $(K, \oplus)$ commutative monoid with identity 0
  - $(K, \otimes)$ commutative monoid with identity 1
  - $\otimes$ distributes over $\oplus$
  - 0 absorptive for $\otimes$

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

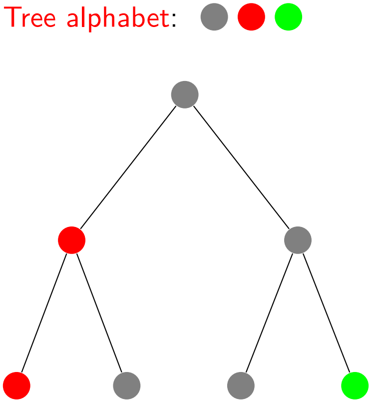
Tree automata

Tree alphabet:



Tree automata

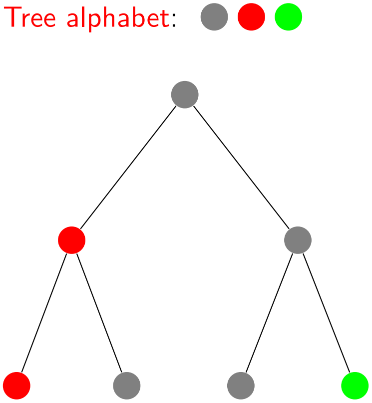
Tree alphabet:



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

Tree automata

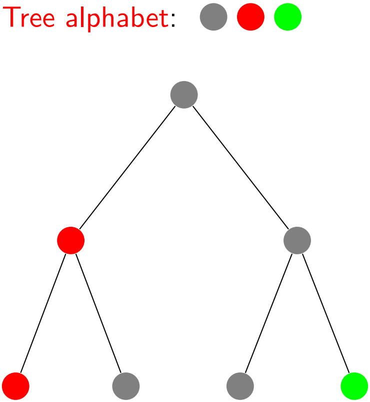
Tree alphabet:



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

Tree automata

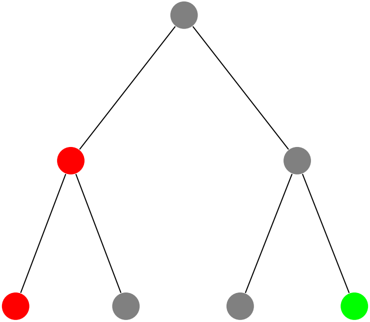
Tree alphabet:



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

Tree automata

Tree alphabet: 

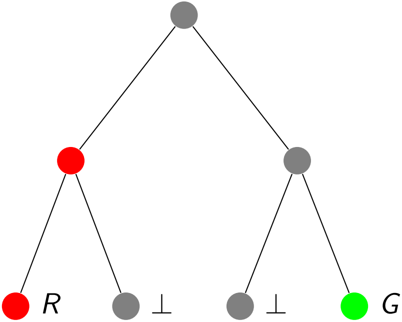


- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

   $\bot$  R  G

Tree automata

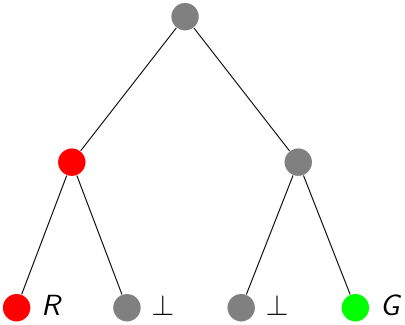
Tree alphabet: 



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

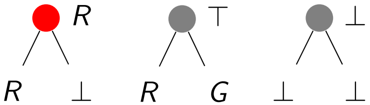
Tree automata

Tree alphabet: 



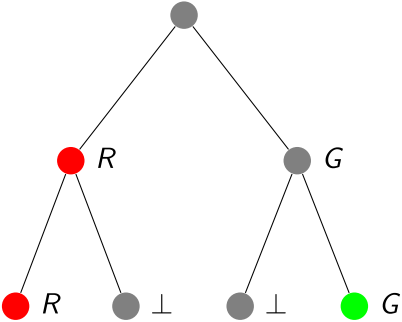
- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

- Transitions (examples):



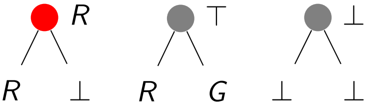
Tree automata

Tree alphabet: 



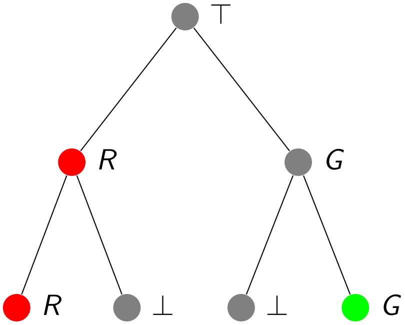
- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\bot, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

- Transitions (examples):



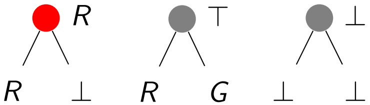
Tree automata

Tree alphabet: 



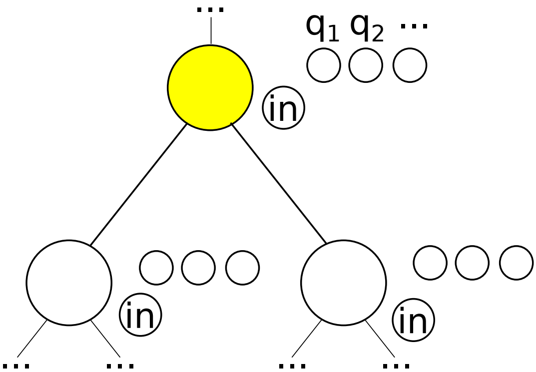
- bNTA: bottom-up nondeterministic tree automaton
- "Is there both a red and green node?"
- States: $\{\perp, G, R, \top\}$
- Final states: $\{\top\}$
- Initial function:

- Transitions (examples):



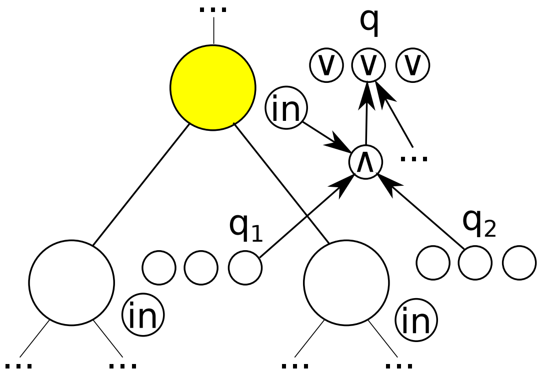
Constructing the provenance circuit

→ Construct a Boolean provenance circuit bottom-up

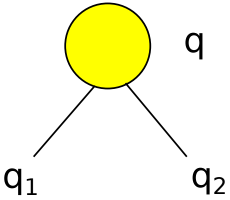


Constructing the provenance circuit

→ Construct a Boolean provenance circuit bottom-up

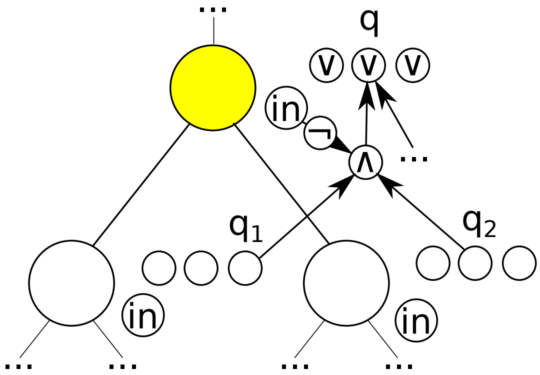


whenever

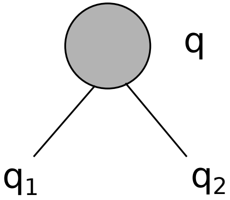


Constructing the provenance circuit

→ Construct a Boolean provenance circuit bottom-up



whenever



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

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
  - → 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

$\rightarrow$ 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