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.
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- **Outdated or stale data**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- Outdated or stale data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**

<table>
<thead>
<tr>
<th>Date</th>
<th>Animal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed 3rd</td>
<td>Kangaroo</td>
</tr>
<tr>
<td>Wed 3rd</td>
<td>Tasmanian devil</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Kangaroo</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Tasmanian devil</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Kangaroo</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Tasmanian devil</td>
</tr>
</tbody>
</table>
Example model: TID

- Consider a relational instance

<table>
<thead>
<tr>
<th>Date</th>
<th>Animal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed 3rd</td>
<td>Kangaroo</td>
</tr>
<tr>
<td>Wed 3rd</td>
<td>Tasmanian devil</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Kangaroo</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Tasmanian devil</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Kangaroo</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Tasmanian devil</td>
</tr>
</tbody>
</table>

- Add probabilities to facts
Example model: **TID**

- Consider a *relational instance*

<table>
<thead>
<tr>
<th>Date</th>
<th>Animal</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed 3rd</td>
<td>Kangaroo</td>
<td>5%</td>
</tr>
<tr>
<td>Wed 3rd</td>
<td>Tasmanian devil</td>
<td>0%</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Kangaroo</td>
<td>6%</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Tasmanian devil</td>
<td>2%</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Kangaroo</td>
<td>20%</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Tasmanian devil</td>
<td>15%</td>
</tr>
</tbody>
</table>

- Add *probabilities* to facts
Example model: TID

- Consider a relational instance

<table>
<thead>
<tr>
<th>Date</th>
<th>Animal</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed 3rd</td>
<td>Kangaroo</td>
<td>5%</td>
</tr>
<tr>
<td>Wed 3rd</td>
<td>Tasmanian devil</td>
<td>0%</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Kangaroo</td>
<td>6%</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Tasmanian devil</td>
<td>2%</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Kangaroo</td>
<td>20%</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Tasmanian devil</td>
<td>15%</td>
</tr>
</tbody>
</table>

- Add probabilities to facts
- Assume independence between facts
Example model: TID

Consider a relational instance

<table>
<thead>
<tr>
<th>Date</th>
<th>Animal</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed 3rd</td>
<td>Kangaroo</td>
<td>5%</td>
</tr>
<tr>
<td>Wed 3rd</td>
<td>Tasmanian devil</td>
<td>0%</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Kangaroo</td>
<td>6%</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Tasmanian devil</td>
<td>2%</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Kangaroo</td>
<td>20%</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Tasmanian devil</td>
<td>15%</td>
</tr>
</tbody>
</table>

• Add probabilities to facts
• Assume independence between facts
  → Semantics: a probability distribution on regular instances
Example model: TID

- Consider a relational instance

<table>
<thead>
<tr>
<th>Date</th>
<th>Animal</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed 3rd</td>
<td>Kangaroo</td>
<td>5%</td>
</tr>
<tr>
<td>Wed 3rd</td>
<td>Tasmanian devil</td>
<td>0%</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Kangaroo</td>
<td>6%</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Tasmanian devil</td>
<td>2%</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Kangaroo</td>
<td>20%</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Tasmanian devil</td>
<td>15%</td>
</tr>
</tbody>
</table>

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

- Consider a *relational instance*

<table>
<thead>
<tr>
<th>Date</th>
<th>Animal</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wed 3rd</td>
<td>Kangaroo</td>
<td>5%</td>
</tr>
<tr>
<td>Wed 3rd</td>
<td>Tasmanian devil</td>
<td>0%</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Kangaroo</td>
<td>6%</td>
</tr>
<tr>
<td>Thu 4th</td>
<td>Tasmanian devil</td>
<td>2%</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Kangaroo</td>
<td>20%</td>
</tr>
<tr>
<td>Fri 5th</td>
<td>Tasmanian devil</td>
<td>15%</td>
</tr>
</tbody>
</table>

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

Existing approaches:

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

Input instances are not arbitrary!

→ Impose structural restrictions on instances
→ Prove fixed-parameter tractability results
This talk

- Parameter: instance treewidth
- Bound it by a constant

→ 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:

```
Q298423
  given name
  mux
    0.4
    0.6
    Bradley
    Chelsea

  ind
  surname
    Manning

  ind
  place of birth
  Crescent

  ind
  occupation
  musician
```

→ 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 \textit{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 $\phi$
  - on variables $x_1 \ldots x_7$
  → $\nu(T)$ 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 $\phi$
  - on variables $x_1 \ldots x_7$
  → $\nu(T)$ satisfies $q$ iff $\nu(\phi)$ is true

Provenance circuit of $q$ on $T$ [Deutch et al., 2014]
  - Boolean circuit $C$
  - with input gates $g_1 \ldots g_7$
  → $\nu(T)$ satisfies $q$ iff $\nu(C)$ is true
Example

Is there both a red and a green node?
Is there both a **red** and a **green** node?

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

---

**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]
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 $C$
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

→ 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:

\[
\begin{array}{cccc}
N & & & \\
\hline
a & b & & \\
b & c & & \\
c & d & & \\
d & e & & \\
e & f & & \\
\end{array}
\]

\[
\begin{array}{cccc}
S & & & \\
\hline
a & c & & \\
b & e & & \\
\end{array}
\]

Gaifman graph:

\[
\begin{array}{ccc}
\text{a} & \text{f} & \\
\hline
\text{b} & \text{e} & \\
\text{c} & \text{d} & \\
\end{array}
\]

Tree decomp.:

\[
\begin{array}{ccc}
\text{a} & \text{b} & \text{c} \\
\text{b} & \text{c} & \text{e} \\
\text{c} & \text{d} & \text{e} \\
\end{array}
\]

Tree encoding:

\[
\begin{array}{c}
N(a_1; a_2) \\
S(a_2; a_3) \\
N(a_3; a_1) \\
N(a_1; a_4) \\
N(a_4; a_1) \\
\end{array}
\]

Tree-like: constant bound on the maximal bag size
Treewidth formal definition

Instance:  
\[
\begin{array}{c|c|c}
\text{N} & \text{S} \\
\hline
a & b & a \\
b & c & c \\
c & d & d \\
d & e & e \\
e & f & f \\
\end{array}
\]

Gaifman graph:  
\[
\begin{array}{c|c|c}
\text{a} & \text{f} & \text{abc} \\
\hline
\text{b} & \text{e} & \text{bce} \\
\text{c} & \text{d} & \text{cde} \\
\text{e} & \text{f} & \text{ef} \\
\end{array}
\]

Tree decomp.:  
\[
\begin{array}{c|c|c}
\text{a} & \text{b} & \text{c} \\
\hline
\text{b} & \text{e} & \text{d} \\
\text{bce} & \text{cde} & \text{ef} \\
\end{array}
\]
Treewidth formal definition

Instance:

Gaifman graph:

Tree decomp.:

Tree encoding:

Instance:

Gaifman graph:

Tree decomp.:

Tree encoding:
Treewidth formal definition

Instance:

$\begin{array}{c}
N \\
\hline
a & b \\
\hline
b & c \\
\hline
c & d \\
\hline
d & e \\
\hline
e & f \\
\hline
S \\
\end{array}$

Gaifman graph:

$\begin{array}{c}
a & f \\
\hline
b & e \\
\hline
c & d \\
\end{array}$

Tree decomp.:

$\begin{array}{c}
abc \\
\hline
cde \\
\hline
ef \\
\end{array}$

Tree encoding:

$\begin{array}{c}
N(a_1, a_2) \\
\hline
N(a_2, a_3) \\
\hline
S(a_1, a_3) \\
\hline
S(a_2, a_4) \\
\hline
N(a_3, a_1) \\
\hline
N(a_4, a_1) \\
\hline
N(a_1, a_4) \\
\end{array}$

$\rightarrow$ 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 $I$.

→ 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
→ 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 $I$.

→ 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
  → 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
  → 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
    → 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
  - 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
  → 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)

- What about incorporating new evidence?
  → 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!


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: □ □ □

Is there both a red and green node?

States: f, G, R, ⊤

Final states: f⊤

Initial function: R G

Transitions (examples):
R ? R ⊤ G R ? ?
Tree automata

Tree alphabet:  

- bNTA: bottom-up nondeterministic tree automaton
- “Is there both a red and green node?”
Tree automata

- **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: \{⊥, G, R, ⊤\}
- Final states: \{⊤\}
Tree automata

bNTA: bottom-up nondeterministic tree automaton

“Is there both a red and green node?”

States: \{\bot, G, R, \top\}

Final states: \{\top\}

Initial function:

\[
\begin{align*}
\bullet & \quad \bot \\
\bullet & \quad R \\
\bullet & \quad G
\end{align*}
\]
Tree automata

Tree alphabet: \( \text{\textcolor{red}{R}} \) \( \text{\textcolor{green}{G}} \) \( \text{\textcolor{gray}{\bot}} \)

- **bNTA**: bottom-up nondeterministic tree automaton
- “Is there both a red and green node?”
- **States**: \( \{ \bot, G, R, \top \} \)
- **Final states**: \( \{ \top \} \)
- **Initial function**:
  \[
  \begin{align*}
  \text{\textcolor{red}{R}} & \rightarrow \text{\textcolor{red}{R}} \\
  \text{\textcolor{gray}{\bot}} & \rightarrow \text{\textcolor{gray}{\bot}} \\
  \text{\textcolor{gray}{\bot}} & \rightarrow \text{\textcolor{gray}{\bot}} \\
  \text{\textcolor{green}{G}} & \rightarrow \text{\textcolor{green}{G}}
  \end{align*}
  \]
Tree automata

Tree alphabet: \( \bullet \bullet \bullet \)

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

**Transitions (examples):**
Tree automata

- **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
  - \textcolor{red}{R}
  - \textcolor{green}{G}
- **Transitions** (examples):
  - \begin{array}{c}
    \text{\textcolor{red}{R}} \\
    \text{\bot} \\
    \text{\bot} \\
    \text{\textcolor{green}{G}}
  \end{array}
Tree automata

Tree alphabet: $\top$

**bNTA**: bottom-up nondeterministic tree automaton

“Is there both a red and green node?”

- **States**: $\{\bot, G, R, \top\}$
- **Final states**: $\{\top\}$
- **Initial function**:
  - $\bot$
  - $R$
  - $G$

**Transitions** (examples):

- $R$  $\bot$  $G$  $\bot$  $\bot$
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

Diagram showing a Boolean circuit with variables $q_1$, $q_2$, and $q$, connected by logical operators and inputs.
Example: block-independent disjoint (BID) instances

<table>
<thead>
<tr>
<th>name</th>
<th>city</th>
<th>iso</th>
<th>$p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>pods</td>
<td>melbourne</td>
<td>au</td>
<td>0.8</td>
</tr>
<tr>
<td>pods</td>
<td>sydney</td>
<td>au</td>
<td>0.2</td>
</tr>
<tr>
<td>icalp</td>
<td>tokyo</td>
<td>jp</td>
<td>0.1</td>
</tr>
<tr>
<td>icalp</td>
<td>kyoto</td>
<td>jp</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Example: block-independent disjoint (BID) instances

<table>
<thead>
<tr>
<th>name</th>
<th>city</th>
<th>iso</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>pods</td>
<td>melbourne</td>
<td>au</td>
<td>0.8</td>
</tr>
<tr>
<td>pods</td>
<td>sydney</td>
<td>au</td>
<td>0.2</td>
</tr>
<tr>
<td>icalp</td>
<td>tokyo</td>
<td>jp</td>
<td>0.1</td>
</tr>
<tr>
<td>icalp</td>
<td>kyoto</td>
<td>jp</td>
<td>0.9</td>
</tr>
</tbody>
</table>

- Evaluating a fixed CQ is \#P-hard in general
Example: block-independent disjoint (BID) instances

<table>
<thead>
<tr>
<th>name</th>
<th>city</th>
<th>iso</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>pods</td>
<td>melbourne</td>
<td>au</td>
<td>0.8</td>
</tr>
<tr>
<td>pods</td>
<td>sydney</td>
<td>au</td>
<td>0.2</td>
</tr>
<tr>
<td>icalp</td>
<td>tokyo</td>
<td>jp</td>
<td>0.1</td>
</tr>
<tr>
<td>icalp</td>
<td>kyoto</td>
<td>jp</td>
<td>0.9</td>
</tr>
</tbody>
</table>

- 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

→ 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