Static analysis over tree-structured data using graph decompositions

Filip Murlak
University of Warsaw, Poland

Contains joint work with Mikołaj Bojańczyk, Wojciech Czerwiński, Claire David, Filip Mazowiecki, Pawel Parys, and Adam Witkowski.

ALCOP 2017
Glasgow, Scotland
Problems

Old solutions

New solution

More problems with solutions

Some problems without solutions
Data

Students Table

<table>
<thead>
<tr>
<th>Student</th>
<th>ID*</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>084</td>
</tr>
<tr>
<td>Jane Bloggs</td>
<td>100</td>
</tr>
<tr>
<td>John Smith</td>
<td>182</td>
</tr>
</tbody>
</table>

Activities Table

<table>
<thead>
<tr>
<th>ID*</th>
<th>Activity*</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>084</td>
<td>Swimming</td>
<td>$17</td>
</tr>
<tr>
<td>084</td>
<td>Tennis</td>
<td>$36</td>
</tr>
<tr>
<td>100</td>
<td>Squash</td>
<td>$40</td>
</tr>
<tr>
<td>100</td>
<td>Swimming</td>
<td>$17</td>
</tr>
<tr>
<td>182</td>
<td>Tennis</td>
<td>$36</td>
</tr>
</tbody>
</table>

<xypair>
    <xaxis axistype="independent">
        <property>Wavelength</property>
        <value>254.0</value>
        <unit>nm</unit>
    </xaxis>
    <yaxis axistype="dependent">
        <property>Absorbance</property>
        <value>0.1234</value>
    </yaxis>
</xypair>
Data trees

trees finite, unranked, ordered

labels $a, b, c, \ldots$ from a finite alphabet (tags)

data values $0, 1, 2, \ldots$ from an infinite data domain (contents)
Schemas describe allowed shapes of data trees

```xml
<xs:element name="brightstar">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="magnitude" type="xs:decimal"/>
      <xs:element name="distance" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Define several types of trees, each specified (recursively) by

- the label of the root,
- possible sequences of immediate subtree types (regexp);

and choose some of the types as allowed.
Schemas describe allowed shapes of data trees

Define several types of trees, each specified (recursively) by

1. the label of the root,
2. possible sequences of immediate subtree types (regexp);

and choose some of the types as allowed.

Example: *a*-only path from root to leaf, *b*’s elsewhere

- type $\tau$: root label $a$, immediate subtree types $\sigma^* \tau \sigma^* + \epsilon$;
- type $\sigma$: root label $b$, immediate subtree types $\sigma^*$;
- choose: $\tau$. 
Conjunctive queries over data trees

\[
\exists x_1 \cdots \exists x_5 \\
\quad \text{child}(x_1, x_2) \land \text{child}(x_2, x_3) \land \text{child}(x_3, x_4) \land \\
\quad \land \text{desc}(x_1, x_5) \land \text{desc}(x_5, x_4) \land \\
\quad \land a(x_1) \land a(x_4) \land c(x_5) \land \\
\quad \land x_2 \sim x_3
\]
Datalog on data trees

extensional predicates \( \text{child}, \text{desc}, \sim, a, b, c, \ldots; \)

intensional predicates defined recursively using conjunctive queries;

monadic only unary intensional predicates;

linear at most one intensional atom per rule.

\[
p(x) \leftarrow a(x) \land \\
\text{desc}(x, y) \land c(y) \land x \sim y \land \\
\text{child}(x, z) \land p(z)
\]

\[
p(x) \leftarrow b(x)
\]
**Static analysis problems**

**Satisfiability:** Is query $P$ (CQ, UCQ, Datalog, FO, etc.) satisfied in some data tree (conforming to given schema)?

**Equivalence:** Are queries $P$, $Q$ equivalent on all data trees?

**Containment:** Does $P$ imply $Q$ on all data trees?

The staple of data management: query optimization, consistency tests, evaluation modulo constraints, constraint entailment, . . .

By Trakhtenbrot’s theorem, all **undecidable** for FO queries.
Static analysis problems

Satisfiability: Is query $P$ (CQ, UCQ, Datalog, FO, etc.) satisfied in some data tree (conforming to given schema)?

Equivalence: Are queries $P, Q$ equivalent on all data trees?

Containment: Does $P$ imply $Q$ on all data trees?

The staple of data management: query optimization, consistency tests, evaluation modulo constraints, constraint entailment, . . .

By Trakhtenbrot’s theorem, all undecidable for FO queries.

$P \ \text{sat} \iff \neg P \iff \bot \iff \neg P \Rightarrow \bot$

$P \land \neg Q, \ Q \land \neg P \ \text{unsat} \iff P \iff Q \iff P \Rightarrow Q, \ Q \Rightarrow P$

$P \land \neg Q \ \text{unsat} \iff P \iff P \Rightarrow P \land Q \iff P \Rightarrow Q$
Problems

Old solutions

New solution

More problems with solutions

Some problems without solutions
Containment of CQs over arbitrary structures

[Chandra, Merlin ’77]

Def: \( Q \in \text{CQ} \leadsto A_Q \): universe \( \text{Var}Q \),
relations given by atoms of \( Q \)

Fact: \( A \models Q \) iff exists \( h: A_Q \rightarrow A \)

Thm: \( P \Rightarrow Q \) iff exists \( g: A_Q \rightarrow A_P \)
Containment of CQs over arbitrary structures

[Chandra, Merlin ’77]

Def: \( Q \in CQ \mapsto \mathbb{A}_Q \): universe \( \text{Var}Q \), relations given by atoms of \( Q \)

Fact: \( \mathbb{A} \models Q \) iff exists \( h: \mathbb{A}_Q \to \mathbb{A} \)

Thm: \( P \Rightarrow Q \) iff exists \( g: \mathbb{A}_Q \to \mathbb{A}_P \)

\( \iff \) If \( g: \mathbb{A}_Q \to \mathbb{A}_P \) and \( h: \mathbb{A}_P \to \mathbb{A} \), then \( h \circ g: \mathbb{A}_Q \to \mathbb{A} \).

\( \Rightarrow \) \( \mathbb{A}_P \models P \) and \( P \Rightarrow Q \), so \( \mathbb{A}_P \models Q \). Exists \( h: \mathbb{A}_Q \to \mathbb{A}_P \).
Containment of CQs over arbitrary structures

[Chandra, Merlin ’77]

**Def:** \( Q \in CQ \leadsto A_Q \): universe \( \text{Var} Q \), relations given by atoms of \( Q \)

**Fact:** \( A \models Q \) iff exists \( h: A_Q \rightarrow A \)

**Thm:** \( P \Rightarrow Q \) iff exists \( g: A_Q \rightarrow A_P \)

(\( \Leftarrow \)) If \( g: A_Q \rightarrow A_P \) and \( h: A_P \rightarrow A \), then \( h \circ g: A_Q \rightarrow A \).

(\( \Rightarrow \)) \( A_P \models P \) and \( P \Rightarrow Q \), so \( A_P \models Q \). Exists \( h: A_Q \rightarrow A_P \).

To decide containment, test existence of a homomorphism.
Containment for UCQs over trees without data

[Michlau, Suciu ’04]

Each UCQ is equivalent to a union of tree-shaped CQs:

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

\equiv

\text{a} \land \neg \text{b} \land \text{c}

\lor

\text{a} \lor \text{b} \lor \text{c}

Containment for UCQs over trees without data

[Miklau, Suciu ’04]

Each UCQ is equivalent to a union of tree-shaped CQs:

\[
\begin{align*}
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{c}
\end{array} & \equiv \\
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{c}
\end{array} \lor \\
\begin{array}{c}
\text{a} \\
\text{c} \\
\text{b}
\end{array}
\end{align*}
\]

For a tree shaped CQ \( \pi \) build an equivalent tree automaton:

- it computes bottom-up the set of matched subtrees of \( \pi \);
- knowing which subtrees of \( \pi \) match at the children of node \( v \) or strictly below, one can tell which match at \( v \) or strictly below.
Containment for UCQs over trees without data

[Miklau, Suciu ’04]

Each UCQ is equivalent to a union of tree-shaped CQs:

For a tree shaped CQ $\pi$ build an equivalent tree automaton:
- it computes bottom-up the set of matched subtrees of $\pi$;
- knowing which subtrees of $\pi$ match at the children of node $v$ or strictly below, one can tell which match at $v$ or strictly below.

Tree automata are effectively closed under Boolean combinations.

Test emptiness of the automaton corresponding to $P \land \neg Q$. 
Containment for UCQs over data trees

[Björklund, Martens, Schwentick ’08]

Can restrict to trees with data values $c_1, \ldots, c_{||P||}$ and distinct nulls.

- Let $T$ be a tree satisfying $P$ and not $Q$.
- $P$ touches $\leq ||P||$ data values in $T$; replace with $c_1, \ldots, c_{||P||}$.
- In each node not touched by $P$ put a unique fresh data value.
- The resulting tree $T'$ still satisfies $P$ and not $Q$. 

In such trees, $x \sim y$ holds iff either $x = y$ or $x \sim c_i$ and $y \sim c_i$.

By considering all possibilities, replace $P$, $Q$ with $P'$, $Q'$ using only $x = y$, $x \sim c_i$, $y \sim c_i$.

Check containment over the finite alphabet $\Sigma \times \{\bot, c_1, \ldots, c_n\}$.
Containment for UCQs over data trees

[Björklund, Martens, Schwentick ’08]

Can restrict to trees with data values $c_1, \ldots, c_{\|P\|}$ and distinct nulls.

- Let $T$ be a tree satisfying $P$ and not $Q$.
- $P$ touches $\leq \|P\|$ data values in $T$; replace with $c_1, \ldots, c_{\|P\|}$.
- In each node not touched by $P$ put a unique fresh data value.
- The resulting tree $T'$ still satisfies $P$ and not $Q$.

In such trees, $x \sim y$ holds iff either $x = y$ or $x \sim c_i$ and $y \sim c_i$.

By considering all possibilities, replace $P, Q$ with $P', Q'$ using only $x = y, x \sim c_i, y \sim c_i$.

Check containment over the finite alphabet $\Sigma \times \{\bot, c_1, \ldots, c_n\}$. 
Equivalence for Datalog

Equivalence for Datalog is undecidable:

- with descendant [Abiteboul, Bourhis, Muscholl, Wu 2013]
- for non-linear programs [Mazowiecki, Murlak, Witkowski 2014]
- for non-monadic programs (descendant is easily simulated).
Equivalence for Datalog

Equivalence for Datalog is undecidable:

- with descendant [Abiteboul, Bourhis, Muscholl, Wu 2013]
- for non-linear programs [Mazowiecki, Murlak, Witkowski 2014]
- for non-monadic programs (descendant is easily simulated).

Theorem (Mazowiecki, Murlak, Witkowski 2014)

Equivalence for linear monadic Datalog without desc is decidable.

Can’t we restrict reused datavalues like before?
Equivalence for Datalog

Equivalence for Datalog is undecidable:
▶ with descendant [Abiteboul, Bourhis, Muscholl, Wu 2013]
▶ for non-linear programs [Mazowiecki, Murlak, Witkowski 2014]
▶ for non-monadic programs (descendant is easily simulated).

Theorem (Mazowiecki, Murlak, Witkowski 2014)
Equivalence for linear monadic Datalog without desc is decidable.

Can’t we restrict reused datavalues like before?
▶ Let \( T \) be a tree satisfying \( P \) and not \( Q \).
▶ Then \( T \) satisfies some CQ \( P_0 \), an unravelling of \( P \).
▶ \( P_0 \) touches \( \leq \| P_0 \| \) data values in \( T \), like before,
▶ but \( \| P_0 \| \) can be arbitrarily large...
Example

\[
P \leftarrow DOWN_0(x)
\]

\[
DOWN_i(x) \leftarrow child(x, y) \land a(y) \land DOWN_{i+1}(y)
\]

\[
DOWN_N(x) \leftarrow UP_N(x) \land (N+1)-parent(x, y) \land child(y, z) \land c(z) \land x \sim z
\]

\[
UP_i(x) \leftarrow a(x) \land parent(x, y) \land child(y, z) \land b(z) \land DOWN_i(z)
\]

\[
UP_{i+1}(x) \leftarrow b(x) \land parent(x, y) \land UP_{i-1}(y)
\]

\[
UP_0(x) \leftarrow true
\]

\[
Q \leftarrow x \sim y \land i-parent(x, x') \land i-parent(y, y') \land a(x') \land b(y')
\]
Clique-width

Instead of processing structures, process their hierarchical decompositions (derivations).

Construct (derive) coloured structures using operations:

\[ i \rightarrow j \] – change colour \( i \) to \( j \);

\[ R(i_1, \ldots, i_r) \] – add to \( R \) all tuples of nodes with colours \( (i_1, \ldots, i_r) \);

\[ i \mapsto j \] – create a new node of colour \( i \);

\[ \oplus \] – take disjoint union of two structures.

clique-width(\( A \)) = least number of colours sufficient to construct \( A \)
Examples

Linear orders: clique-width 2
Examples

Linear orders: clique-width 2

Paths: clique-width 3
Trees: clique-width 3
Cographs: clique-width 2
Distance-hereditary graphs: clique-width 3
Graphs of tree-width $k$: clique-width $3 \cdot 2^k - 1$
Examples

Linear orders: clique-width 2

Paths: clique-width 3
Trees: clique-width 3
Cographs: clique-width 2
Distance-hereditary graphs: clique-width 3
Graphs of tree-width $k$: clique-width $3 \cdot 2^k - 1$
Examples

Linear orders: clique-width 2

```
red ↦→ yellow
yellow ≤ red
red  yellow
```

Paths: clique-width 3
Trees: clique-width 3
Cographs: clique-width 2
Distance-hereditary graphs: clique-width 3
Graphs of tree-width $k$: clique-width $3 \cdot 2^k - 1$
Examples

Linear orders: clique-width 2

Paths: clique-width 3
Trees: clique-width 3
Cographs: clique-width 2
Distance-hereditary graphs: clique-width 3
Graphs of tree-width $k$: clique-width $3 \cdot 2^{k-1}$
Examples

Linear orders: clique-width 2

Paths: clique-width 3
Trees: clique-width 3
Cographs: clique-width 2
Distance-hereditary graphs: clique-width 3
Graphs of tree-width $k$: clique-width $3 \cdot 2^k - 1$
Examples

Linear orders: clique-width 2

Paths: clique-width 3
Trees: clique-width 3
Cographs: clique-width 2
Distance-hereditary graphs: clique-width 3
Graphs of tree-width $k$: clique-width $3 \cdot 2^k - 1$
Examples

Linear orders: clique-width 2

Paths: clique-width 3
Trees: clique-width 3
Cographs: clique-width 2
Distance-hereditary graphs: clique-width 3
Graphs of tree-width $k$: clique-width $3 \cdot 2^k - 1$
Examples

Linear orders: clique-width 2

Paths: clique-width 3
Trees: clique-width 3
Cographs: clique-width 2
Distance-hereditary graphs: clique-width 3
Graphs of tree-width $k$: clique-width $3 \cdot 2^{k-1}$
Examples

Linear orders: clique-width 2

Paths: clique-width 3
Trees: clique-width 3
Cographs: clique-width 2
Distance-hereditary graphs: clique-width 3
Graphs of tree-width $k$: clique-width $3 \cdot 2^k - 1$
Examples

Linear orders: clique-width 2

Paths: clique-width 3
Examples

Linear orders: clique-width 2

\[
\begin{align*}
\text{red} &\leftrightarrow \text{yellow} \\
\text{yellow} &\leq \text{red} \\
\text{red} &\mapsto \text{yellow} \\
\text{red} &\leftrightarrow \text{yellow} \\
\text{yellow} &\leq \text{red} \\
\text{red} &\mapsto \text{yellow} \\
\text{red} &\leftrightarrow \text{yellow} \\
\text{yellow} &\leq \text{red} \\
\text{red} &\mapsto \text{yellow} \\
\end{align*}
\]

Paths: clique-width 3

Trees: clique-width 3

Cographs: clique-width 2

Distance-hereditary graphs: clique-width 3

Graphs of tree-width $k$: clique-width $3 \cdot 2^k - 1$
Examples

Linear orders: clique-width 2

Paths: clique-width 3

Trees: clique-width 3

Cographs: clique-width 2

Distance-hereditary graphs: clique-width 3

Graphs of tree-width $k$: clique-width $3 \cdot 2^{k-1}$
Bounded clique-width means simple

Many NP-complete problems are in P for graphs of bounded clique-width.

**Fixed-parameter tractable** with clique-width as parameter: time $f(k) \cdot n^c$ on inputs of size $n$ and clique-width at most $k$, where $f$ is some function, and $c$ is an absolute constant.

**Hamiltonicity**
Is there a path in graph $G$ that visits each node exactly once?

**3-colorability**
Can nodes of the graph $G$ be coloured so that each edge connects nodes of different colours?
Courcelle’s theorem

Monadic second order logic (MSO)

$$\varphi, \psi ::= R(x_1, \ldots, x_r) \mid \neg \varphi \mid \varphi \land \psi \mid \varphi \lor \psi \mid \exists x \varphi \mid \forall x \varphi \mid \exists X \varphi \mid \forall X \varphi \mid X(x)$$

3-colorability

$$\exists X_1 \exists X_2 \exists X_3 \ \forall x \ X_1(x) \lor X_2(x) \lor X_3(x) \land \forall x \forall y E(x, y) \Rightarrow \bigwedge_i \neg (X_i(x) \land X_i(y))$$

Theorem (Courcelle)

For every $$k \in \mathbb{N}$$ and $$\varphi \in \text{MSO}$$ one can construct an automaton recognizing $$k$$-derivations yielding models of $$\varphi$$. 
Courcelle’s theorem applied to parametrized complexity

Theorem (Courcelle)

For every $k \in \mathbb{N}$ and $\varphi \in \text{MSO}$ one can construct an automaton recognizing $k$-derivations yielding models of $\varphi$.

Corollary

Each set of structures definable in MSO can be decided in polynomial time over graphs of bounded cliquewidth.

- Compute $k$-derivation $e$ for the input structure (poly-time);
- construct the automaton $A$ for $k$ and the defining formula $\varphi$;
- run the automaton $A$ on $e$. 
Courcelle’s theorem applied to static analysis

Theorem (Courcelle)

For every $k \in \mathbb{N}$ and $\varphi \in \text{MSO}$ one can construct an automaton recognizing $k$-derivations yielding models of $\varphi$.

Corollary

For every $k \in \mathbb{N}$, it is decidable if given $\varphi \in \text{MSO}$ has a model of clique-width at most $k$.

- Construct the automaton $A$ for $k$ and the formula $\varphi$;
- test emptiness of the automaton $A$ (poly-time).
Datalog containment via bounded clique-width

[Bojańczyk, Murlak, Witkowski ’15]

Theorem
Let \( P, Q \) be monadic, linear Datalog programs without descendant. If \( P \land \neg Q \) is satisfiable, it is satisfiable in a data tree of clique-width at most \( 10 \cdot \| P \|^2 \).

Corollary
Containment for linear monadic Datalog programs without descendant is decidable.

- Rewrite monadic programs \( P, Q \) into \( \varphi_P, \varphi_Q \in \text{MSO} \).
- Write \( \varphi_{\text{datatree}} \in \text{MSO} \) saying that the structure is a data tree.
- Test satisfiability of \( \varphi_P \land \neg \varphi_Q \land \varphi_{\text{datatree}} \).
- For tight complexity, adjust Courcelle’s theorem to Datalog.
Problems

Old solutions

New solution

More problems with solutions

Some problems without solutions
A monadic Datalog program is downward if in all rules for $S(x)$, all mentioned nodes are descendants of $x$.

**Theorem**

Let $P, Q$ be downward Datalog programs. If $P \land \neg Q$ is satisfiable, it is satisfiable in a data tree of clique-width at most $5 \cdot \|P\|$. 

**Corollary**

Containment for downward Datalog programs is decidable.
Non-mixing constraints

[Czerwiński, David, Murlak, Parys ’16]

In database systems, correctness of data is expressed with integrity constraints:

\[\varphi(\bar{x}) \Rightarrow \alpha_\sim(\bar{x}) \quad \text{and} \quad \varphi(\bar{x}) \Rightarrow \alpha_\approx(\bar{x})\]

with \(\varphi \in UCQ(chd, desc, \Sigma), \alpha_\sim \in UCQ(\sim), \alpha_\approx \in UCQ(\approx)\).

Validity: Does each data tree of schema \(S\) satisfy set \(\Delta\) of non-mixing constraints?

Entailment: Does each data tree of schema \(S\) that satisfies \(\Delta\) also satisfies constraint \(\delta\)?

Theorem

*Both problems allow counter-examples of bounded clique-width.*
Problems

Old solutions

New solution

More problems with solutions

Some problems without solutions
Open problems

Containment of Datalog programs
  ▶ in the presence of a schema;
  ▶ with sibling order.

Non-mixing constraints with
  ▶ free use of comparisons with constants;
  ▶ Skolem functions.