Nondeterministic Unranked Tree Automata with Sibling Equality Constraints

Karianto Wong

RWTH Aachen University

FSTTCS 2009

Joint work with Christof Löding
Unranked Tree Automata with Sibling Equalities

- Bottom-up tree automaton $\mathcal{A} = (Q, \Sigma, \Delta, F)$
- Finite state set $Q$ with final states $F \subseteq Q$
- Symbols in $\Sigma$ have no fixed arities
- Transitions:

$$L \subseteq Q^* \text{ regular}$$

equality constraints between sibling subtrees, e.g.:

```
q_1 \cdots q_k \in L
```

```
t_1
```

```
t_k
```

```
q
```

```
a
```
Unranked Tree Automata with Sibling Equalities

- Bottom-up tree automaton \( \mathcal{A} = (Q, \Sigma, \Delta, F) \)
- Finite state set \( Q \) with final states \( F \subseteq Q \)
- Symbols in \( \Sigma \) have no fixed arities
- Transitions: \( (L, \alpha, a, q) \)
  - \( L \subseteq Q^* \) regular
  - \( a \) current symbol
  - \( q \) target state

Equality constraints between sibling subtrees, e.g.:

- “first=last”
- “all subtrees are equal”
- “first=last, but both are different from all others”
Motivations

- Natural extension of ranked tree automata with equality constraints between subtrees
  \(\rightsquigarrow\) [Mongy, Bogaert & Tison, Tommasi, \ldots (in the 80’s & 90’s)]

- Unranked trees: formal model for semi-structured data

- Trees encode data (e.g. natural numbers)
  \(\rightsquigarrow\) data words can be coded as trees:

- Automata on data words: test equality between data
  (see, e.g., survey by [Segoufin’06])
  \(\rightsquigarrow\) data equalities \(\approx\) subtree equalities
Outline

Automata with Constraints between Siblings Subtrees

Emptiness Problem

UTACS and Data Languages
Outline

Automata with Constraints between Siblings Subtrees

Emptiness Problem

UTACS and Data Languages
Constraints among Unboundedly Many Siblings

Symbols have no fixed arities $\leadsto$ unbounded number of sibling pairs to be compared

Example:

“first and last subtrees are equal, but different from the others”
Constraints among Unboundedly Many Siblings

Symbols have no fixed arities \( \sim \) unbounded number of sibling pairs to be compared

Example:

“first and last subtrees are equal, but different from the others”

First idea:

- use monadic second-order logic over state sequences:

\[
\varphi ::= x < y \mid \text{Succ}(x, y) \mid \text{Lab}_q(x) \mid X(x) \\
\mid \psi \lor \theta \mid \neg \psi \mid \exists x. \psi \mid \exists X. \psi
\]

- extend the vocabulary by subtree equality; e.g.:

\[
\exists x \exists y \left( x = \min \land y = \max \land t_x = t_y \land \forall z (z \neq x \land z \neq y \rightarrow t_z \neq t_x) \right)
\]
Constraints among Unboundedly Many Siblings

Symbols have no fixed arities \(\leadsto\) unbounded number of sibling pairs to be compared

Example:

“first and last subtrees are equal, but different from the others”

First idea:

- use monadic second-order logic over state sequences:

\[
\varphi ::= \ x < y \mid \text{Succ}(x, y) \mid \text{Lab}_q(x) \mid X(x) \\
\quad \mid \varphi \lor \theta \mid \neg \varphi \mid \exists x. \varphi \mid \exists X. \varphi
\]

- extend the vocabulary by subtree equality; e.g.:

\[
\exists x \exists y (x = \text{min} \land y = \text{max} \land t_x = t_y \land \forall z (z \neq x \land z \neq y \rightarrow t_z \neq t_x))
\]

\(\leadsto\) Emptiness is undecidable since using trees we can encode data words, and satisfiability of FO logic over data words is undecidable [Bojańczyk et al.'06].
Constraints among Unboundedly Many Siblings – cont’d

Idea: separate addressing and subtree comparison

\[ \phi(x, y) \]: MSO-formula with free \( x, y \)

\( \leadsto \) use MSO-formula only to address pairs of positions to be compared
Constraints among Unboundedly Many Siblings – cont’d

Idea: separate addressing and subtree comparison

\[ \leadsto \text{use MSO-formula only to address pairs of positions to be compared} \]

Four types of atomic sibling constraints:

1. \( \forall x \forall y . \varphi(x, y) \rightarrow t_x = t_y \)

\( \varphi(x, y) \): MSO-formula with free \( x \), \( y \)

\[
\begin{array}{c}
\text{if} \quad q_1 \cdots q_x \cdots q_y \cdots q_k \\
\text{then} \quad t_x = t_y
\end{array}
\]
Constraints among Unboundedly Many Siblings – cont’d

Idea: separate addressing and subtree comparison

⇝ use MSO-formula only to address pairs of positions to be compared

Four types of atomic sibling constraints:

- $\forall x \forall y . \varphi(x, y) \rightarrow t_x = t_y$
- $\forall x \forall y . \varphi(x, y) \rightarrow t_x \neq t_y$

$\varphi(x, y)$: MSO-formula with free $x, y$
Constraints among Unboundedly Many Siblings – cont’d

Idea: separate addressing and subtree comparison

⇝ use MSO-formula only to address pairs of positions to be compared

Four types of atomic sibling constraints:

▶ $\forall x \forall y. \varphi(x, y) \rightarrow t_x = t_y$

▶ $\forall x \forall y. \varphi(x, y) \rightarrow t_x \neq t_y$

▶ $\exists x \exists y. \varphi(x, y) \land t_x = t_y$

$\varphi(x, y):$ MSO-formula with free $x, y$

for some $x, y$

and $t_x = t_y$
Constraints among Unboundedly Many Siblings – cont’d

Idea: separate addressing and subtree comparison
⇝ use MSO-formula only to address pairs of positions to be compared

Four types of atomic sibling constraints:

- $\forall x \forall y . \varphi(x, y) \rightarrow t_x = t_y$
- $\forall x \forall y . \varphi(x, y) \rightarrow t_x \neq t_y$
- $\exists x \exists y . \varphi(x, y) \land t_x = t_y$
- $\exists x \exists y . \varphi(x, y) \land t_x \neq t_y$

$\varphi(x, y)$: MSO-formula with free $x, y$

For some $x, y$ and $t_x \neq t_y$
Constraints among Unboundedly Many Siblings – cont’d

Idea: separate addressing and subtree comparison
⇝ use MSO-formula only to address pairs of positions to be compared

Four types of atomic sibling constraints:

\[ \forall x \forall y \cdot \varphi(x, y) \rightarrow t_x = t_y \]
\[ \forall x \forall y \cdot \varphi(x, y) \rightarrow t_x \neq t_y \]
\[ \exists x \exists y \cdot \varphi(x, y) \land t_x = t_y \]
\[ \exists x \exists y \cdot \varphi(x, y) \land t_x \neq t_y \]

**Sibling constraints:** Boolean combinations of atomic constraints

**Example:** “first and last subtree are equal, but different from the others”
\[ [ \forall x \forall y \cdot x = \text{min} \land y = \text{max} \rightarrow t_x = t_y ] \land \\
[ \forall x \forall y \cdot ((x = \text{min} \land x < y < \text{max}) \lor (y = \text{max} \land \text{min} < x < y)) \rightarrow t_x \neq t_y ] \]
**Unranked Tree Automaton with Constraints between Siblings:**

- Bottom-up tree automaton \( \mathcal{A} = (Q, \Sigma, \Delta, F) \)
- Finite state set \( Q \) with final states \( F \subseteq Q \)
- Finite, unranked alphabet \( \Sigma \)
- Transitions in \( \Delta \):
  \[
  (L, \alpha, a, q)
  \]
  \( \Delta \) contains \( q \) for \( q \in F \)

**Remarks:**

- MSO-formulas only used as addressing mechanism
- No reuse of formulas in other constraints allowed
Unranked Tree Automaton with Constraints between Siblings:

- Bottom-up tree automaton $\mathcal{A} = (Q, \Sigma, \Delta, F)$
- Finite state set $Q$ with final states $F \subseteq Q$
- Finite, unranked alphabet $\Sigma$
- Transitions in $\Delta$:

\[
L, \alpha, a, q \subseteq Q^* \text{ regular}
\]

equality constraints between sibling subtrees

- Remarks:
  - MSO-formulas only used as addressing mechanism
  - No reuse of formulas in other constraints allowed

**Theorem** [Löding&Wong’07]. Nondeterministic UTACS are strictly more powerful than deterministic UTACS.
Outline

Automata with Constraints between Siblings Subtrees

Emptiness Problem

UTACS and Data Languages
**Emptiness Decidability**

**Theorem.** Emptiness for (nondeterministic) UTACS is decidable.

In [Löding&Wong’07]: decidability for deterministic UTACS.

In this paper: decidability for nondeterministic UTACS.

- The methods used are basically the same as in deterministic case ...  
- ... but a lot more technicalities are required.
Deciding Emptiness – the Deterministic Case

Generic emptiness algorithm for bottom-up tree automaton:

- Iteratively mark states reachable by some tree (and keep the tree).
- In each round: check whether some transition can be applied using trees that are currently available.

Checking applicability of transitions w.r.t. equality constraints:
- By determinism, reduce distinction between trees to distinction between states.
  - If the states reached are different, then so are the trees.
- Example: if \( t_2 \neq t_3 \) is required, it suffices to know \( q_2 \neq q_3 \).
- For each state, collect a certain number of trees.
  - Example: if \( t_2 \neq t_4 \) is required, then the transition can only be applied if there are at least two trees evaluating to \( q_2 \).
Deciding Emptiness – the Deterministic Case

Generic emptiness algorithm for bottom-up tree automaton:

- Iteratively mark states \textit{reachable} by some tree (and keep the tree).
- In each round: check \textit{whether some transition can be applied} using trees that are currently available.

Checking applicability of transitions w.r.t. \textit{equality constraints}:

- By determinism, reduce distinction between trees to distinction between states.
  \[\sim\] \textit{If the states reached are different, then so are the trees}

Example: if “\(t_2 \neq t_3\)” is required, it suffices to know \(q_2 \neq q_3\)
Deciding Emptiness – the Deterministic Case

Generic emptiness algorithm for bottom-up tree automaton:

- Iteratively mark states \textit{reachable} by some tree (and keep the tree).
- In each round: check \textit{whether some transition can be applied} using trees that are currently available.

Checking applicability of transitions w.r.t. \textit{equality constraints}:

- By determinism, reduce distinction between trees to distinction between states.
  \begin{itemize}
  \item If the states reached are different, then so are the trees
  \end{itemize}
  \textbf{Example:} if \textit{“}t_2 \neq t_3\textit{”} is required, it suffices to know \textit{q}_2 \neq \textit{q}_3

- For each state, collect a certain number of trees.
  \textbf{Example:} if \textit{“}t_2 \neq t_4\textit{”} is required, then the transition can only be applied if there are \textit{at least two} trees evaluating to \textit{q}_2.
... But How Many Trees to Collect?

Ranked setting: number of distinct trees needed $\leq$ maximal rank
$\leadsto$ bound on the number of trees to collect

Unranked setting: ?
... But How Many Trees to Collect?

Ranked setting: number of distinct trees needed $\leq$ maximal rank ⇾ bound on the number of trees to collect

Unranked setting:

**Lemma.** There exists a bound $B$ such that: for each application of a transition $\tau = (L, \alpha, a, q)$ using $w = q_1 \ldots q_k$, there is a replacement $w' = q_1' \ldots q_\ell'$ such that the application of $\tau$ using $w'$ needs $\leq B$ distinct trees for each state.

Remark: Our proof yields only non-elementary upper bound for $B$. 
The Nondeterministic Case

Key observations in deterministic case: focus on the (unique) state reached by a tree

If the states reached are distinct, then so are the trees

For each state, a certain number of trees are needed
The Nondeterministic Case

Key observations in deterministic case: focus on the (unique) state reached by a tree

If the states reached are distinct, then so are the trees

Nondeterministic case: pseudo-determinization directly in the algorithm

Proceed from states to sets of states!

Further ingredients: consider collections of transitions instead of single transitions
Outline

Automata with Constraints between Siblings Subtrees

Emptiness Problem

UTACS and Data Languages
Encoding Data Words as Unranked Trees

- label (finite alphabet)
- data (infinite domain, e.g. \(\mathbb{N}\))

\[(a, 2)(b, 3)(c, 1)(c, 2) \sim \begin{array}{c}
\top \\
a \\
b \\
c \\
c \\
\end{array} \]

- labels at odd positions
- data at even positions
- comparison between data values \(\approx\) comparison between subtrees at even positions

\(\Rightarrow\) Use UTACS to define languages of data words.

\(\Rightarrow\) Emptiness is decidable for these languages of data words.
Encoding Data Words as Unranked Trees

(a, 2)(b, 3)(c, 1)(c, 2) ~\rightarrow

- labels at odd positions
- data at even positions
- comparison between data values ≈ comparison between subtrees at even positions

Use UTACS to define languages of data words.

Emptiness is decidable for these languages of data words.
Certain formulas of logic over data words (with data equality \( \sim \)) can be translated into UTACS, namely formulas corresponding to sibling constraints.

**Example:** “The data at first and last position are equal, but different from the data at the other positions”

\[
\begin{align*}
& \forall x \forall y. \, x = \min \land y = \max \rightarrow x \sim y \\
& \forall x \forall y. \, ((x = \min \land x < y < \max) \lor (y = \max \land \min < x < y)) \rightarrow x \not\sim y
\end{align*}
\]

\( \Rightarrow \) For such formulas, satisfiability reduces to emptiness of UTACS and is thus decidable.
A Decidable Logic for Data Languages

Certain formulas of logic over data words (with data equality $\sim$) can be translated into UTACS, namely formulas corresponding to sibling constraints.

Example: “The data at first and last position are equal, but different from the data at the other positions”

$$\forall x \forall y . x = \min \land y = \max \rightarrow x \sim y$$

$$\land$$

$$\forall x \forall y . ((x = \min \land x < y < \max) \lor (y = \max \land \min < x < y)) \rightarrow x \not\sim y$$

$\leadsto$ For such formulas, satisfiability reduces to emptiness of UTACS and is thus decidable.

Another example: “between every two positions with the same data value, there exists a position labeled with $a$”

$$\forall x \forall y . \neg \exists z . (x < z < y \land \text{Lab}_a(z)) \rightarrow x \not\sim y$$

$\leadsto$ It is still open whether this language is definable in $\text{FO}^2(\sim, <, \text{Succ})$ [Bojanczyk et al ’06]
Conclusion

Summary:
- Use of MSO formulas as constraint addressing mechanism
- Emptiness is decidable
- Connection with languages of data words
- Universality is undecidable
Conclusion

Summary:

- Use of MSO formulas as constraint addressing mechanism
- Emptiness is decidable
- Connection with languages of data words
- Universality is undecidable

Open problems & further prospects:

- Complexity issues
- Comparing trees w.r.t. other relations, e.g., structural equality
**Theorem.** *Universality is undecidable for nondeterministic UTACS.*

Proof sketch:

- reduce the **halting problem** for 2-register machines
- encode computations \((p_1, d_1, e_1) \ldots (p_m, d_m, e_m)\) as a tree:

```
⊤
p_1⊥ a … a ⊥ b … b $ … $p_m⊥ a … a ⊥ b … b
|    |    |    |    |    |    |    |    
  i_{11}  i_{1d_1} j_{11} j_{1e_1}  i_{11}  i_{md_m} j_{m1} j_{me_m}
```

- construct nondeterministic UTACS accepting all trees that do not represent a halting computation

\[\leadsto\] halting computation exists \iff some tree is not accepted