Nondeterministic Unranked Tree Automata with Sibling Equality Constraints

Karianto Wong

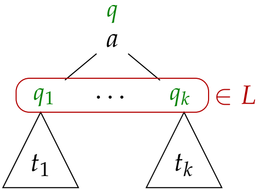
RWTH Aachen University

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 Σ have no fixed arities
- Transitions:



$$(\, L \, , \, \alpha \, , \, a \, , \, q \,)$$

$L \subseteq Q^*$ regular

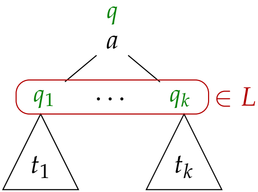
target state

current symbol

equality constraints between sibling subtrees, e.g.:

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 Σ have no fixed arities
- Transitions:



$$(L \, , \, \alpha \, , \, a \, , \, q \,)$$

$L \subseteq Q^*$ regular

target state

current symbol

equality constraints between sibling subtrees, e.g.:

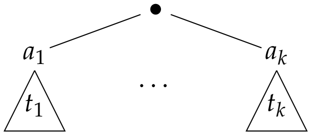
- “first=last”
- “all subtrees are equal”
- “first=last, but both are different from all others”

Motivations

- Natural extension of ranked tree automata with equality constraints between subtrees
  ⇝ [Mongy, Bogaert & Tison, Tommasi, … (in the 80's & 90's)]
- Unranked trees: formal model for semi-structured data
- Trees encode data (e.g. natural numbers)
  ⇝ data words can be coded as trees:

$$(a_1, i_1) \ldots (a_k, i_k) \quad \rightsquigarrow$$

label   data



Automata on data words: test equality between data (see, e.g., survey by [Segoufin'06])

⇝ data equalities ≈ subtree equalities

Outline

Automata with Constraints between Siblings Subtrees

Emptiness Problem

UTACS and Data Languages

Outline

Automata with Constraints between Siblings Subtrees

Emptiness Problem

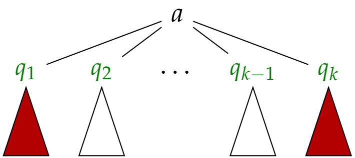
UTACS and Data Languages

Constraints among Unboundedly Many Siblings

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

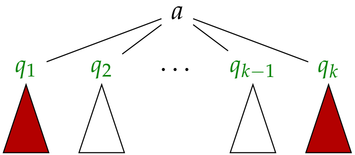
Example:

> "first and last subtrees are equal,
> but different from the others"



Constraints among Unboundedly Many Siblings

Symbols have no fixed arities $\rightsquigarrow$ 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:

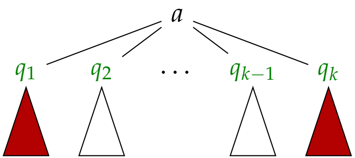
$$\begin{aligned} \varphi \; ::= \quad & x < y \;\mid\; \mathrm{Succ}(x,y) \;\mid\; \mathrm{Lab}_q(x) \;\mid\; X(x) \\ & \mid\; \psi \vee \theta \;\mid\; \neg\psi \;\mid\; \exists x.\psi \;\mid\; \exists X.\psi \end{aligned}$$

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

$$\exists x \exists y \,(\, x = \min \wedge y = \max \wedge t_x = t_y \wedge \forall z \,(\, z \neq x \wedge z \neq y \;\rightarrow\; t_z \neq t_x \,)\,)$$

Constraints among Unboundedly Many Siblings

Symbols have no fixed arities $\rightsquigarrow$ 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:

$$\begin{aligned}\varphi \ ::= \quad & x < y \ \mid \ \mathrm{Succ}(x,y) \ \mid \ \mathrm{Lab}_q(x) \ \mid \ X(x) \\ \mid \ & \psi \vee \theta \ \mid \ \neg\psi \ \mid \ \exists x.\psi \ \mid \ \exists X.\psi\end{aligned}$$

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

$$\exists x \exists y \, (\, x = \min \wedge y = \max \wedge t_x = t_y \wedge \forall z \, (\, z \neq x \wedge z \neq y \ \rightarrow \ t_z \neq t_x \,) \,)$$

$\rightsquigarrow$ 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

$\rightsquigarrow$ use MSO-formula only to address pairs of positions to be compared

Constraints among Unboundedly Many Siblings – cont'd

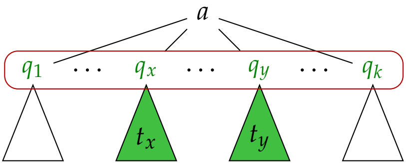
Idea: separate addressing and subtree comparison

$\rightsquigarrow$ use MSO-formula only to address pairs of positions to be compared

Four types of atomic sibling constraints:

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

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

if


$\models \varphi(x,y)$ then $t_x = t_y$

Constraints among Unboundedly Many Siblings – cont'd

Idea: separate addressing and subtree comparison

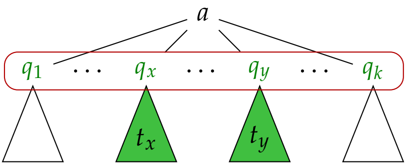
$\rightsquigarrow$ use MSO-formula only to address pairs of positions to be compared

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

if



$\models \varphi(x,y)$ then $t_x \neq t_y$

Constraints among Unboundedly Many Siblings – cont'd

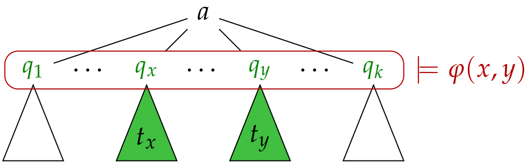
Idea: separate addressing and subtree comparison

$\rightsquigarrow$ use MSO-formula only to address pairs of positions to be compared

Four types of atomic sibling constraints:

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

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



for some x, y

and $t_x = t_y$

Constraints among Unboundedly Many Siblings – cont'd

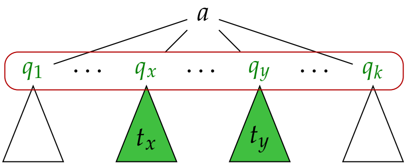
Idea: separate addressing and subtree comparison

$\rightsquigarrow$ use MSO-formula only to address pairs of positions to be compared

Four types of atomic sibling constraints:

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

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



$\models \varphi(x,y)$ for some x, y

and $t_x \neq t_y$

Constraints among Unboundedly Many Siblings – cont'd

Idea: separate addressing and subtree comparison

$\rightsquigarrow$ use MSO-formula only to address pairs of positions to be compared

Four types of atomic sibling constraints:

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

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

Sibling constraints: Boolean combinations of atomic constraints

Example: "first and last subtree are equal, but different from the others"

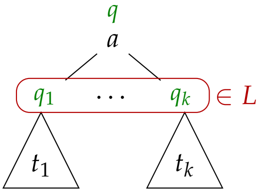
$$\big[\, \forall x \forall y \,.\, x = \min \wedge y = \max \;\rightarrow\; t_x = t_y \,\big] \wedge$$

$$\big[\, \forall x \forall y \,.\, \big((x = \min \wedge x < y < \max) \vee (y = \max \wedge \min < x < y)\big) \rightarrow\; t_x \neq t_y \,\big]$$

UTACS

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 Σ
- Transitions in Δ:



$$(\, L \, , \, \alpha \, , \, a \, , \, q \,)$$

$L \subseteq Q^*$ regular

equality constraints between sibling subtrees

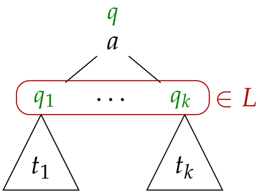
Remarks:

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

UTACS

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 Σ
- Transitions in Δ:



$$(L , \alpha , a , q)$$

$L \subseteq Q^*$ 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.

Deciding Emptiness – the Deterministic Case

Generic emptiness algorithm for bottom-up tree automaton:

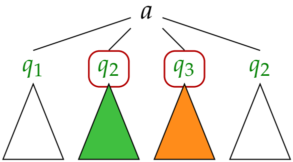
- Iteratively mark states reachable by some tree (and keep the tree).
- In each round: check whether some transition can be applied using trees that are currently available.

Checking applicability of transitions w.r.t. equality constraints:

- By determinism, reduce distinction between trees to distinction between states.

  $\rightsquigarrow$ 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 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:

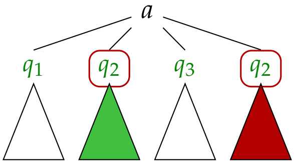
- By determinism, reduce distinction between trees to distinction between states.

  $\rightsquigarrow$ 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.

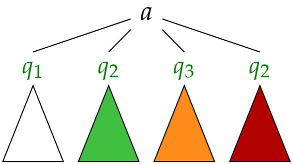
  Example: if "$t_2 \neq t_4$" is required, then the transition can only be applied if there are at least two trees evaluating to q_2.



... But How Many Trees to Collect?

Ranked setting: number of distinct trees needed $\leq$ maximal rank

$\rightsquigarrow$ bound on the number of trees to collect



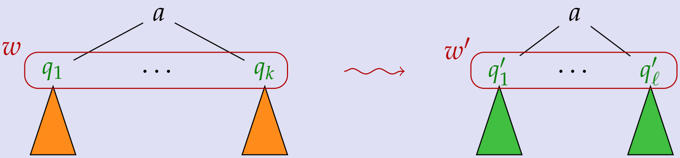
Unranked setting: ?

. . . But How Many Trees to Collect?

Ranked setting: number of distinct trees needed $\leq$ maximal rank
$\rightsquigarrow$ 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 τ using w' needs $\leq B$ distinct trees for each state.*

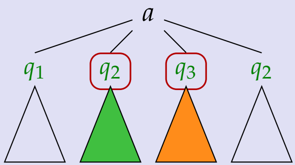


Remark: Our proof yields only non-elementary upper bound for B.

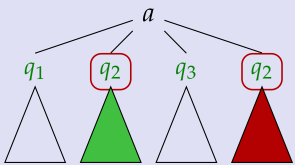
The Nondeterministic Case

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

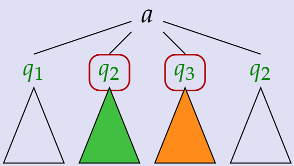


The Nondeterministic Case

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

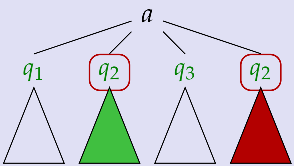
sets of states

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



set of states

For each ~~state,~~ a certain number of trees are needed



Nondeterministic case: pseudo-determinization directly in the algorithm

$\rightsquigarrow$ 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

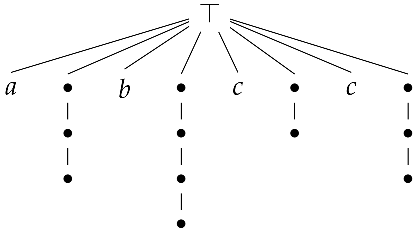
UTACS and Data Languages

Encoding Data Words as Unranked Trees

label (finite alphabet)

$$(a,2)(b,3)(c,1)(c,2) \rightsquigarrow$$

data (infinite domain, e.g. $\mathbb{N}$)



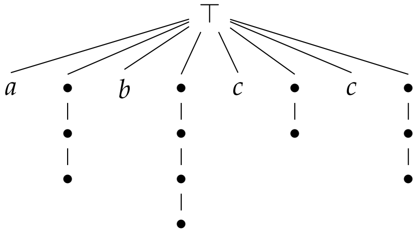
- labels at odd positions
- data at even positions
- comparison between data values $\approx$ comparison between subtrees at even positions

Encoding Data Words as Unranked Trees

label (finite alphabet)

$(a,2)(b,3)(c,1)(c,2) \quad \rightsquigarrow$

data (infinite domain, e.g. $\mathbb{N}$)



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

$\rightsquigarrow$ Use UTACS to define languages of data words.

$\rightsquigarrow$ Emptiness is decidable for these languages of data words.

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”

$$\left[\ \forall x \forall y \,.\, x = \min \wedge y = \max \ \rightarrow\ x \sim y\ \right] \wedge$$

$$\left[\forall x \forall y \,.\, \big((x = \min \wedge x < y < \max) \vee (y = \max \wedge \min < x < y)\big) \rightarrow\ x \not\sim y\right]$$

$\rightsquigarrow$ 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"

$$\left[\ \forall x \forall y \,.\, x = \min \wedge y = \max \ \rightarrow\ x \sim y\ \right] \wedge$$

$$\left[\forall x \forall y \,.\, \big((x = \min \wedge x < y < \max) \vee (y = \max \wedge \min < x < y)\big) \rightarrow\ x \not\sim y\right]$$

$\rightsquigarrow$ 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"

$$\left[\ \forall x \forall y \,.\, \neg\, \exists z \,.\, \big(\ x < z < y\ \wedge\ \mathrm{Lab}_a(z)\,\big) \rightarrow\ x \not\sim y\ \right]$$

$\rightsquigarrow$ It is still open whether this language is definable in $\mathrm{FO}^2(\sim, <, \mathrm{Succ})$ [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

Outline

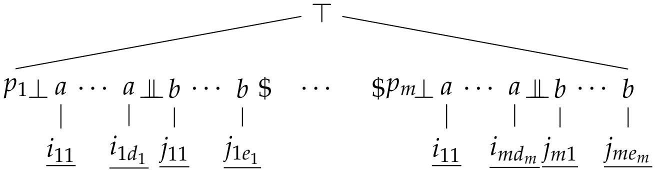
Appendix

Universality

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:



$\underline{i}$: unary tree representing natural number i

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

$\rightsquigarrow$ halting computation exists $\iff$ some tree is not accepted