Unranked Tree Automata with Sibling Equalities and Disequalities

Wong Karianto Christof Löding

Lehrstuhl für Informatik 7
RWTH Aachen

ICALP 2007
Wrocław, 9–13 July 2007

RWTH AACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

Motivations(1): Tree Automata with Subtree Equalities

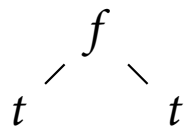
Finite (bottom-up) tree automata:

- ranked alphabet: each symbol has fixed rank/arity
- nice closure properties, determinizable
- decidable emptiness

Motivations(1): Tree Automata with Subtree Equalities

Finite (bottom-up) tree automata:

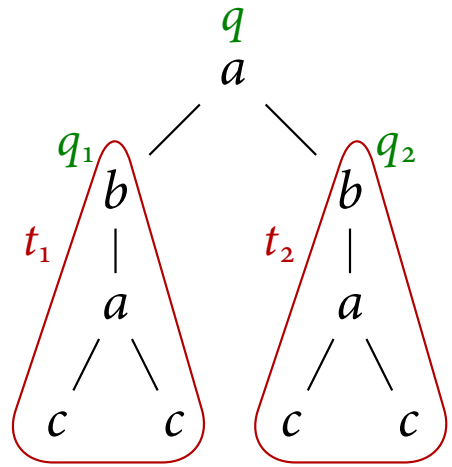
- ranked alphabet: each symbol has fixed rank/arity
- nice closure properties, determinizable
- decidable emptiness

But language of trees  is not recognizable

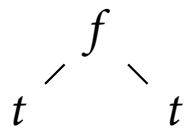
Motivations(1): Tree Automata with Subtree Equalities

Finite (bottom-up) tree automata:

- ranked alphabet: each symbol has fixed rank/arity
- nice closure properties, determinizable
- decidable emptiness



$(q_1, q_2, t_1 = t_2, a, q)$

But language of trees  is not recognizable

$\rightsquigarrow$ tree automata with sibling equality constraints (Rec$_{\neq}$ [Bogaert&Tison'92])

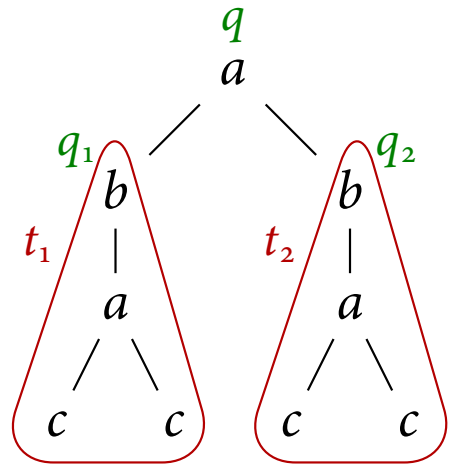
- transitions: $(q_1, \ldots, q_k, \alpha, a, q)$
- α: Boolean combination of $t_i = t_j$, $t_i \neq t_j$ with $i, j \in \{1, \ldots, k\}$

  Example: "$t_1 = t_2$" means "left and right subtree are equal"
- closed under Boolean operations, determinizable
- decidable emptiness

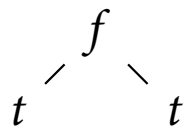
Motivations(1): Tree Automata with Subtree Equalities

Finite (bottom-up) tree automata:

- ranked alphabet: each symbol has fixed rank/arity
- nice closure properties, determinizable
- decidable emptiness



$(q_1, q_2, t_1 = t_2, a, q)$

But language of trees  is not recognizable

$\rightsquigarrow$ tree automata with sibling equality constraints
(Rec$_{\neq}$ [Bogaert&Tison'92])

- transitions: $(q_1, \ldots, q_k, \alpha, a, q)$
- α: Boolean combination of $t_i = t_j$, $t_i \neq t_j$ with $i, j \in \{1, \ldots, k\}$

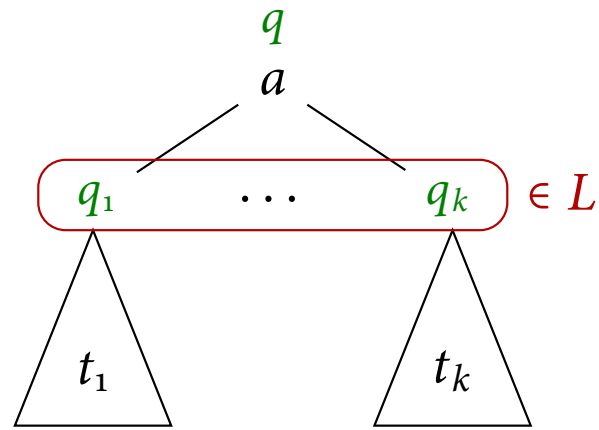
  Example: "$t_1 = t_2$" means "left and right subtree are equal"
- closed under Boolean operations, determinizable
- decidable emptiness

In this talk: extend the class Rec$_{\neq}$ to unranked trees

Motivations(2): Unranked Trees

Unranked trees:

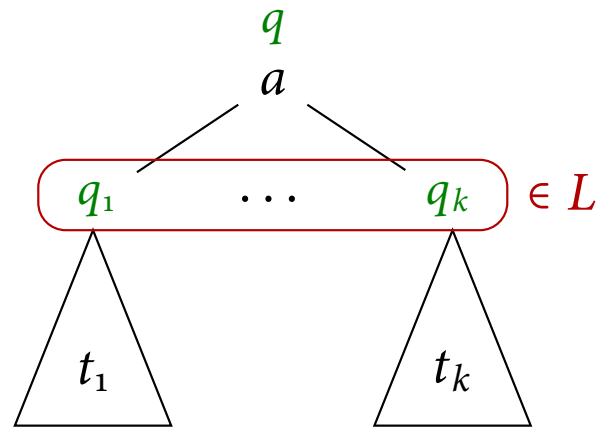
- Formal model for XML documents
- Symbols have no fixed arities
- Number of successors of a node is unbounded
- Finite unranked tree automata: transitions (L, a, q) with L regular language over the state set



Motivations(2): Unranked Trees

Unranked trees:

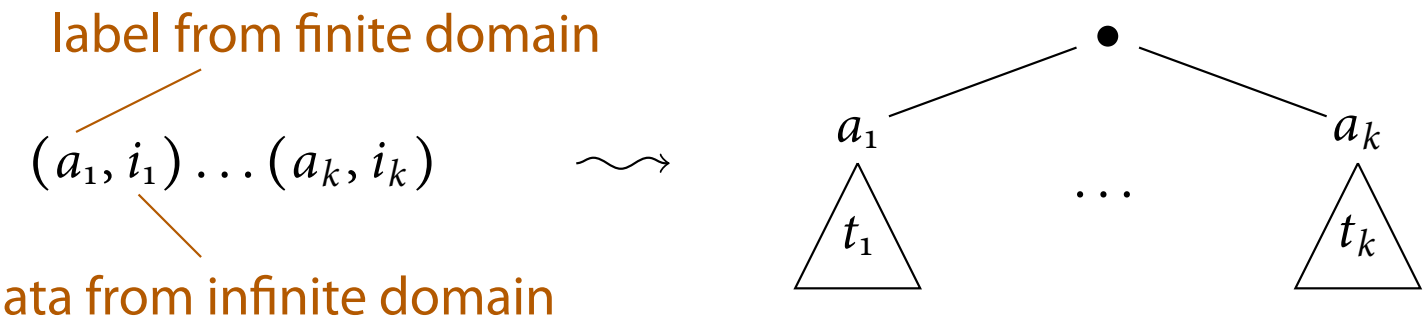
- Formal model for XML documents
- Symbols have no fixed arities
- Number of successors of a node is unbounded
- Finite unranked tree automata: transitions (L, a, q) with L regular language over the state set



Subtree equality in unranked trees:

- Subtrees as data encoding (e.g. of natural numbers)

  For instance, data words (e.g. [Bojańczyk et al.'06]) can be coded as trees:



  Automata on data words: test equality between data
  $\rightsquigarrow$ data equalities $\approx$ subtree equalities

Outline

Unranked Tree Automata with Sibling (Dis)Equalities

Decidability of Emptiness: the Deterministic Case

Outline

Unranked Tree Automata with Sibling (Dis)Equalities

Decidability of Emptiness: the Deterministic Case

Constraints among Unboundedly Many Siblings

Transition (L, α, a, q) with $L \subseteq Q^*$ regular

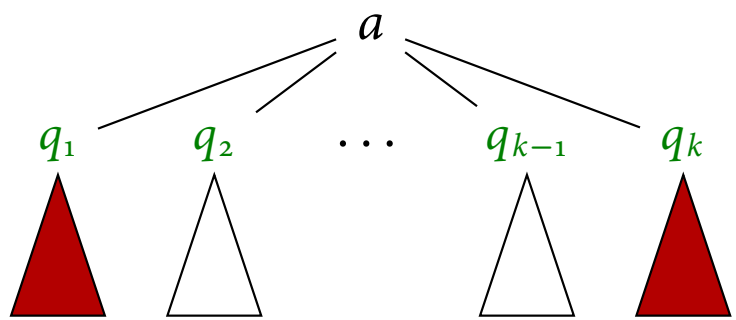
- Number of successors of a node is finite, but unbounded
- Constraint α has to consider all possible number of successors.
- Use of regular $L \subseteq Q^*$ allows finite representation despite unbounded length.

Constraints among Unboundedly Many Siblings

Transition (L, α, a, q) with $L \subseteq Q^*$ regular

- Number of successors of a node is finite, but unbounded
- Constraint α has to consider all possible number of successors.
- Use of regular $L \subseteq Q^*$ allows finite representation despite unbounded length.

Example constraint:



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

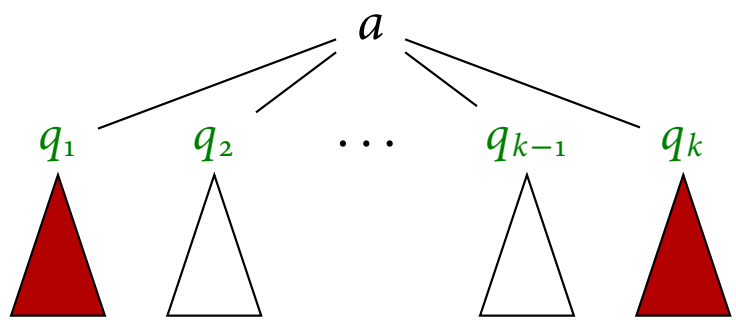
- for fixed number of successors k: "$t_1 = t_k \wedge \bigwedge_{1<i<k} t_1 \neq t_i$"
- k is unbounded in unranked case
  $\rightsquigarrow$ we need a formalism to define unbounded number of constraints while still allowing finite representation

Constraints among Unboundedly Many Siblings

Transition (L, α, a, q) with $L \subseteq Q^*$ regular

- Number of successors of a node is finite, but unbounded
- Constraint α has to consider all possible number of successors.
- Use of regular $L \subseteq Q^*$ allows finite representation despite unbounded length.

Example constraint:



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

- for fixed number of successors k: "$t_1 = t_k \wedge \bigwedge_{1<i<k} t_1 \neq t_i$"
- k is unbounded in unranked case
  $\rightsquigarrow$ we need a formalism to define unbounded number of constraints while still allowing finite representation

Proposal: use logic formulas over Q-sequences

Using Logics over Q-Sequences

First-order (FO) / monadic second-order (MSO) logic over Q-sequences:

- interpreted in word structures $w = q_1 \dots q_k \in Q^+$
- FO-variables $x, y, z, \dots$ over positions in $\{1, \dots, k\}$
- MSO-variables $X, Y, Z, \dots$ over subsets of $\{1, \dots, k\}$
- $\varphi ::= x < y \mid x = y \mid X(x) \mid q(x) \mid \psi \vee \theta \mid \neg\psi \mid \exists x.\psi \mid \exists X.\psi$
- write $w \vDash \varphi$ for "w satisfies φ"

label q at position x

Using Logics over Q-Sequences

First-order (FO) / monadic second-order (MSO) logic over Q-sequences:

- interpreted in word structures $w = q_1 \dots q_k \in Q^+$
- FO-variables $x, y, z, \dots$ over positions in $\{1, \dots, k\}$
- MSO-variables $X, Y, Z, \dots$ over subsets of $\{1, \dots, k\}$
- $\varphi ::= x < y \mid x = y \mid X(x) \mid q(x) \mid \psi \vee \theta \mid \neg\psi \mid \exists x.\psi \mid \exists X.\psi$
- write $w \vDash \varphi$ for "w satisfies φ"

label q at position x

Saying "first and last subtrees are equal, but different from the others" with logic formulas:

- intuitively: extend the vocabulary by subtree equality and say

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

Using Logics over Q-Sequences

First-order (FO) / monadic second-order (MSO) logic over Q-sequences:

- interpreted in word structures $w = q_1 \dots q_k \in Q^+$
- FO-variables $x, y, z, \dots$ over positions in $\{1, \dots, k\}$
- MSO-variables $X, Y, Z, \dots$ over subsets of $\{1, \dots, k\}$
- $\varphi ::= x < y \mid x = y \mid X(x) \mid q(x) \mid \psi \vee \theta \mid \neg\psi \mid \exists x.\psi \mid \exists X.\psi$
- write $w \vDash \varphi$ for "w satisfies φ"

label q at position x

Saying "first and last subtrees are equal, but different from the others" with logic formulas:

- intuitively: extend the vocabulary by subtree equality and say

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

- But this could lead to an automaton with undecidable emptiness 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 formula only to address pairs of positions to be compared

Atomic constraint: $(\ \varphi(x, y)\ ,\ type\ )$

Constraint types: $\exists_{\mathsf{EQ}}\ ,\ \forall_{\mathsf{EQ}}\ ,\ \exists_{\mathsf{NEQ}}\ ,\ \forall_{\mathsf{NEQ}}$

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

Constraints among Unboundedly Many Siblings – cont'd

Idea: separate addressing and subtree comparison

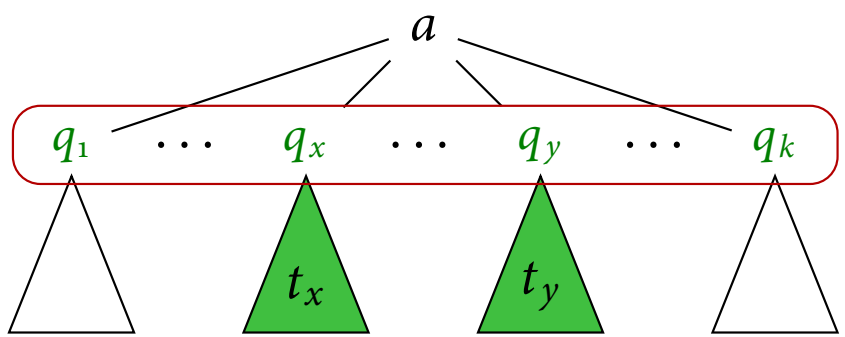
$\rightsquigarrow$ use formula only to address pairs of positions to be compared

Atomic constraint: $(\ \varphi(x, y)\ ,\ type\ )$ $\varphi(x, y)$: MSO-formula with two free variables

Constraint types: $\exists_{\text{EQ}}$, $\forall_{\text{EQ}}$, $\exists_{\text{NEQ}}$, $\forall_{\text{NEQ}}$

Semantics: $w = q_1 \dots q_k \in Q^*$ and tree sequence $t_1 \dots t_k$ satisfy $(\varphi, type)$ if:

- $\exists_{\text{EQ}}$: there exist positions x, y in w with $w \vDash \varphi(x, y)$ and $t_x = t_y$



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

and $t_x = t_y$

Constraints among Unboundedly Many Siblings – cont'd

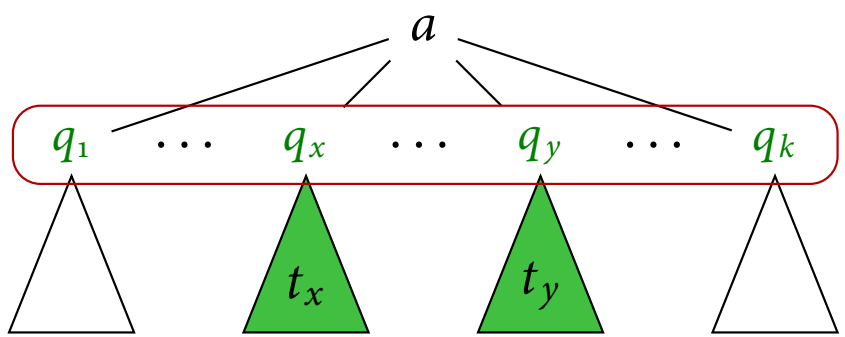
Idea: separate addressing and subtree comparison
$\rightsquigarrow$ use formula only to address pairs of positions to be compared

Atomic constraint: $(\ \varphi(x, y)\ ,\ type\ )$ $\varphi(x, y)$: MSO-formula with two free variables

Constraint types: $\exists_{EQ}$, $\forall_{EQ}$, $\exists_{NEQ}$, $\forall_{NEQ}$

Semantics: $w = q_1 \dots q_k \in Q^*$ and tree sequence $t_1 \dots t_k$ satisfy $(\varphi, type)$ if:

- $\exists_{EQ}$: there exist positions x, y in w with $w \vDash \varphi(x, y)$ and $t_x = t_y$
- $\exists_{NEQ}$: similar with $\neq$ instead of $=$



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

and $t_x \neq t_y$

Constraints among Unboundedly Many Siblings – cont'd

Idea: separate addressing and subtree comparison

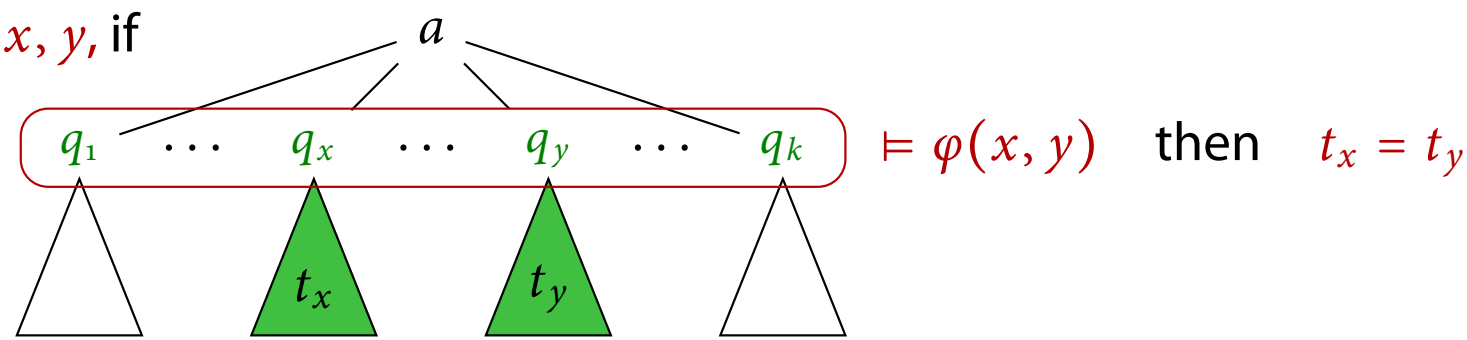
$\rightsquigarrow$ use formula only to address pairs of positions to be compared

Atomic constraint: $(\varphi(x, y), type)$ $\varphi(x, y)$: MSO-formula with two free variables

Constraint types: $\exists_{EQ}$, $\forall_{EQ}$, $\exists_{NEQ}$, $\forall_{NEQ}$

Semantics: $w = q_1 \dots q_k \in Q^*$ and tree sequence $t_1 \dots t_k$ satisfy $(\varphi, type)$ if:

- $\exists_{EQ}$: there exist positions x, y in w with $w \vDash \varphi(x, y)$ and $t_x = t_y$
- $\exists_{NEQ}$: similar with $\neq$ instead of $=$
- $\forall_{EQ}$: for all positions x, y in w, if $w \vDash \varphi(x, y)$, then $t_x = t_y$



Constraints among Unboundedly Many Siblings – cont'd

Idea: separate addressing and subtree comparison
$\rightsquigarrow$ use formula only to address pairs of positions to be compared

Atomic constraint: $(\ \varphi(x, y)\ ,\ type\ )$

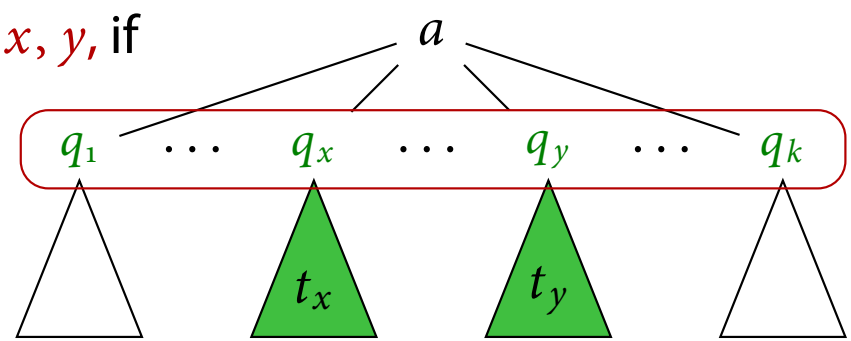
$\varphi(x, y)$: MSO-formula with two free variables

Constraint types: $\exists_{EQ}$, $\forall_{EQ}$, $\exists_{NEQ}$, $\forall_{NEQ}$

Semantics: $w = q_1 \dots q_k \in Q^*$ and tree sequence $t_1 \dots t_k$ satisfy $(\varphi, type)$ if:

- $\exists_{EQ}$: there exist positions x, y in w with $w \vDash \varphi(x, y)$ and $t_x = t_y$
- $\exists_{NEQ}$: similar with $\neq$ instead of $=$
- $\forall_{EQ}$: for all positions x, y in w, if $w \vDash \varphi(x, y)$, then $t_x = t_y$
- $\forall_{NEQ}$: similar with $\neq$ instead of $=$

for all x, y, if

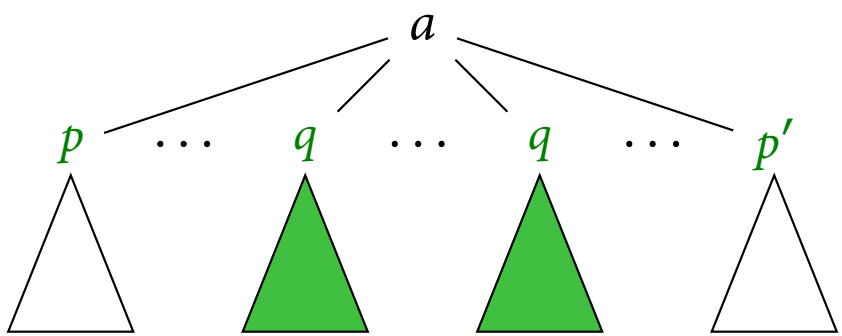


$\vDash \varphi(x, y)$ then $t_x \neq t_y$

Constraints among Unboundedly Many Siblings – Examples

Constraints: Boolean combinations of atomic constraints $(\varphi, type)$

Example: "There are at least two positions labeled with q, and all positions labeled q have equal subtrees, and all other positions have distinct subtrees"



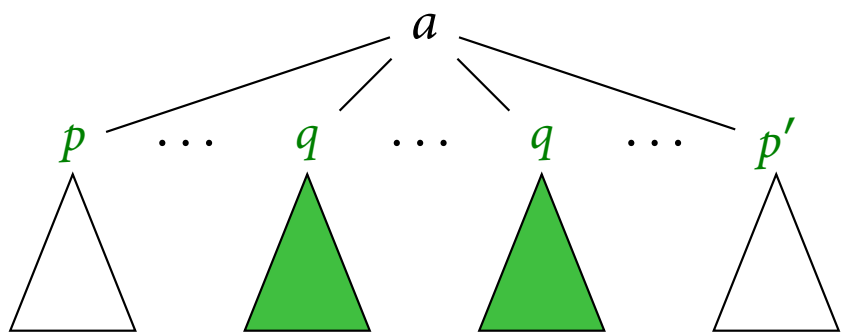
Express this by conjunction $(\varphi, \exists_{\mathsf{EQ}}) \wedge (\psi, \forall_{\mathsf{EQ}}) \wedge (\theta, \forall_{\mathsf{NEQ}})$ where:

$$\begin{aligned} \varphi(x,y) &:= \quad x < y \wedge q(x) \wedge q(y) \\ \psi(x,y) &:= \quad q(x) \wedge q(y) \\ \theta(x,y) &:= \quad q(x) \wedge \neg q(y) \end{aligned}$$

Constraints among Unboundedly Many Siblings – Examples

Constraints: Boolean combinations of atomic constraints $(\varphi, type)$

Example: "There are at least two positions labeled with q, and all positions labeled q have equal subtrees, and all other positions have distinct subtrees"



Express this by conjunction $(\varphi, \exists_{\mathsf{EQ}}) \wedge (\psi, \forall_{\mathsf{EQ}}) \wedge (\theta, \forall_{\mathsf{NEQ}})$ where:

$$\begin{aligned} \varphi(x,y) &:= \quad x < y \wedge q(x) \wedge q(y) \\ \psi(x,y) &:= \quad q(x) \wedge q(y) \\ \theta(x,y) &:= \quad q(x) \wedge \neg q(y) \end{aligned}$$

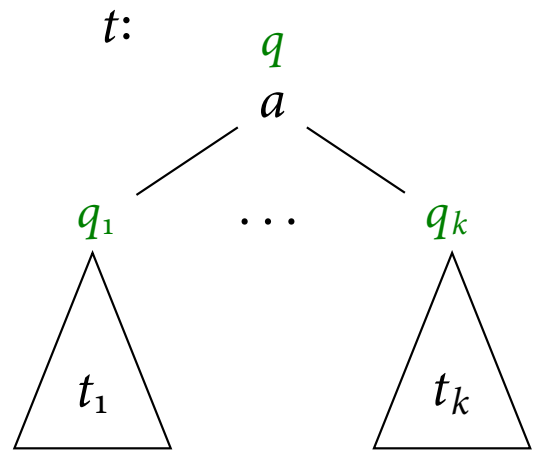
Remark:

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

UTACS

Unranked Tree Automaton with Constraints between Siblings $\mathcal{A} = (Q, \Sigma, \Delta, F)$:

- finite, unranked alphabet Σ
- state set Q and set F of accepting states
- run: bottom-up assignment of states to nodes
- transition (L, α, a, q) with
  - $L \subseteq Q^*$ regular
  - α is Boolean combination of atomic sibling constraints $(\varphi, type)$



Some results from the ranked case directly transferable, e.g.:

- closure under union and intersection

Determinism vs. Nondeterminism

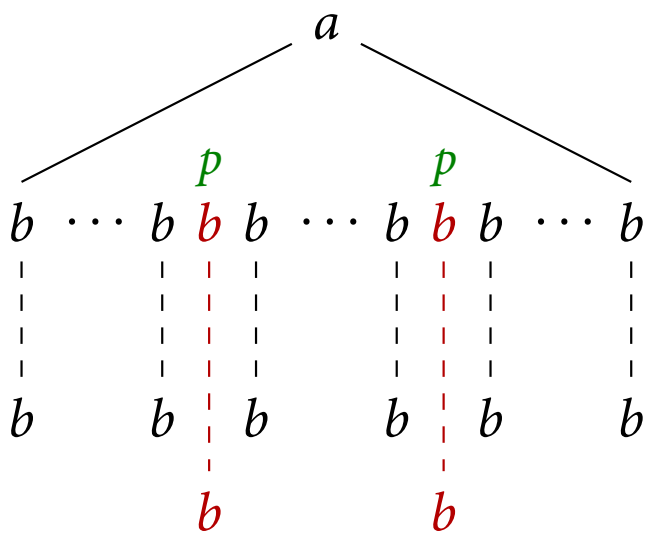
UTACS $\mathcal{A} = (Q, \Sigma, \Delta, F)$ is called deterministic if for each Σ-tree t there is at most one state q with $t \to_{\mathcal{A}} q$.

Proposition

Deterministic UTACS $\subsetneq$ Nondet. UTACS

Separating language contains trees of this form:

- root labeled with a
- root's children are strands of b's
- all but two b-strands are equal, and the latter two are equal



Nondeterministic UTACS: guess these two positions and mark them with a special state, and use this state in the constraints
$\rightsquigarrow$ not possible with deterministic UTACS since length of b-strands is unbounded

Determinism vs. Nondeterminism

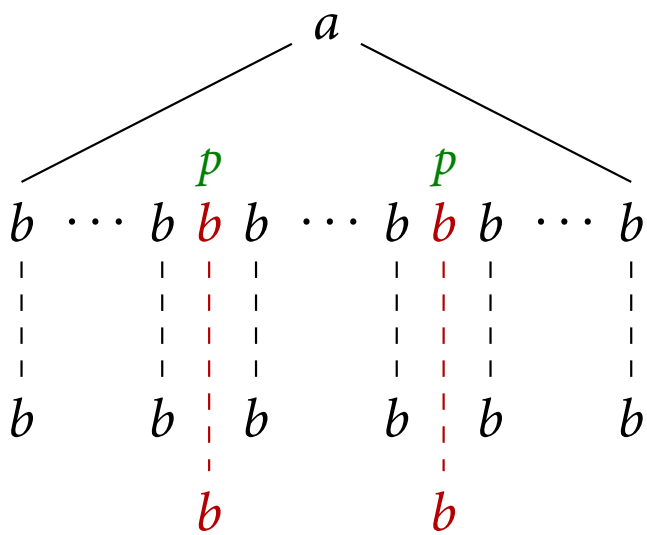
UTACS $\mathcal{A} = (Q, \Sigma, \Delta, F)$ is called deterministic if for each Σ-tree t there is at most one state q with $t \to_{\mathcal{A}} q$.

Proposition

Deterministic UTACS $\subsetneq$ Nondet. UTACS

Separating language contains trees of this form:

- root labeled with a
- root's children are strands of b's
- all but two b-strands are equal, and the latter two are equal



Nondeterministic UTACS: guess these two positions and mark them with a special state, and use this state in the constraints
$\rightsquigarrow$ not possible with deterministic UTACS since length of b-strands is unbounded

Remark: Determinization possible if atomic constraints do not make use of Q (via subset construction)

Outline

Unranked Tree Automata with Sibling (Dis)Equalities

Decidability of Emptiness: the Deterministic Case

Deciding Emptiness

Generic emptiness algorithm for bottom-up tree automaton:

- Check whether some final state q is reachable, i.e., there is at least one tree t with $t \to_{\mathcal{A}} q$.
- Procedure:
  1. Keep track of reachable states and the trees evaluating to them.
  2. For each transition, check if it is applicable using trees that are currently available.
  3. If so, mark the state reached by the transition and keep the constructed tree.

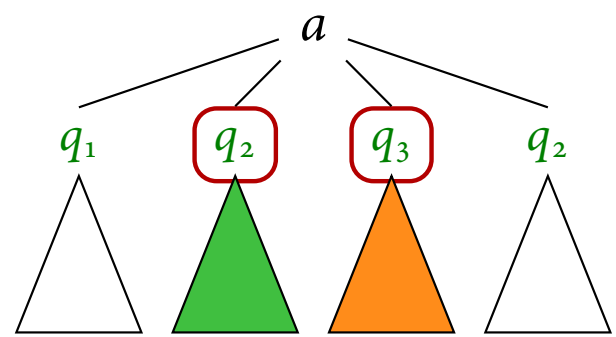
Deciding Emptiness

Generic emptiness algorithm for bottom-up tree automaton:

- Check whether some final state q is reachable, i.e., there is at least one tree t with $t \to_{\mathcal{A}} q$.
- Procedure:
  1. Keep track of reachable states and the trees evaluating to them.
  2. For each transition, check if it is applicable using trees that are currently available.
  3. If so, mark the state reached by the transition and keep the constructed tree.

But more work is needed to handle (dis)equality constraints:

- If $t_2 \neq t_3$ is required:
  $\rightsquigarrow$ trees must be distinct whenever the states are
  $\rightsquigarrow$ this holds with determinism



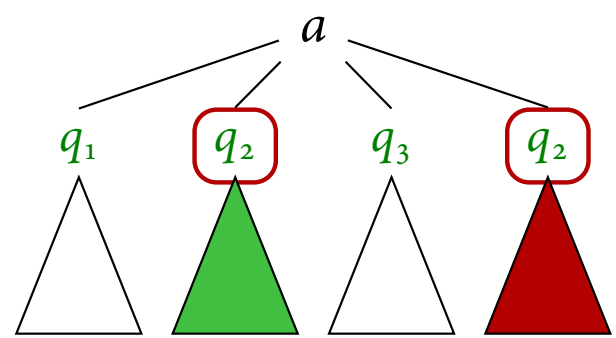
Deciding Emptiness

Generic emptiness algorithm for bottom-up tree automaton:

- Check whether some final state q is reachable, i.e., there is at least one tree t with $t \to_{\mathcal{A}} q$.
- Procedure:
  1. Keep track of reachable states and the trees evaluating to them.
  2. For each transition, check if it is applicable using trees that are currently available.
  3. If so, mark the state reached by the transition and keep the constructed tree.

But more work is needed to handle (dis)equality constraints:

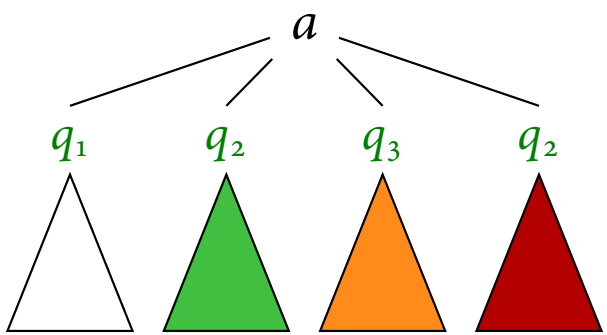
- If $t_2 \neq t_3$ is required:
  $\rightsquigarrow$ trees must be distinct whenever the states are
  $\rightsquigarrow$ this holds with determinism



- If "$t_2 \neq t_4$" is required:
  $\rightsquigarrow$ transition can only be applied if there are at least two trees evaluating to q_2.
  $\rightsquigarrow$ for each state, a certain number of trees evaluating to it are needed

… But How Many Trees to Collect?

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



Unranked case: …

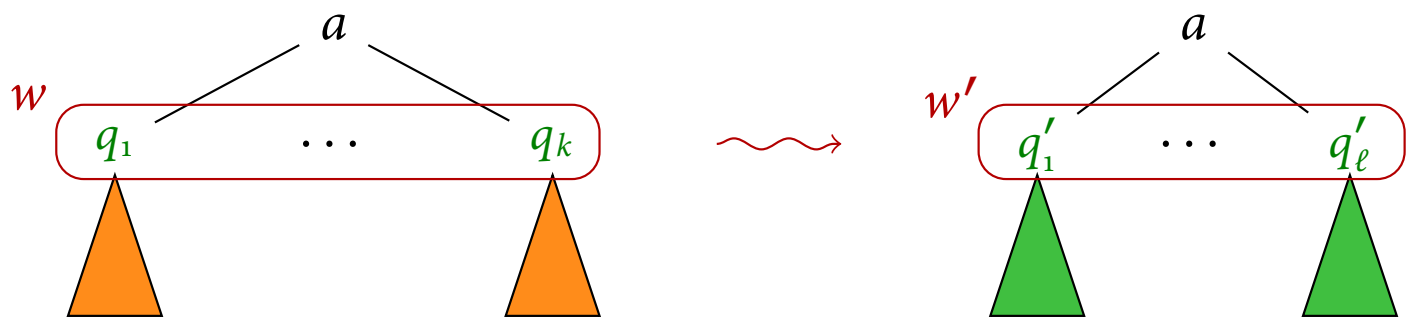
… But How Many Trees to Collect?

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

Unranked case: …

Lemma (Bound Lemma)

There exists a bound N such that: for each application of a transition $\tau = (L, \alpha, a, q)$ using $w = q_1 \dots q_k$, there is a replacement $w' = q'_1 \dots q'_\ell$ such that the application of τ using w' needs $\leq N$ distinct trees for each state.



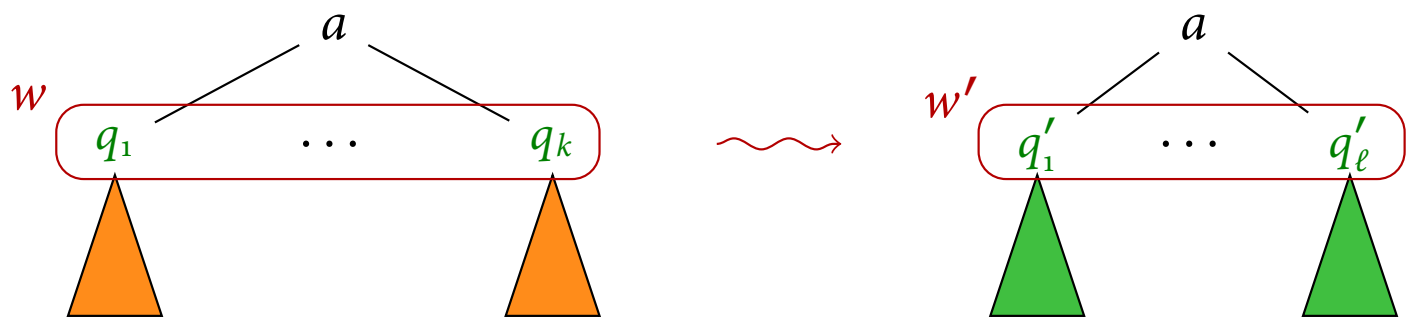
… But How Many Trees to Collect?

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

Unranked case: …

Lemma (Bound Lemma)

There exists a bound N such that: for each application of a transition $\tau = (L, \alpha, a, q)$ using $w = q_1 \dots q_k$, there is a replacement $w' = q'_1 \dots q'_\ell$ such that the application of τ using w' needs $\leq N$ distinct trees for each state.



Thus, emptiness algorithm needs to collect $\leq N$ distinct trees for each state.

Theorem: *The emptiness problem for deterministic UTACS is decidable.*

Conclusions

- Extension of tree automata with subtree (dis)equalities to unranked setting
- Use of MSO formulas as constraint addressing mechanism
- Decidability of emptiness for deterministic automata

$\rightsquigarrow$ First step to studying comparisons between data with tree representation

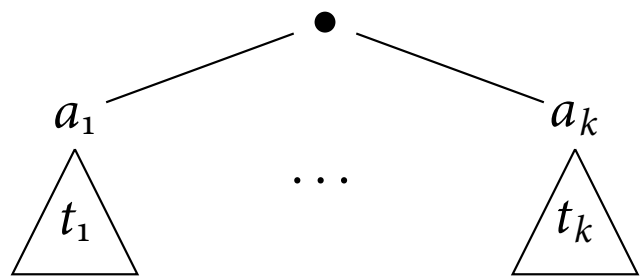
Conclusions

- Extension of tree automata with subtree (dis)equalities to unranked setting
- Use of MSO formulas as constraint addressing mechanism
- Decidability of emptiness for deterministic automata

$\rightsquigarrow$ First step to studying comparisons between data with tree representation

Next: compare the outputs of a preprocessing instead of the subtrees themselves
E.g., when considering representation of data words as trees, ignore the actual roots of the subtrees, i.e. compare t_i's instead of $a_i(t_i)$'s.

$$(a_1, i_1) \dots (a_k, i_k) \rightsquigarrow$$



$\rightsquigarrow$ incorporate transducer into our automaton model

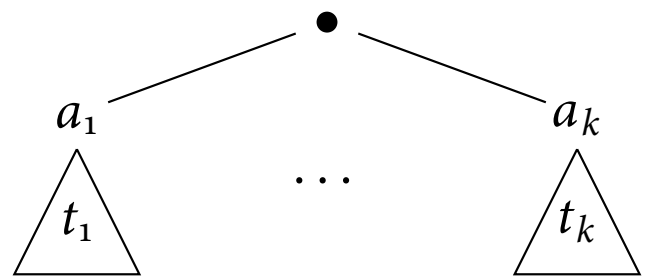
Conclusions

- Extension of tree automata with subtree (dis)equalities to unranked setting
- Use of MSO formulas as constraint addressing mechanism
- Decidability of emptiness for deterministic automata

$\rightsquigarrow$ First step to studying comparisons between data with tree representation

Next: compare the outputs of a preprocessing instead of the subtrees themselves
E.g., when considering representation of data words as trees, ignore the actual roots of the subtrees, i.e. compare t_i's instead of $a_i(t_i)$'s.

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



$\rightsquigarrow$ incorporate transducer into our automaton model

Further prospects:

- nondeterministic UTACS
- relation to automata on data words