# Finite Automata on Unranked Trees: Extensions by Arithmetical and Equality Constraints

Karianto Wong

RWTH Aachen University

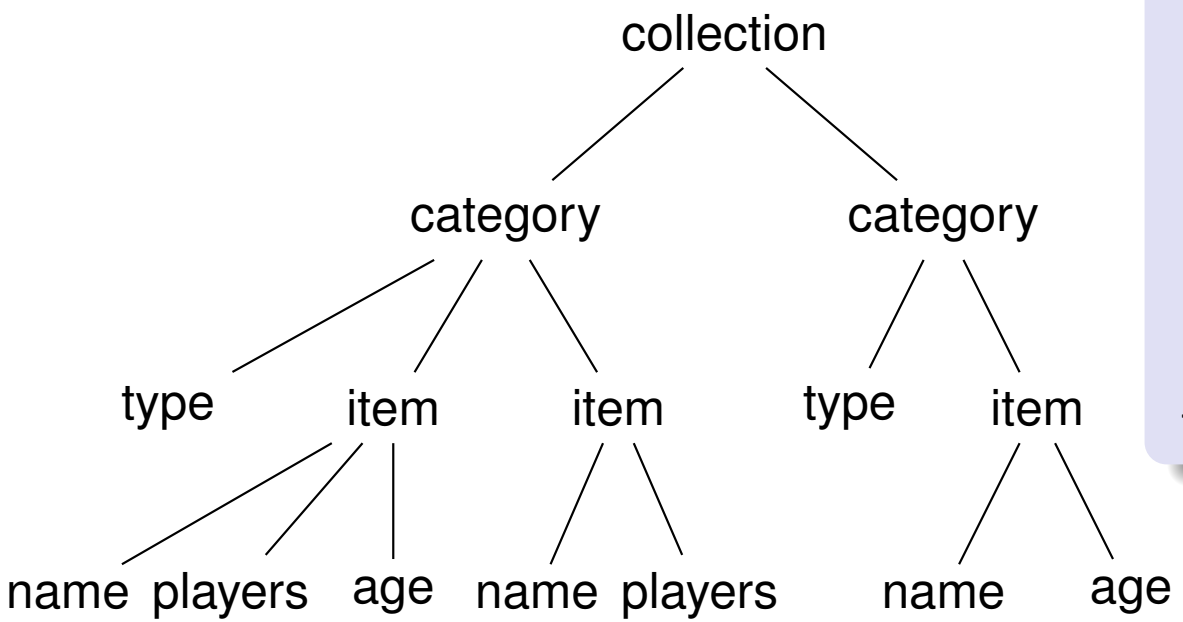
Oberseminar

# Motivation and Context

Unranked trees:

- finite, ordered trees
- number of children of a node is not bounded a priori (i.e., does not depend on the node's label)
- formal models for XML documents

Example: a collection of games




```
<collection>
  <category>
    <name>Board Games</name>
    <item>
      <name>Go</name>
      <players>2</players>
      <age>9–99</age>
    </item>
      <name>Chess</name>
      <players>2</players>
    </item>
  </category>
  <category>
    <name>Puzzles</name>
    <item>
      <name>Rubik's Cube</name>
      <age>8–12</name>
    </item>
  </category>
</collection>
```

# Motivation and Context (2)

XML-related tasks and problems (see, e.g., survey [Schwentick, 2007]);

- specification (“define languages of unranked trees”)
- validation (“does a tree satisfy a specification?”)
- satisfiability (“is a specification satisfiable?”)
- query (“select some nodes or some subtrees of a tree”)
- . . .

Automata-theoretical approach, e.g.:

- (parallel) tree automata
- tree-walking automata
- Document-Type Definitions
  (i.e., extended context-free grammars)

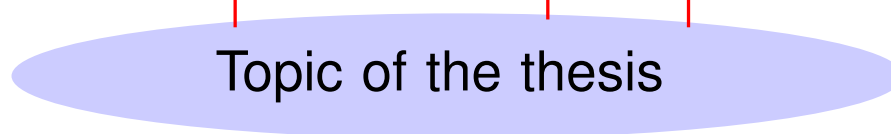
# Motivation and Context (2)

XML-related tasks and problems (see, e.g., survey [Schwentick, 2007]);

- specification (“define languages of unranked trees”)
- validation (“does a tree satisfy a specification?”)
- satisfiability (“is a specification satisfiable?”)
- query (“select some nodes or some subtrees of a tree”)
- . . .

Automata-theoretical approach, e.g.:

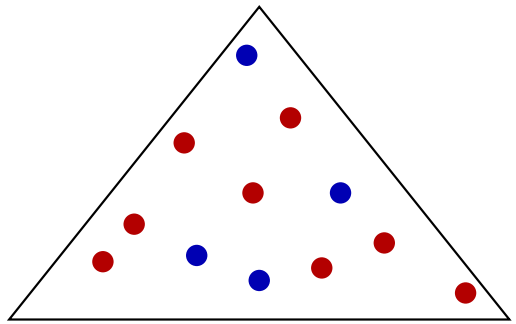
- (parallel) tree automata
- tree-walking automata
- Document-Type Definitions
  (i.e., extended context-free grammars)



# Scope and Contributions of the Thesis

Extensions of finite, bottom-up unranked tree automata:

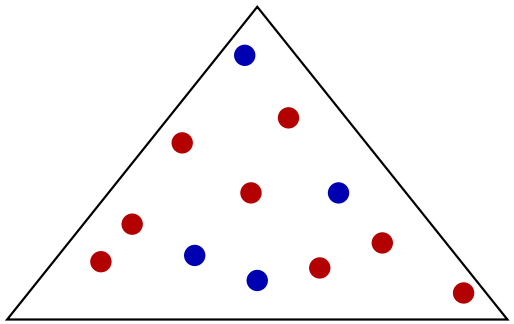
1. Arithmetical (i.e., counting) properties
   Example: “There are twice as many red nodes as blue nodes”



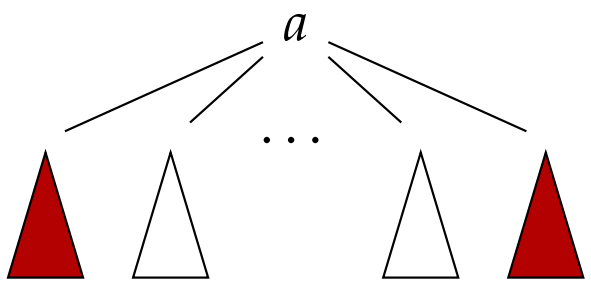
# Scope and Contributions of the Thesis

Extensions of finite, bottom-up unranked tree automata:

1. Arithmetical (i.e., counting) properties
   Example: “There are twice as many red nodes as blue nodes”



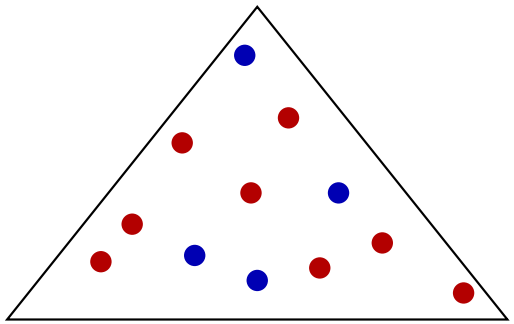
2. Equality between subtrees
   Example: “The first subtree of the root is equal to the last one”



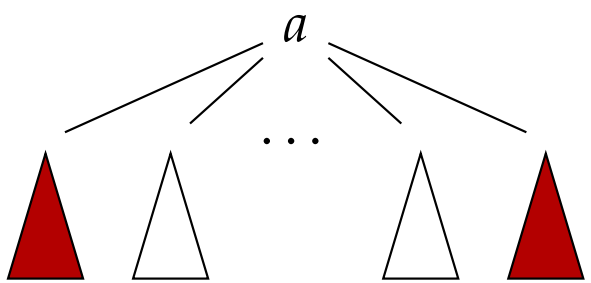
# Scope and Contributions of the Thesis

Extensions of finite, bottom-up unranked tree automata:

1. Arithmetical (i.e., counting) properties
   Example: “There are twice as many red nodes as blue nodes”



2. Equality between subtrees
   Example: “The first subtree of the root is equal to the last one”




For both extensions, study the following:

- expressive power
- decidability of emptiness (i.e. satisfiability)

# Outline

1. Automata with arithmetical constraints
   - Parikh unranked tree automata
   - Emptiness problem

2. Automata with subtree-equality constraints
   - Equality constraints between siblings
   - Emptiness problem

3. Conclusion

# Outline

1. Automata with arithmetical constraints
   - Parikh unranked tree automata
   - Emptiness problem
2. Automata with subtree-equality constraints
   - Equality constraints between siblings
   - Emptiness problem
3. Conclusion

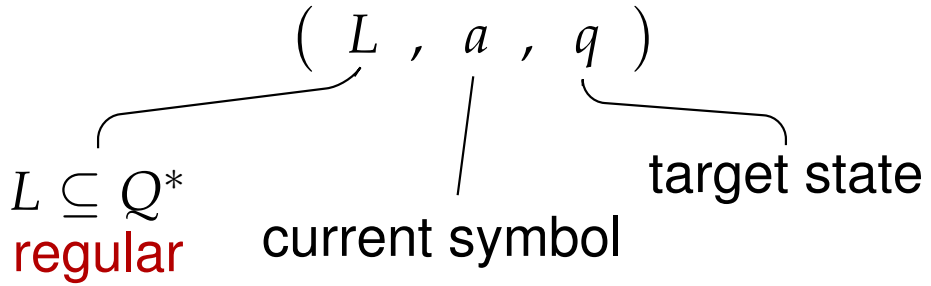
# Outline

1. Automata with arithmetical constraints
   - Parikh unranked tree automata
   - Emptiness problem
2. Automata with subtree-equality constraints
   - Equality constraints between siblings
   - Emptiness problem
3. Conclusion

# Finite Automata on Unranked Trees

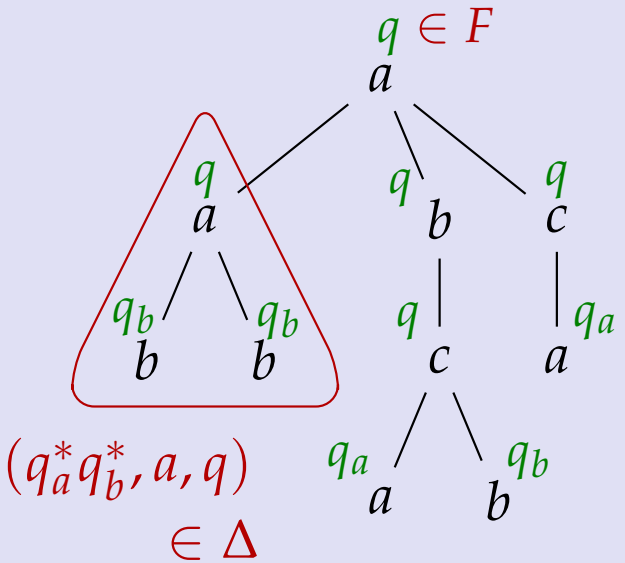
Unranked tree automaton $\mathcal{A} = (Q, \Sigma, \Delta, F)$

- Bottom-up tree automaton
- Finite state set $Q$ with final states $F \subseteq Q$
- Symbols in $\Sigma$ have no fixed arities
- Transition relation $\Delta$ with transitions




- Run is assignment of states to nodes complying with $\Delta$
- Tree is accepted if there exists some run with state in $F$ assigned to the root.

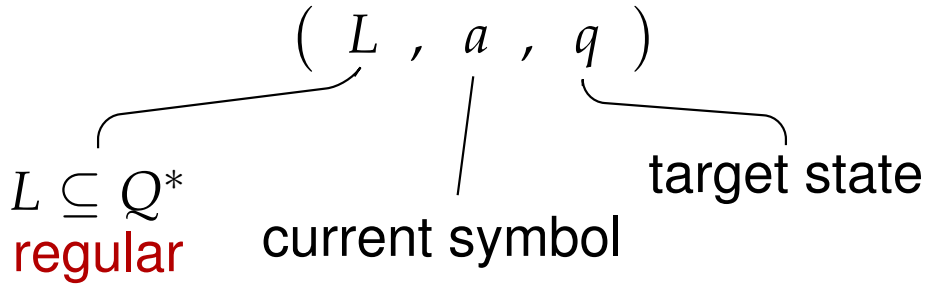
$\Sigma = \{a, b, c\}$
$\mathcal{T}(\mathcal{A})$ = set of trees over $\Sigma$ with leaf sequences of the form $a^*b^*$



# Finite Automata on Unranked Trees

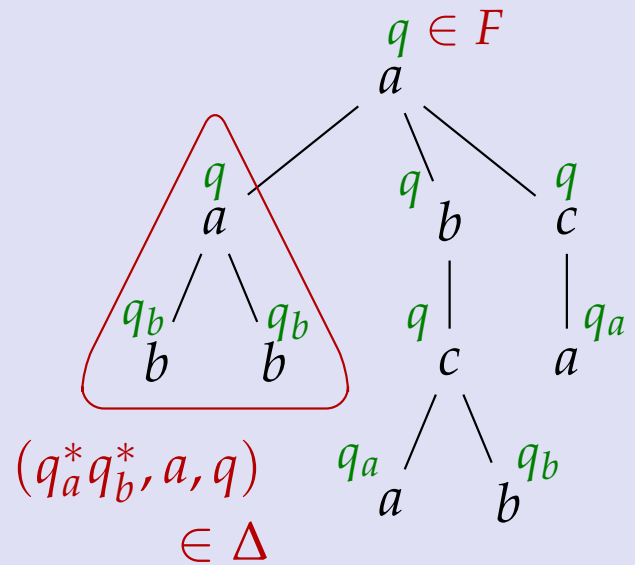
Unranked tree automaton $\mathcal{A} = (Q, \Sigma, \Delta, F)$

- Bottom-up tree automaton
- Finite state set $Q$ with final states $F \subseteq Q$
- Symbols in $\Sigma$ have no fixed arities
- Transition relation $\Delta$ with transitions




- Run is assignment of states to nodes complying with $\Delta$
- Tree is accepted if there exists some run with state in $F$ assigned to the root.

$\Sigma = \{a, b, c\}$
$\mathcal{T}(\mathcal{A})$ = set of trees over $\Sigma$ with leaf sequences of the form $a^*b^*$




Extensions: how to incorporate arithmetical properties?

Example: "There are as many $a$-labeled nodes as $b$-labeled ones"

# Parikh Automata

Introduced in [Klaedtke/Rueß, 2002] for words and ranked trees

Main idea:

- Run is assignment of states and vectors of natural numbers to nodes
- Acceptance is based on existence of run satisfying:
  1. the root is assigned a final state (and a final vector)
  2. the sum of all occurring vectors satisfies a Presburger formula (first-order logic over $(\mathbb{N}, +, <)$ – satisfiability decidable)

# Parikh Automata

Introduced in [Klaedtke/Rueß, 2002] for words and ranked trees

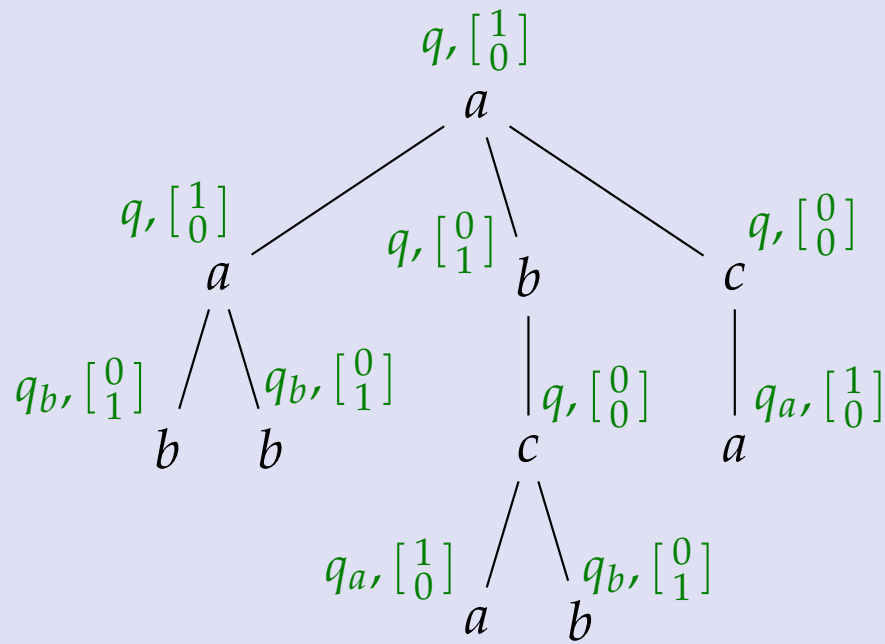
Main idea:

- Run is assignment of states and vectors of natural numbers to nodes
- Acceptance is based on existence of run satisfying:
  1. the root is assigned a final state (and a final vector)
  2. the sum of all occurring vectors satisfies a Presburger formula (first-order logic over $(\mathbb{N}, +, <)$ – satisfiability decidable)

Example property: "there are as many $a$-nodes as $b$-nodes"

- use vectors of dimension two
- assign $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ to $a$-nodes
- assign $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ to $b$-nodes
- use Presburger formula

$$\psi(x_1, x_2) := (x_1 = x_2)$$



# Local Arithmetical Constraints

Remark: The example property is a global one, i.e., it is posed to (run) tree as a whole.
[Example: “There are as many $a$-nodes as $b$-nodes (in the whole tree)”]

How about local properties (i.e. concerning only the children of a node)?
Example: “Each node has at least as many $a$-labeled children as $b$-labeled ones”

# Local Arithmetical Constraints

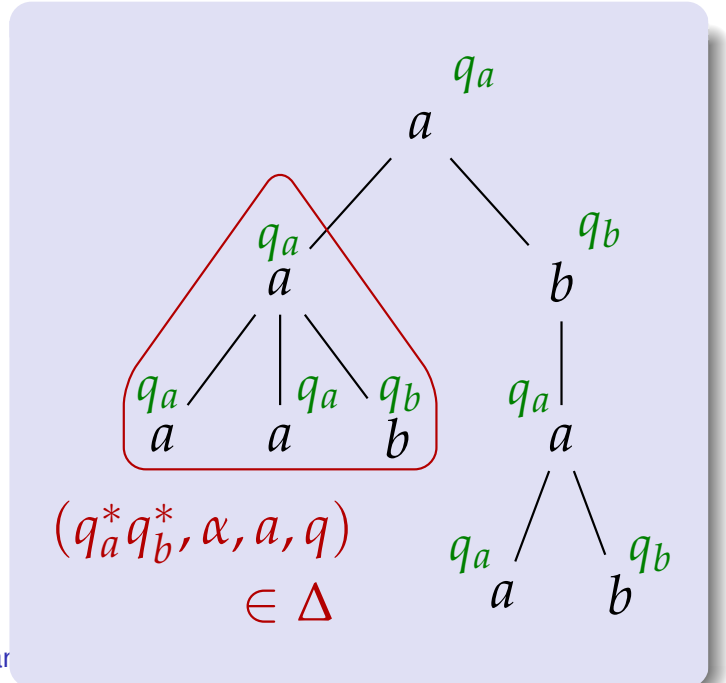
Remark: The example property is a global one, i.e., it is posed to (run) tree as a whole.
[Example: "There are as many $a$-nodes as $b$-nodes (in the whole tree)"]

How about local properties (i.e. concerning only the children of a node)?
Example: "Each node has at least as many $a$-labeled children as $b$-labeled ones"

Idea: Constrain application of transitions by Presburger formulas
[Seidl/Schwentick/Muscholl, Dal Zilio/Lugiez, 2003]

- transition: $(L, \alpha, a, q)$
- $\alpha$: Presburger formula with free variables interpreted as number of occurrences of states in the child nodes

Example: $\alpha(x_{q_a}, x_{q_b}) = (x_{q_a} \geq x_{q_b})$



# Local Arithmetical Constraints

Remark: The example property is a global one, i.e., it is posed to (run) tree as a whole.
[Example: "There are as many $a$-nodes as $b$-nodes (in the whole tree)"]

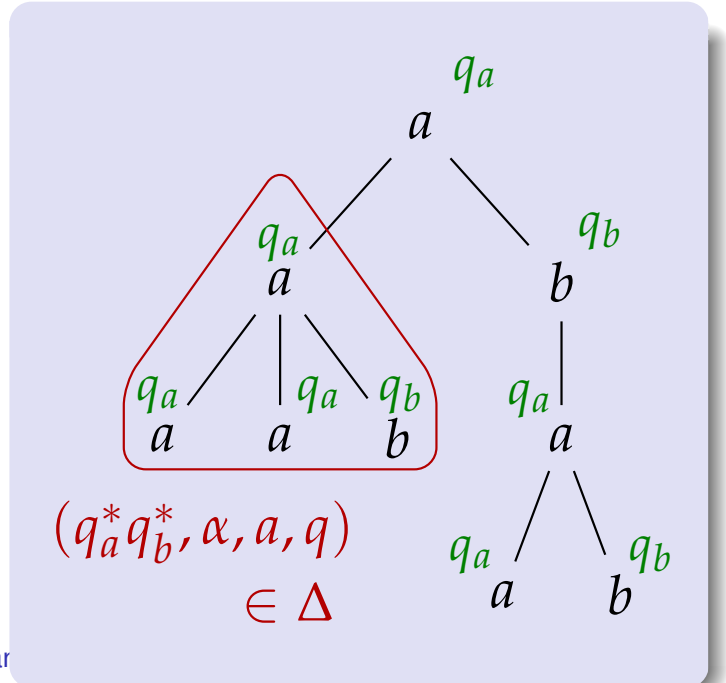
How about local properties (i.e. concerning only the children of a node)?
Example: "Each node has at least as many $a$-labeled children as $b$-labeled ones"

Idea: Constrain application of transitions by Presburger formulas
[Seidl/Schwentick/Muscholl, Dal Zilio/Lugiez, 2003]

- transition: $(L, \alpha, a, q)$
- $\alpha$: Presburger formula with free variables interpreted as number of occurrences of states in the child nodes

Example: $\alpha(x_{q_a}, x_{q_b}) = (x_{q_a} \geq x_{q_b})$

Next: combine both types of constraints



# Parikh Unranked Tree Automata

Parikh unranked tree automaton $\mathcal{A} = (Q, D, \Sigma, \Delta, F, \psi)$

- Finite state set $Q$, unranked alphabet $\Sigma$
- Finite auxiliary set $D \subseteq \mathbb{N}^k$ of vectors of dimension $k \geq 1$
- Transition relation $\Delta$ with transitions

$$( \; L \; , \; \alpha \; , \; a \; , \; (q, \bar{d}) \; )$$

$L \subseteq (Q \times D)^*$ regular

$\alpha$ Presburger formula with $k$ free variables

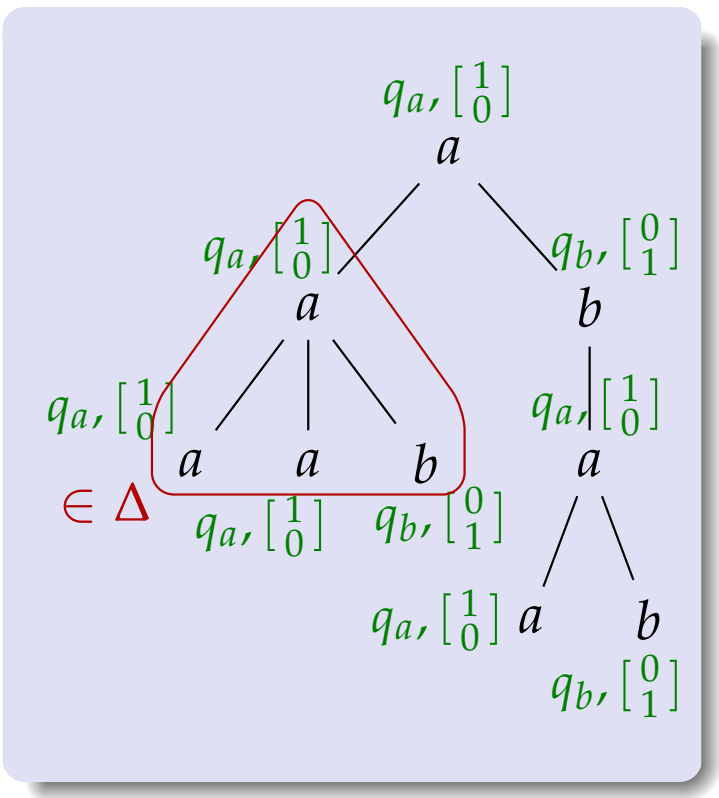
$a$ current symbol

$(q, \bar{d}) \in Q \times D$ target pair

- Set of final pairs $F \subseteq Q \times D$
- Presburger formula $\psi$ with $k$ free variables

Runs: comply with (constrained) transitions

Accepting runs: $F$ at the root and the sum of all occurring vectors satisfies $\psi$



# Outline

1. Automata with arithmetical constraints
   - Parikh unranked tree automata
   - Emptiness problem
2. Automata with subtree-equality constraints
   - Equality constraints between siblings
   - Emptiness problem
3. Conclusion

# Emptiness Problem

**Theorem.** *Given a Parikh-UTA $\mathcal{A} = (Q, D, \Sigma, \Delta, F, \psi)$, it is decidable whether $\mathcal{T}(\mathcal{A})$ is empty.*

Proof combines techniques used by [Klaedtke/Rueß, 2003] and by [Seidl/Schwentick/Muscholl, 2003].

Main ingredients:

- decidability of satisfiability for Presburger logic [Presburger, 1930]
- Parikh's Theorem [Parikh, 1966]

# Emptiness Problem

**Theorem.** *Given a Parikh-UTA $\mathcal{A} = (Q, D, \Sigma, \Delta, F, \psi)$, it is decidable whether $\mathcal{T}(\mathcal{A})$ is empty.*

Proof combines techniques used by [Klaedtke/Rueß, 2003] and by [Seidl/Schwentick/Muscholl, 2003].

Main ingredients:

- decidability of satisfiability for Presburger logic [Presburger, 1930]
- Parikh's Theorem [Parikh, 1966]

**Definition** (Presburger-definable sets)**.**

A Presburger formula $\psi(x_1, \ldots, x_k)$ defines a subset of $\mathbb{N}^k$:

$$[\![ \psi ]\!] = \{(d_1, \ldots, d_k) \in \mathbb{N}^k \mid (\mathbb{N}, +, <) \models \psi(d_1, \ldots, d_k)\}$$

**Theorem** [Presburger, 1930]**.** *Given a Presburger formula $\psi(x_1, \ldots, x_k)$, it is deciable whether $[\![ \psi ]\!] = \emptyset$.*

# Parikh's Theorem

Word language $L$ over alphabet $\Sigma = \{a_1, \ldots, a_k\}$ also defines a subset of $\mathbb{N}^k$ (the Parikh image of $L$):

$$\Phi(L) = \{(|w|_{a_1}, \ldots, |w|_{a_k}) \in \mathbb{N}^k \mid w \in L\}$$

**Theorem** [Parikh, 1966]. *If $L$ is a context-free language, then $\Phi(L)$ is definable by a Presburger formula.*

# Parikh's Theorem

Word language $L$ over alphabet $\Sigma = \{a_1, \ldots, a_k\}$ also defines a subset of $\mathbb{N}^k$ (the Parikh image of $L$):

$$\Phi(L) = \{(|w|_{a_1}, \ldots, |w|_{a_k}) \in \mathbb{N}^k \mid w \in L\}$$

**Theorem** [Parikh, 1966]. *If $L$ is a context-free language, then $\Phi(L)$ is definable by a Presburger formula.*

Let $Q$ be a finite set (of states) and $D \subseteq \mathbb{N}^k$. A language $R \subseteq (Q \times D)^*$ defines a subset of $\mathbb{N}^k$ (the extended Parikh image of $R$):

$$\widetilde{\Phi}(R) = \{\textstyle\sum_{i=1}^{m} \overline{d}_i \mid \langle q_1, \overline{d}_1 \rangle \ldots \langle q_m, \overline{d}_m \rangle \in R\}$$

**Lemma** [Klaedtke/Rueß, 2003]. *If $\Phi(R)$ is definable by a Presburger formula, then so is $\widetilde{\Phi}(R)$.*

# Deciding Emptiness of Parikh-UTA

**Theorem.** *Given a Parikh-UTA $\mathcal{A} = (Q, D, \Sigma, \Delta, F, \psi)$, it is decidable whether $\mathcal{T}(\mathcal{A})$ is empty.*

**Proof sketch.**

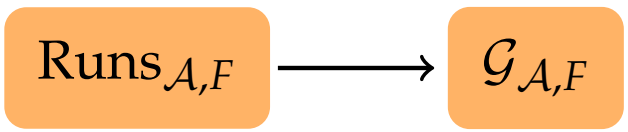
$\mathrm{Runs}_{\mathcal{A},F}$: set of runs (trees over $Q \times D$) such that the root's label is in $F$

# Deciding Emptiness of Parikh-UTA

**Theorem.** *Given a Parikh-UTA $\mathcal{A} = (Q, D, \Sigma, \Delta, F, \psi)$, it is decidable whether $\mathcal{T}(\mathcal{A})$ is empty.*

**Proof sketch.**

$\mathrm{Runs}_{\mathcal{A},F}$: set of runs (trees over $Q \times D$) such that the root's label is in $F$



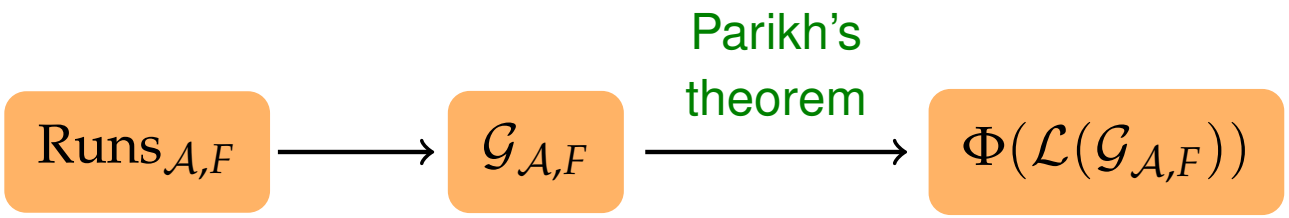

ext. context-free grammar for linearizations of runs

# Deciding Emptiness of Parikh-UTA

**Theorem.** *Given a Parikh-UTA $\mathcal{A} = (Q, D, \Sigma, \Delta, F, \psi)$, it is decidable whether $\mathcal{T}(\mathcal{A})$ is empty.*

**Proof sketch.**

$\mathrm{Runs}_{\mathcal{A},F}$: set of runs (trees over $Q \times D$) such that the root's label is in $F$




ext. context-free grammar for linearizations of runs

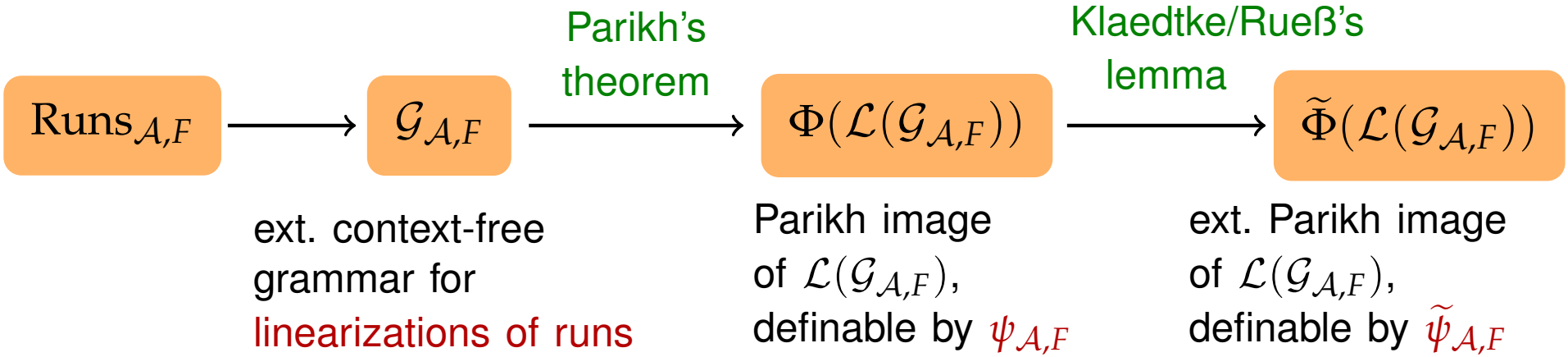
Parikh image of $\mathcal{L}(\mathcal{G}_{\mathcal{A},F})$, definable by $\psi_{\mathcal{A},F}$

# Deciding Emptiness of Parikh-UTA

**Theorem.** *Given a Parikh-UTA $\mathcal{A} = (Q, D, \Sigma, \Delta, F, \psi)$, it is decidable whether $\mathcal{T}(\mathcal{A})$ is empty.*

**Proof sketch.**

$\mathrm{Runs}_{\mathcal{A},F}$: set of runs (trees over $Q \times D$) such that the root's label is in $F$

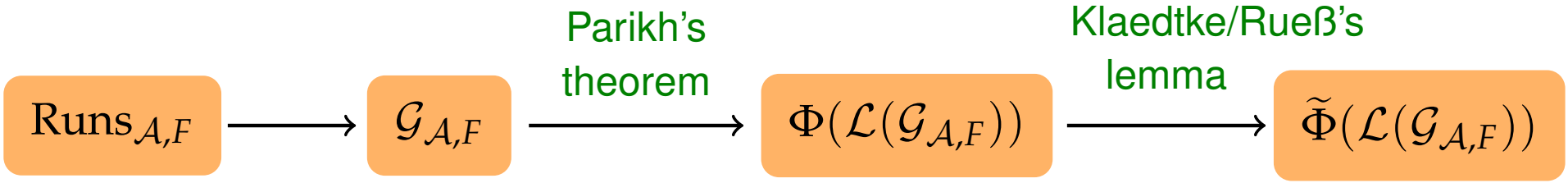


# Deciding Emptiness of Parikh-UTA

**Theorem.** *Given a Parikh-UTA $\mathcal{A} = (Q, D, \Sigma, \Delta, F, \psi)$, it is decidable whether $\mathcal{T}(\mathcal{A})$ is empty.*

**Proof sketch.**

$\mathrm{Runs}_{\mathcal{A},F}$: set of runs (trees over $Q \times D$) such that the root's label is in $F$




$$\mathcal{T}(\mathcal{A}) \neq \emptyset \iff \exists\ \rho \in \mathrm{Runs}_{\mathcal{A},F} \text{ such that sum of vectors in } \rho \text{ satisfies } \psi$$
$$\iff \exists\ w_\rho \in \mathcal{L}(\mathcal{G}_{\mathcal{A},F})) \text{ s.t. sum of vectors in } w_\rho \text{ satisfies } \psi$$
$$\iff \exists\ \overline{d}_\rho \in \widetilde{\Phi}(\mathcal{L}(\mathcal{G}_{\mathcal{A},F})) \text{ such that } \overline{d}_\rho \text{ satisfies } \psi$$
$$\iff [\![\ \widetilde{\psi}_{\mathcal{A},F} \wedge \psi\ ]\!] \neq \emptyset \qquad \rightsquigarrow \text{ decidable [Presburger]}$$

# Parikh Unranked Tree Automata – Summary

Automata with arithmetical constraints expressed in Presburger logic:

- Global constraints: posed to runs as a whole
- Local constraints: posed to application of transitions
- Closed under intersection and union, but not under complementation

Expressiveness of local and global constraints incomparable, i.e.:

- Some languages only definable by means of global constraints, e.g., "there are as many $a$-nodes as $b$-nodes"
- Some languages only definable by means of local constraints, e.g., "each node has at least as many $a$-labeled children as $b$-labeled ones"

Decision problems:

- Emptiness is decidable
- Universality is undecidable (holds already for Parikh word automata)

# Outline

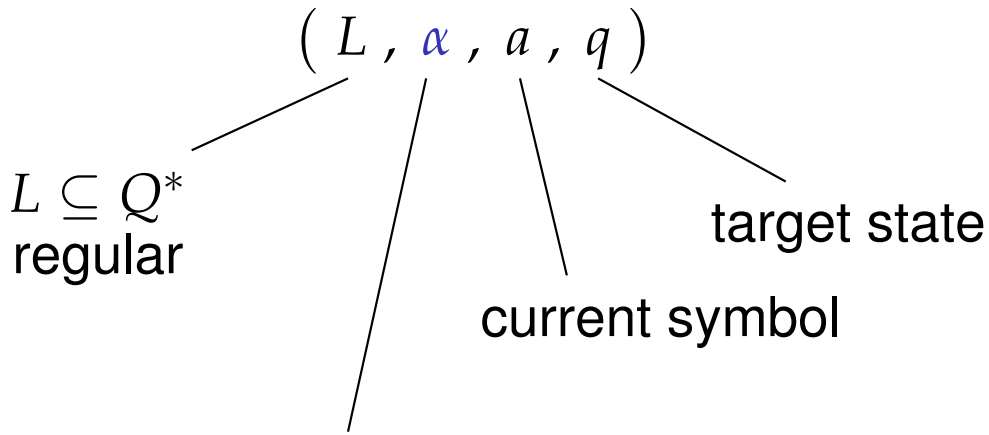
1. Automata with arithmetical constraints
   - Parikh unranked tree automata
   - Emptiness problem

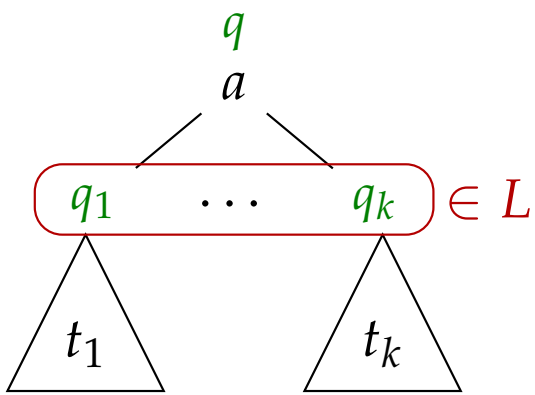
2. Automata with subtree-equality constraints
   - Equality constraints between siblings
   - Emptiness problem

3. Conclusion

# Automata with Subtree-Equality between Siblings

- unranked tree automaton $\mathcal{A} = (Q, \Sigma, \Delta, F)$
- Transitions:

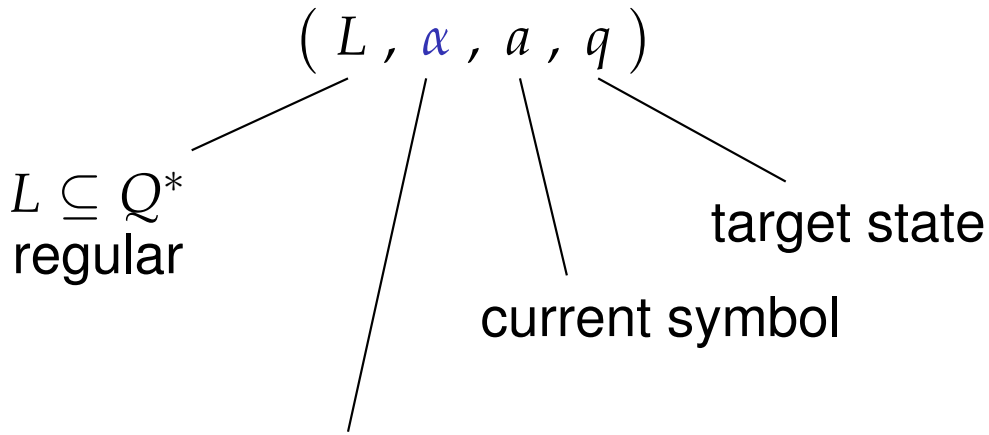


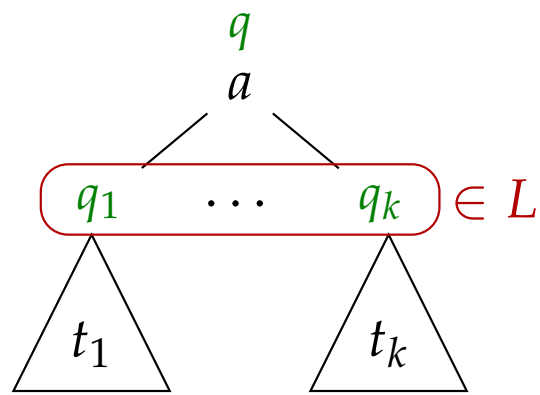




equality constraints between direct subtrees (siblings), e.g.:

# Automata with Subtree-Equality between Siblings

- unranked tree automaton $\mathcal{A} = (Q, \Sigma, \Delta, F)$
- Transitions:



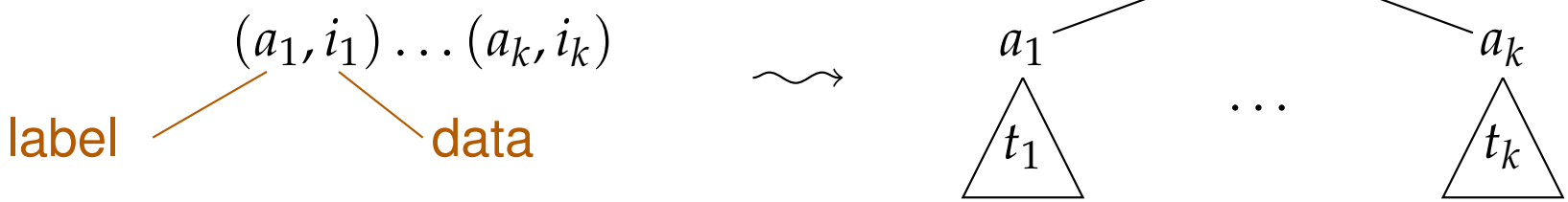

equality constraints between direct subtrees (siblings), e.g.:

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

Remark. In the ranked setting, emptiness is undecidable if equality constraints between arbitrary subtrees are allowed.

# Motivations

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

1. Automata with arithmetical constraints
   - Parikh unranked tree automata
   - Emptiness problem

2. Automata with subtree-equality constraints
   - Equality constraints between siblings
   - Emptiness problem

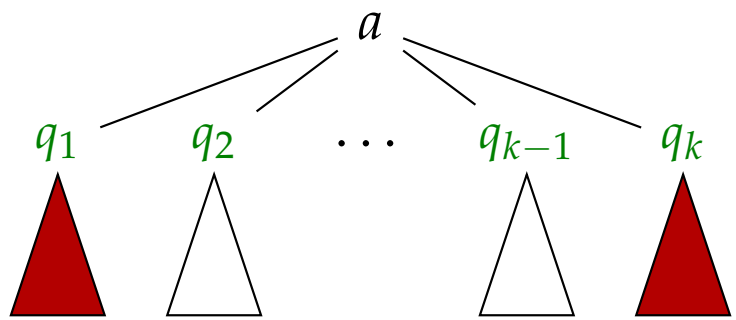
3. Conclusion

# 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"

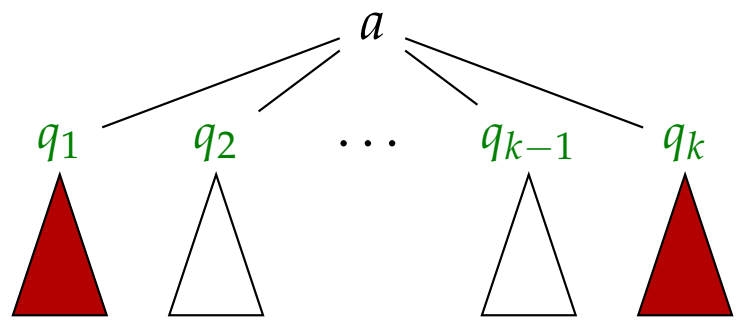


# Constraints among Unboundedly Many Siblings

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:

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

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

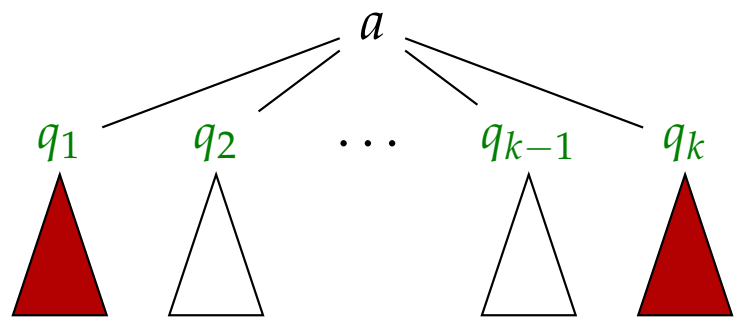
$$\exists x \exists y \,(\, x = \min \wedge y = \max \wedge t_x = t_y \wedge \forall z \,(\, z \neq x \wedge z \neq y \rightarrow t_z \neq t_x \,)\,)$$

# 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:

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

- 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

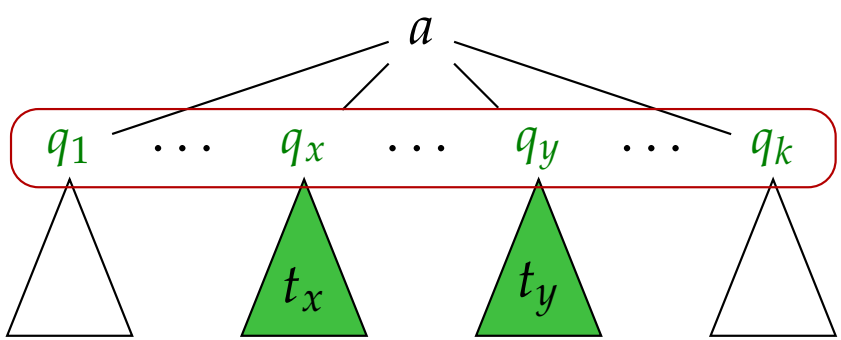
Idea: separate addressing and subtree comparison

$\rightsquigarrow$ 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

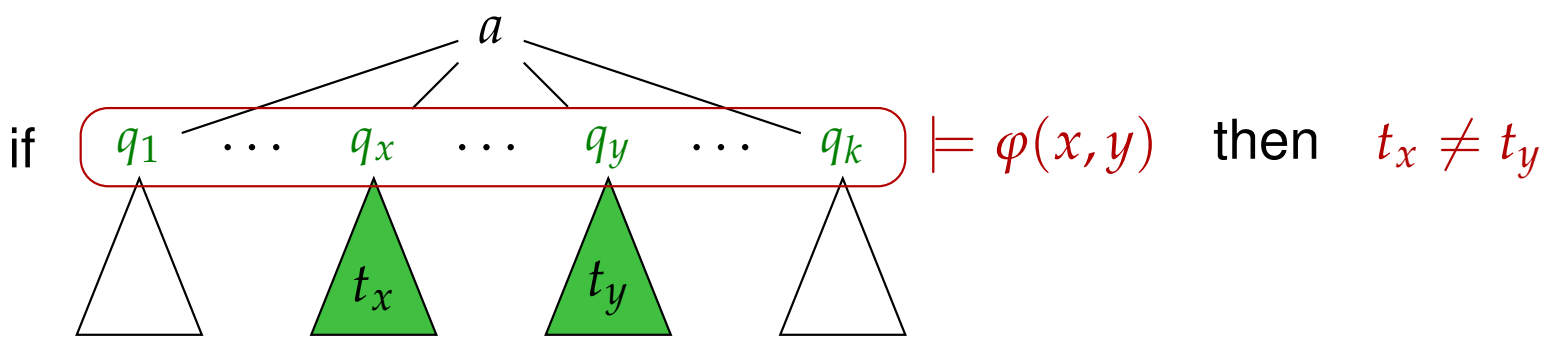
Idea: separate addressing and subtree comparison

$\rightsquigarrow$ 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 $x, y$



# Constraints among Unboundedly Many Siblings – cont'd

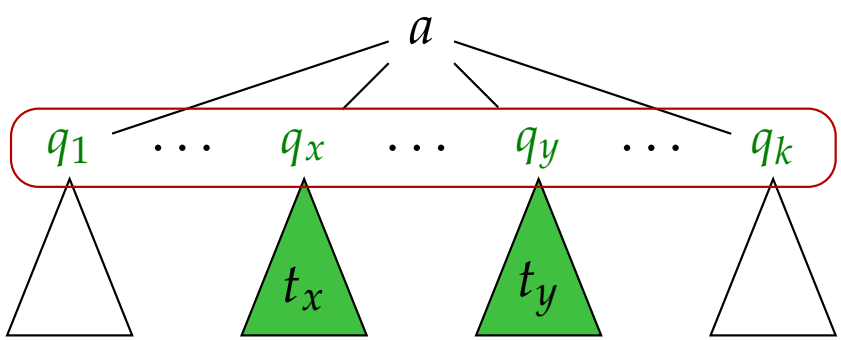
Idea: separate addressing and subtree comparison

$\rightsquigarrow$ 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 $x, y$




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

and $t_x = t_y$

# Constraints among Unboundedly Many Siblings – cont'd

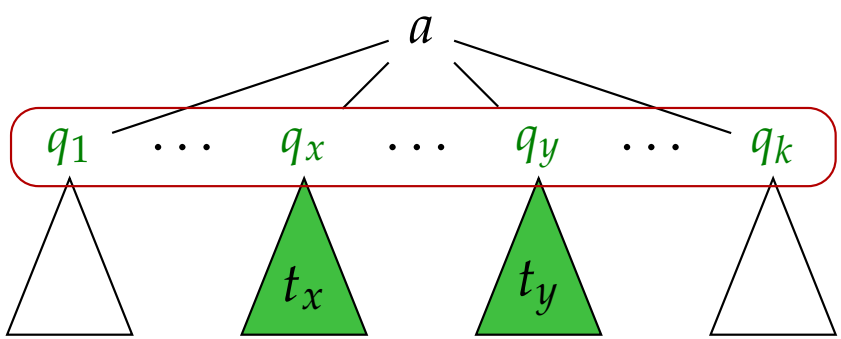
Idea: separate addressing and subtree comparison

$\rightsquigarrow$ use 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"

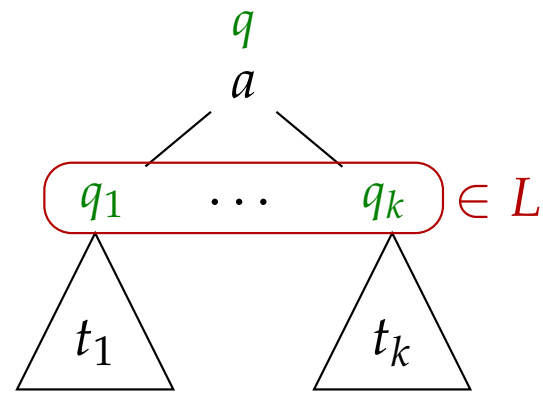
$$\big[\, \forall x \forall y \,.\, x = \min \wedge y = \max \;\rightarrow\; t_x = t_y \,\big] \wedge$$

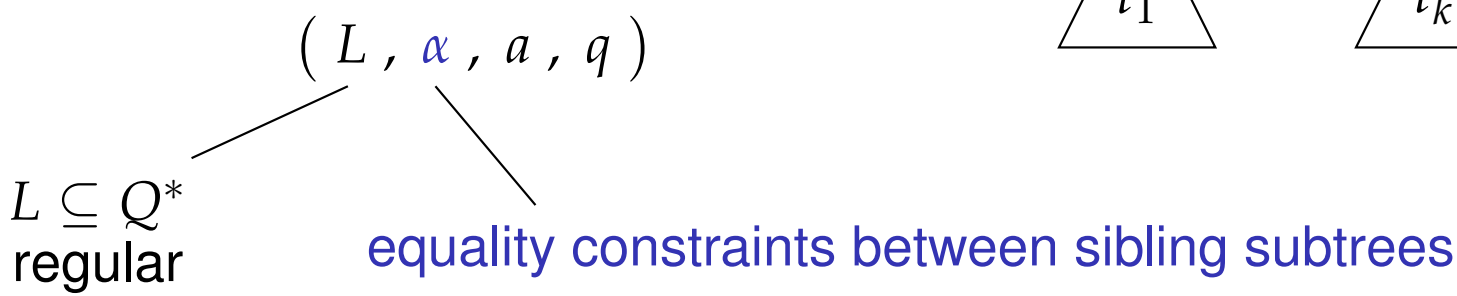
$$\big[\, \forall x \forall y \,.\, \big((x = \min \wedge x < y < \max) \vee (y = \max \wedge \min < x < y)\big) \rightarrow\; t_x \neq t_y \,\big]$$

# 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 $\Sigma$
- Transitions in $\Delta$:




Remarks:

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

# Outline

1. Automata with arithmetical constraints
   - Parikh unranked tree automata
   - Emptiness problem

2. Automata with subtree-equality constraints
   - Equality constraints between siblings
   - Emptiness problem

3. Conclusion

# Emptiness Decidability

**Theorem** [Löding/Wong, 2007 & 2009]**.**
*Emptiness for (nondeterministic) UTACSs is decidable.*

Proof for the deterministic case:

- Adapted from the ranked setting [Bogaert/Tison, 1992].
- Difficulty arises from the unrankedness aspect.

Proof for the nondeterministic case:

- The methods used are basically the same as in deterministic case ...
- ... but a lot more technicalities are required.

**Remark.** Nondeterministic UTACSs are more expressive than deterministic ones.

# 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:

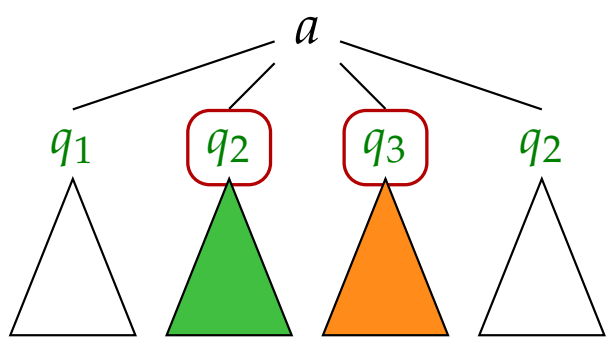
- Iteratively mark states reachable by some tree (and keep the tree).
- In each round: check whether some transition can be applied using trees that are currently available.

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.

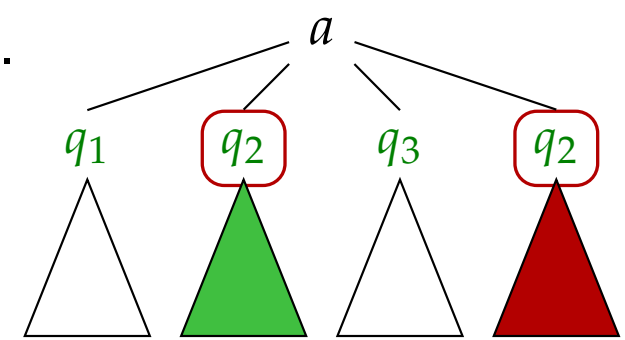
Checking applicability of transitions w.r.t. equality constraints:

- 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.

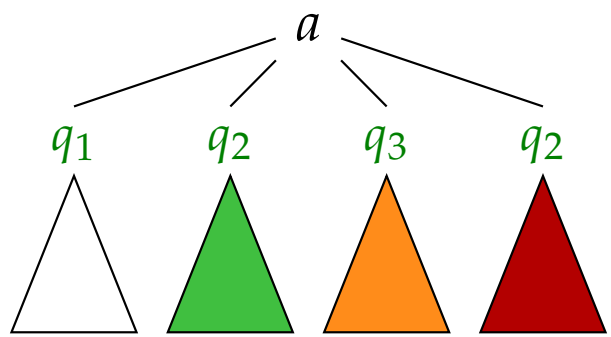
  Example: if "$t_2 \neq t_4$" is required,
  then the transition can only be applied if
  there are at least two trees evaluating to $q_2$.



# . . . 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: ?

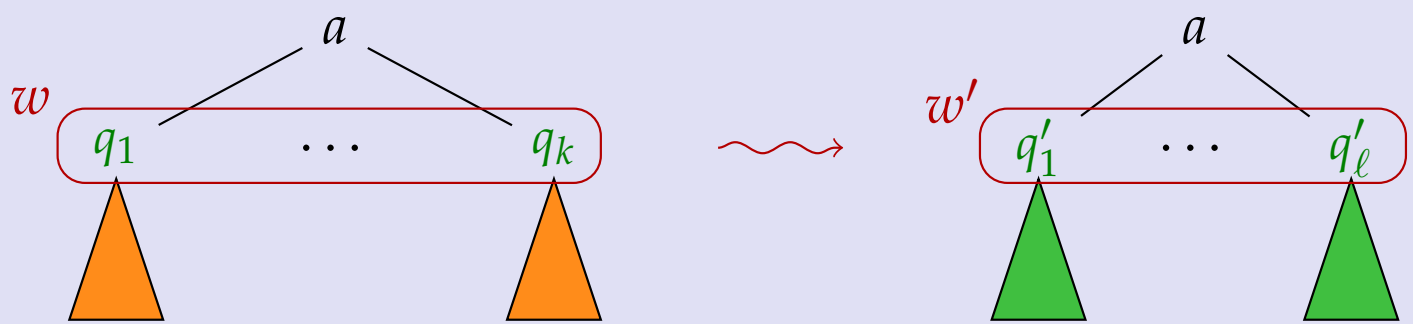
# . . . But How Many Trees to Collect?

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 $w = q_1 \dots q_k$, there is a replacement $w' = q'_1 \dots q'_\ell$ such that the application of $\tau$ using $w'$ needs $\leq B$ distinct trees for each state.*

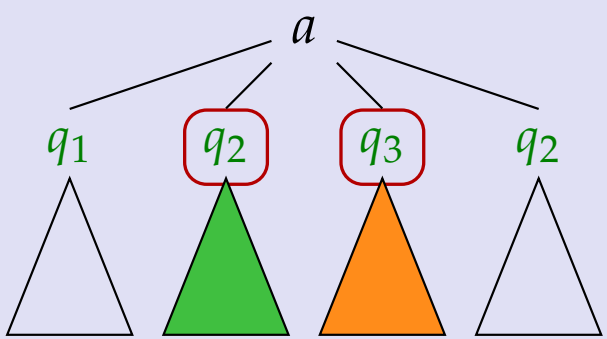



Remark: Our proof yields non-elementary upper bound for $B$.

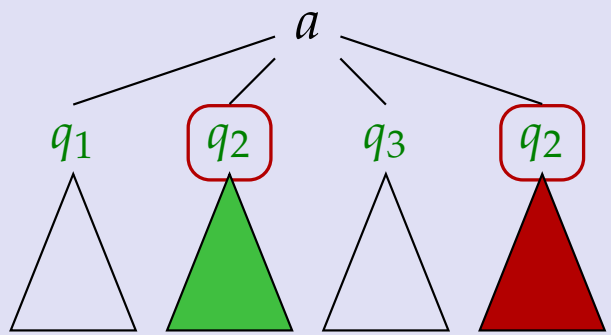
# The Nondeterministic Case

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

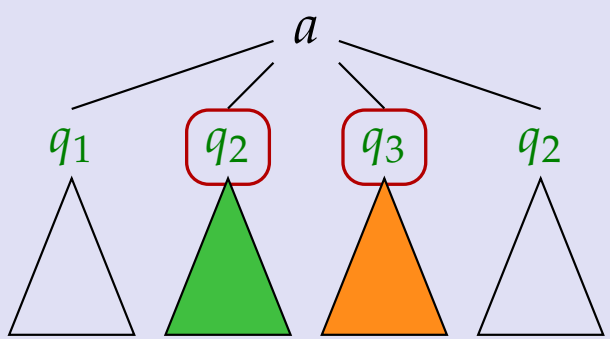


# The Nondeterministic Case

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

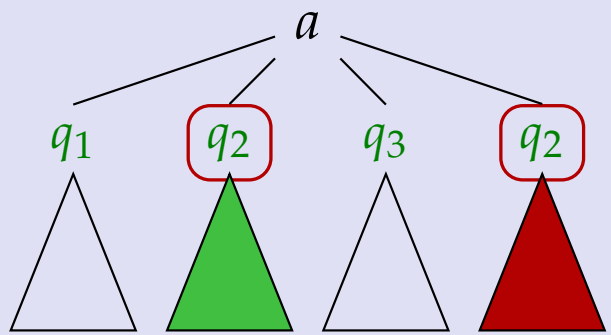
**sets of states**

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

# Automata with Sibling Equalities – Summary

Summary:

- Use of MSO formulas as constraint-addressing mechanism
- Closure under union and intersection
- Nondeterministic UTACSs are more powerful than deterministic ones
- Connection with languages of data words

Decision problems:

- Emptiness is decidable
- Decidability remains if we replace equality with other relations satisfying certain conditions, e.g., structural equality
- Universality is undecidable (via reduction from halting problem for two-register machines)

# Outline

1. Automata with arithmetical constraints
   - Parikh unranked tree automata
   - Emptiness problem

2. Automata with subtree-equality constraints
   - Equality constraints between siblings
   - Emptiness problem

3. Conclusion

# Conclusion

Extensions of unranked tree automata:

- Arithmetical constraints expressed in Presburger logic
- Equality constraints between sibling subtrees, using MSO formulas as constraint-addressing mechanism

For these extensions, the emptiness problem remains decidable.

# Conclusion

Extensions of unranked tree automata:

- Arithmetical constraints expressed in Presburger logic
- Equality constraints between sibling subtrees, using MSO formulas as constraint-addressing mechanism

For these extensions, the emptiness problem remains decidable.

Open problems & further prospects:

- Complexity issues
- Connection with logic
- Further extensions w.r.t.
  - arithmetical properties: frontier constraints, ...
  - subtree equality: global equality constraints, ...

# Appendix

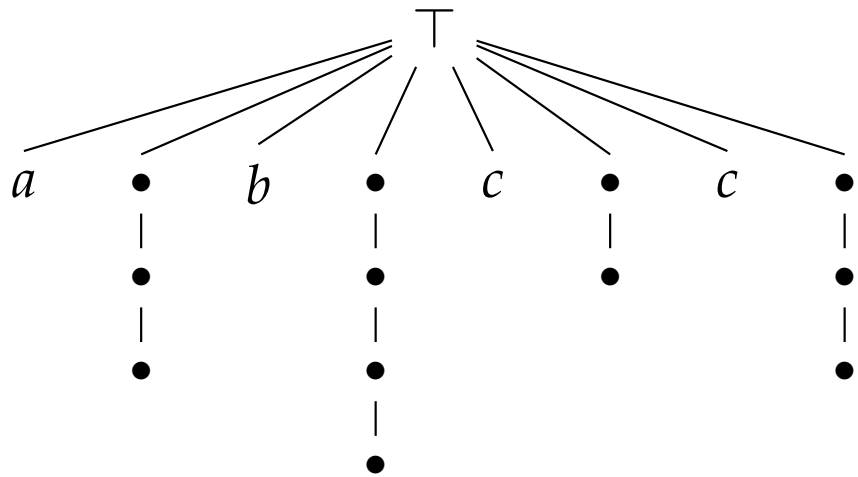
# Encoding Data Words as Unranked Trees

label (finite alphabet)

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

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

$\rightsquigarrow$




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

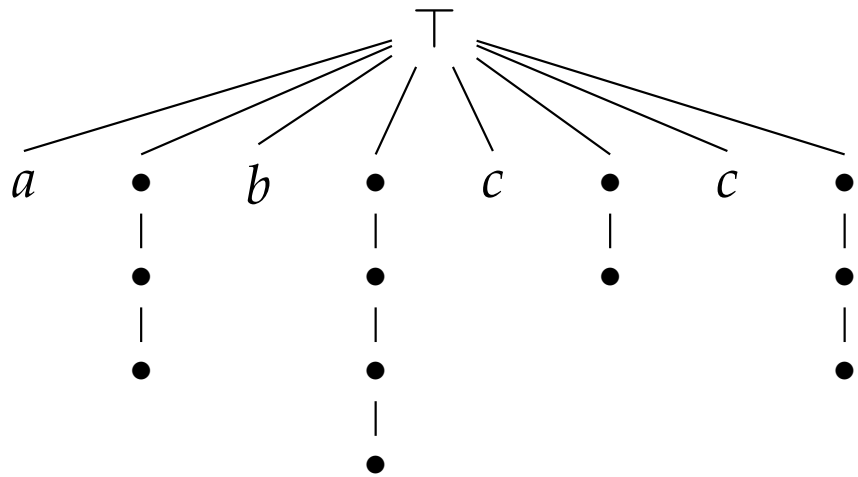
# Encoding Data Words as Unranked Trees

label (finite alphabet)

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

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

$\rightsquigarrow$




- 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"

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

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

$\rightsquigarrow$ 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"

$$\big[\ \forall x \forall y\ .\ x = \min \wedge y = \max\ \rightarrow\ x \sim y\ \big] \wedge$$

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

$\rightsquigarrow$ 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 $a$"

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

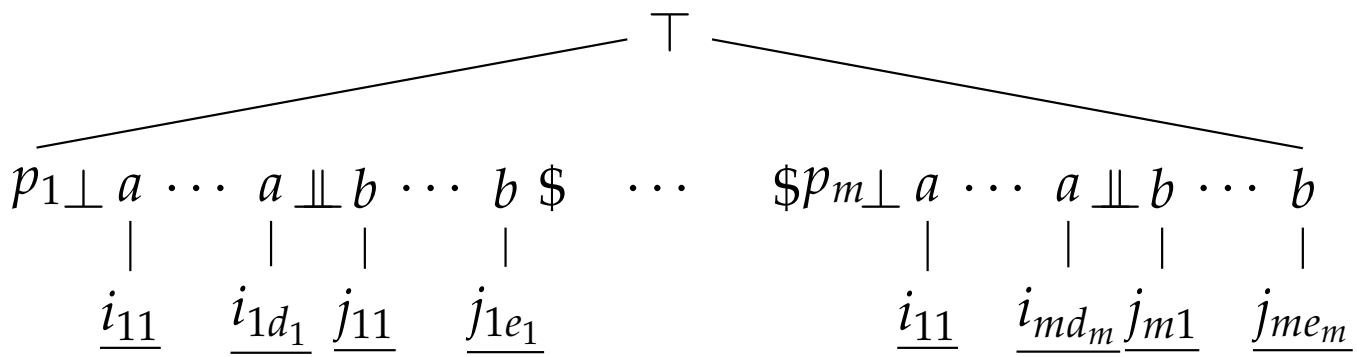
$\rightsquigarrow$ It is still open whether this language is definable in $\mathrm{FO}^2(\sim, <, \mathrm{Succ})$ [Bojanczyk et al '06]

# Universality Problem for UTACS

**Theorem.** *Universality is undecidable for nondeterministic UTACS.*

Proof sketch:

- reduce the halting problem for 2-register machines
- encode computations $(p_1, d_1, e_1) \dots (p_m, d_m, e_m)$ 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