Automata and Logic on Trees

Wim Martens[1] Stijn Vansummeren[2]

[1]University of Dortmund, Germany
[2]Hasselt University, Belgium

Outline

Outline

What am I doing here?

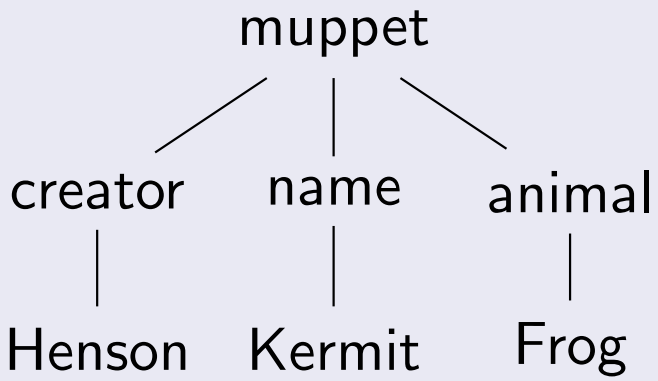
- You know something about finite automata on strings
- You want to know more about finite automata on trees

Why would I want to do that?

Because Tree Automata are important for data on the web

Data on the web is in tree-structured XML

```
<muppet creator="Henson">
  <name> Kermit </name>
  <animal> Frog </animal>
</muppet>
```



Why would I want to do that?

Because Tree Automata are important for data on the web

- They are the basis for XML schema languages and validation
- Form a foundation for XML query languages
- Aid in static verification

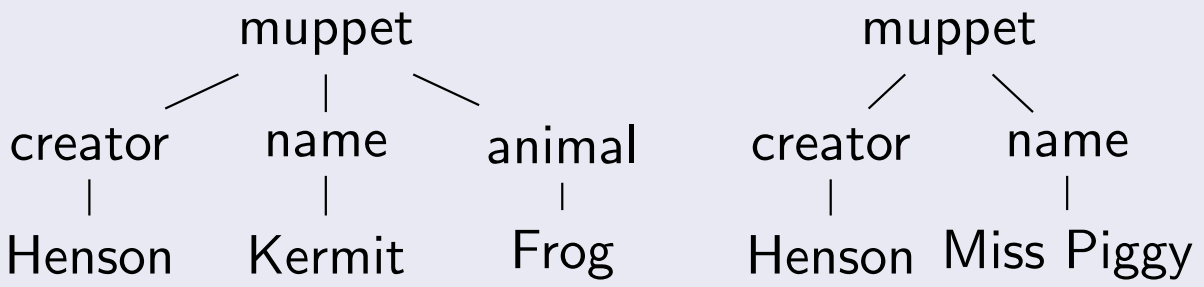
Why would I want to do that?

Because Tree Automata are important for data on the web

A basis for XML schema languages

Constraint: Every muppet node must have a creator and a name, and may have an optional animal.

Examples:



Why would I want to do that?

Because Tree Automata are important for data on the web

A basis for XML schema languages

Constraint: Every muppet node must have a creator and a name, and may have an optional animal.

Many schema languages: DTD, XML Schema, Relax NG, ...

All naturally modeled and executed by tree automata

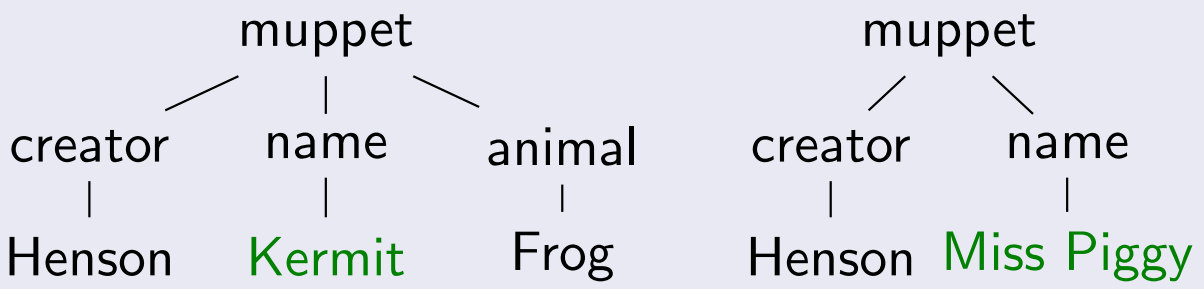
Why would I want to do that?

Because Tree Automata are important for data on the web

As a foundation for XML query languages

Query: Retrieve all muppet names.

Examples:



Why would I want to do that?

Because Tree Automata are important for data on the web

As a foundation for XML query languages

Query: Retrieve all muppet names.

Languages based on Tree Automata: Monadic datalog, Query automata

Why would I want to do that?

Because Tree Automata are important for data on the web

Aid in Static Verification.

Given: Program P, input schema I, output schema O.
Question: Is $P(t) \in O$, for every $t \in I$?

Static Type-checking: XDuce, $\mathbb{C}$Duce, XQuery, XLST, ...

In summary . . .

Tree automata provide the underlying guiding principles for data on the Web

Like the relational calculus (i.e., first-order logic) and the relational algebra provide the underlying principles for relational data.

Who are you guys anyway?

Wim Martens

Stijn Vansummeren

Course Overview

Day 1: Basics on Ranked Tree Automata

Day 2: Algorithms on Ranked Tree Automata

Day 3: Connection with Monadic Second-Order Logic

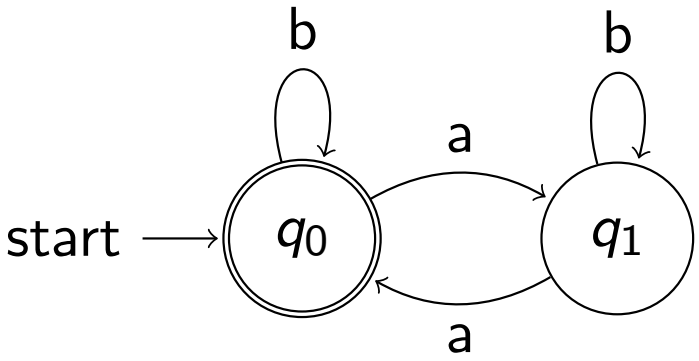
Day 4: Basics and Algorithms on Unranked Tree Automata

Day 5: XML-related applications

Outline

Warmup: Automata on Strings

strings with an even number of a's



Transition Rules

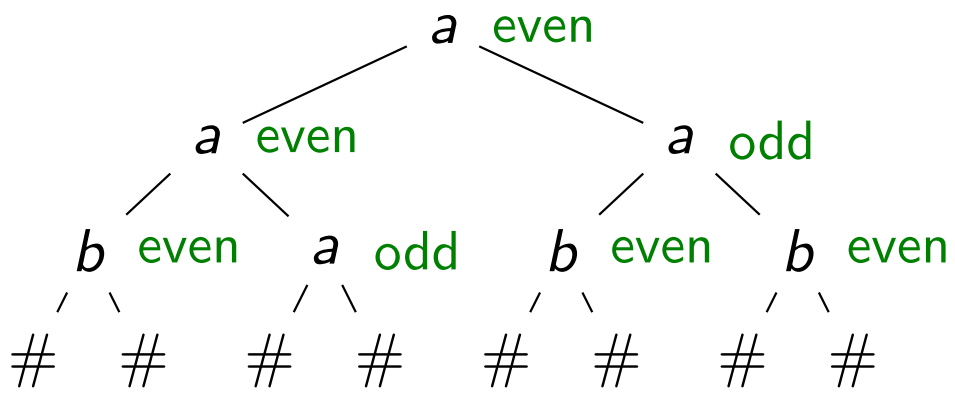
$q_0 \xrightarrow{a} q_1$

$q_0 \xrightarrow{b} q_0$

etc...

From Strings to Trees: Binary Trees

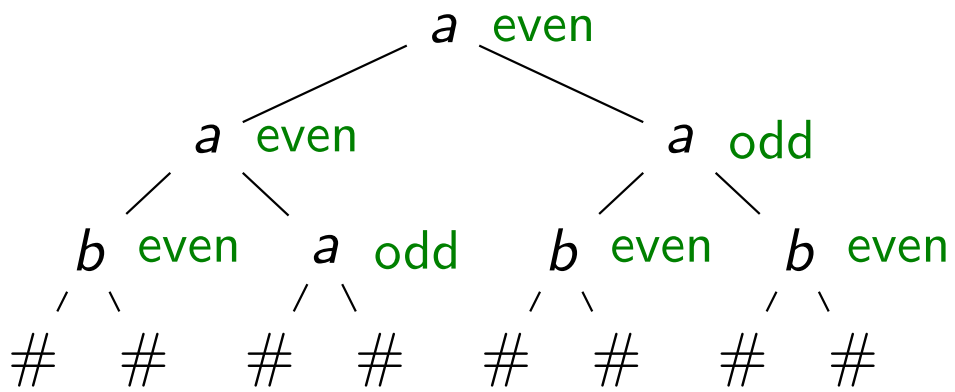
binary trees with an even number of a's



How do we put this into transition rules?

From Strings to Trees: Binary Trees

binary trees with an even number of a's



Transition rules

$(\text{even},\text{odd}) \xrightarrow{a} \text{even}$ (left a-transition)

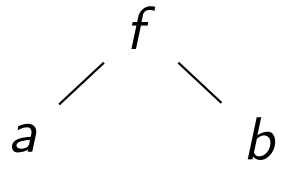
$(\text{even},\text{even}) \xrightarrow{a} \text{odd}$ (right a-transition)

etc...

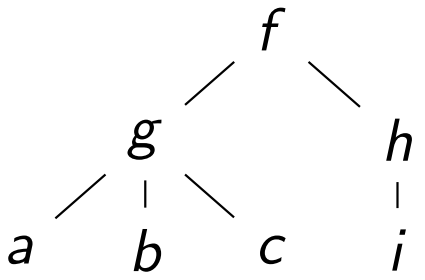
Ranked Trees

What are ranked trees?

A function call $f(a, b)$ is a ranked tree



A function call $f(g(a, b, c), h(i))$ is a ranked tree



Ranked Alphabet

A ranked alphabet symbol is a formalization of a function call

A ranked symbol...

...is a symbol a together with an integer $\text{rank}(a)$

$\text{rank}(a)$ tell us the number of children a is allowed to have

Notation

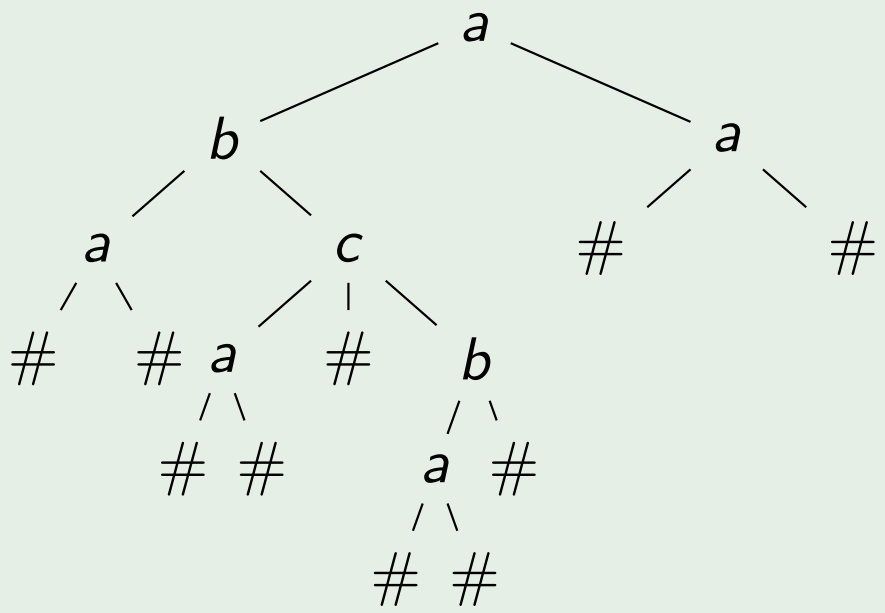
$a^{(k)}$: symbol a with $\text{rank}(a) = k$

Ranked Alphabet: Example

Example

Alphabet: $\{a^{(2)}, b^{(2)}, c^{(3)}, \#^{(0)}\}$

Allowed Tree:



Ranked Tree Automata

A ranked tree automaton A consists of

Alphabet(A): finite set of alphabet symbols

States(A): finite set of states

Rules(A): finite set of transition rules

Final(A): finite set of final states

where

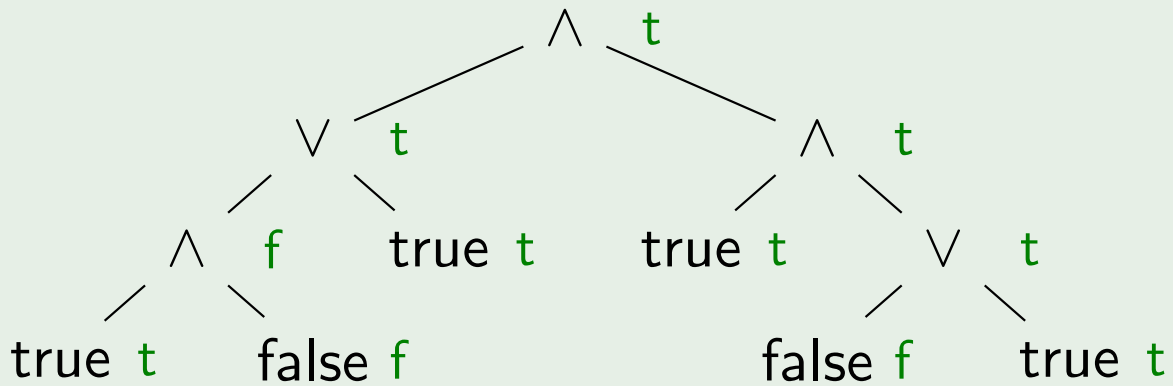
$$\text{Rules}(A) \text{ are of the form } (q_1, \ldots, q_k) \xrightarrow{a^{(k)}} q$$

$$(\text{If } k = 0, \text{ we write } \varepsilon \xrightarrow{a^{(0)}} q)$$

Ranked Tree Automata

How do they work?

Example



$$
\begin{array}{lcl}
\varepsilon & \xrightarrow{\text{true}} & \mathtt{t} \\
\varepsilon & \xrightarrow{\text{false}} & \mathtt{f} \\
(t,t) & \xrightarrow{\wedge} & \mathtt{t} \\
(t,f) & \xrightarrow{\wedge} & \mathtt{f} \\
\ldots & & \\
(f,f) & \xrightarrow{\vee} & \mathtt{f}
\end{array}
$$

If root is labeled by $q \in \mathrm{Final}(A)$:

ACCEPT

Terminology

Terminology

- *Language(A): set of trees accepted by A*
- *Regular tree language: Set of trees S such that $S = Language(A)$ for some ranked tree automaton A*

Example

Tree automaton A over $\{a^{(2)}, b^{(2)}, c^{(3)}, \#^{(0)}\}$ for trees with even number of a's

Alphabet(A): $\{a, b, c, \#\}$

States(A): $\{\text{even}, \text{odd}\}$

Final(A): $\{\text{even}\}$

Rules(A):

$(\text{even}, \text{even}) \xrightarrow{a} \text{odd}$ $(\text{even}, \text{even}) \xrightarrow{b} \text{even}$ $(\text{even}, \text{even}, \textit{even}) \xrightarrow{c} \text{even}$

$(\text{even}, \text{odd}) \xrightarrow{a} \text{even}$ $(\text{even}, \text{odd}) \xrightarrow{b} \text{odd}$ $(\text{even}, \text{even}, \text{odd}) \xrightarrow{c} \text{odd}$

$(\text{odd}, \text{even}) \xrightarrow{a} \text{even}$ $(\text{odd}, \text{even}) \xrightarrow{b} \text{odd}$ $(\text{even}, \text{odd}, \text{even}) \xrightarrow{c} \text{odd}$

$(\text{odd}, \text{odd}) \xrightarrow{a} \text{odd}$ $(\text{odd}, \text{odd}) \xrightarrow{b} \text{even}$ $(\text{even}, \text{odd}, \text{odd}) \xrightarrow{c} \text{even}$

$\varepsilon \xrightarrow{\#} \text{even}$ ...

Deterministic Ranked Tree Automaton

Deterministic:

No two rules of the form

$$(q_1, \ldots, q_k) \xrightarrow{a^{(k)}} q$$
$$(q_1, \ldots, q_k) \xrightarrow{a^{(k)}} q'$$

for different states q and q'

Deterministic Ranked Tree Automaton: Example

$(\text{even}, \text{even}) \xrightarrow{a} \text{odd}$
$(\text{even}, \text{odd}) \xrightarrow{a} \text{even}$
$(\text{odd}, \text{even}) \xrightarrow{a} \text{even}$
$(\text{odd}, \text{odd}) \xrightarrow{a} \text{odd}$
$\varepsilon \xrightarrow{\#} \text{even}$

$(\text{even}, \text{even}) \xrightarrow{b} \text{even}$
$(\text{even}, \text{odd}) \xrightarrow{b} \text{odd}$
$(\text{odd}, \text{even}) \xrightarrow{b} \text{odd}$
$(\text{odd}, \text{odd}) \xrightarrow{b} \text{even}$

$(\text{even}, \text{even}, \textit{even}) \xrightarrow{c} \text{even}$
$(\text{even}, \text{even}, \text{odd}) \xrightarrow{c} \text{odd}$
$(\text{even}, \text{odd}, \text{even}) \xrightarrow{c} \text{odd}$
$(\text{even}, \text{odd}, \text{odd}) \xrightarrow{c} \text{even}$
$\ldots$

Outline

Some immediate natural questions

General questions:

- Are non-deterministic and deterministic ranked tree automata equivalent?
- Are regular tree languages closed under Boolean operations?
- Does it matter whether we read trees top-down or bottom-up?
- Do we have a pumping lemma?
- Can tree automata be minimized?

Complexity questions:

What is the complexity of deciding whether...

- automaton A accepts a tree t?
- automaton A accepts a tree at all?
- automaton A accepts all trees of automaton B?
- a set of automata accept a common tree?

Can Ranked Tree Automata be Determinized?

Take this non-deterministic Tree Automaton:

Example

Automaton with $\text{Final}(A) = q_f$ and rules

$$\varepsilon \xrightarrow{c} q \quad q \xrightarrow{b} q_b \quad q \xrightarrow{b} q \quad q_b \xrightarrow{b} q_f \quad (q, q) \xrightarrow{a} q$$

Can Ranked Tree Automata be Determinized?

Example

Automaton with $\text{Final}(A) = q_f$ and rules

$\varepsilon \xrightarrow{c} q \quad q \xrightarrow{b} q_b \quad q \xrightarrow{b} q \quad q_b \xrightarrow{b} q_f \quad (q, q) \xrightarrow{a} q$

Determinization

$\text{States}(A_{\text{det}}) = \{\{q\}, \{q, q_b\}, \{q, q_b, q_a\}\}$ and rules

$\varepsilon \xrightarrow{c} \{q\}$

$\{q\} \xrightarrow{b} \{q, q_b\}$

$\{q, q_b\} \xrightarrow{b} \{q, q_b, q_f\}$

$\{q, q_b, q_f\} \xrightarrow{b} \{q, q_b, q_f\}$

$(S_1, S_2) \xrightarrow{a} \{q\}$ for all $S_1, S_2 \in \text{States}(A)$

Ranked Automata can be Determinized

Theorem

From each non-deterministic tree automaton, an equivalent deterministic tree automaton can be constructed in exponential time

Corollary

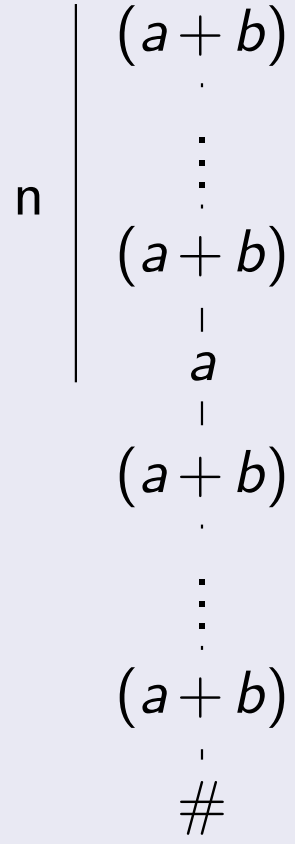
Non-deterministic and deterministic tree automata recognize the same languages

Ranked Automata can be Determinized

Theorem

Ranked Automata can be determinized in exponential time

Is this optimal?



Natural Questions

General questions:

- Are non-deterministic and deterministic ranked tree automata equivalent? **Yes.**
- Are regular tree languages closed under Boolean operations?
- Does it matter whether we read trees top-down or bottom-up?
- Do we have a pumping lemma?
- Can tree automata be minimized?

Complexity questions:

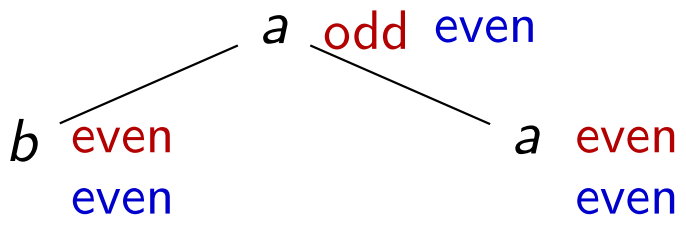
What is the complexity of deciding whether...

- automaton A accepts a tree t?
- automaton A accepts a tree at all?
- automaton A accepts all trees of automaton B?
- a set of automata accept a common tree?

Closure under Union and Intersection: Product Construction

Example

- Automaton A: even number of a's
  - $(\text{even}, \text{even}) \xrightarrow{a} \text{odd}$
- Automaton B: even number of b's
  - $(\text{even}, \text{even}) \xrightarrow{a} \text{even}$



$$((\text{even}, \text{even}), (\text{even}, \text{even})) \xrightarrow{a} (\text{odd}, \text{even})$$

Closure under Union and Intersection

Product Construction:

Given A, B, construct $A \times B$

- $\text{Alphabet}(A \times B) = \text{Alphabet}(A) \cup \text{Alphabet}(B)$
- $\text{States}(A \times B) = \text{States}(A) \times \text{States}(B)$
- $\text{Final}(A \times B) = \{(s_A, s_B) \mid s_A \in \text{Final}(A) \wedge s_B \in \text{Final}(B)\}$
- $\text{Rules}(A \times B) = \{$

$$((s_A^1, s_B^1), \ldots, (s_A^k, s_B^k)) \xrightarrow{a^{(k)}} (s_A, s_B) \mid$$
$$(s_A^1, \ldots, s_A^k) \xrightarrow{a^{(k)}} s_A \in \text{Rules}(A)$$
$$(s_B^1, \ldots, s_B^k) \xrightarrow{a^{(k)}} s_B \in \text{Rules}(B)\}$$

Closure under Union and Intersection: Union

Given A, B, construct $A \cup B$

- $\text{Alphabet}(A \cup B) = \text{Alphabet}(A) \cup \text{Alphabet}(B)$
- $\text{States}(A \cup B) = \text{States}(A) \times \text{States}(B)$
- $\text{Rules}(A \cup B) = \{$

$$((s_A^1, s_B^1), \ldots, (s_A^k, s_B^k)) \xrightarrow{a^{(k)}} (s_A, s_B) \mid$$
$$(s_A^1, \ldots, s_A^k) \xrightarrow{a^{(k)}} s_A \in \text{Rules}(A)$$
$$(s_B^1, \ldots, s_B^k) \xrightarrow{a^{(k)}} s_B \in \text{Rules}(B)\}$$

- $\text{Final}(A \cup B) = \{(s_A, s_B) \mid s_A \in \text{Final}(A) \vee s_B \in \text{Final}(B)\}$

Closure under Union and Intersection: Intersection

Given A, B, construct $A \cap B$

- $\text{Alphabet}(A \cap B) = \text{Alphabet}(A) \cup \text{Alphabet}(B)$
- $\text{States}(A \cap B) = \text{States}(A) \times \text{States}(B)$
- $\text{Rules}(A \cap B) = \{$

$$((s_A^1, s_B^1), \ldots, (s_A^k, s_B^k)) \xrightarrow{a^{(k)}} (s_A, s_B) \mid$$
$$(s_A^1, \ldots, s_A^k) \xrightarrow{a^{(k)}} s_A \in \text{Rules}(A)$$
$$(s_B^1, \ldots, s_B^k) \xrightarrow{a^{(k)}} s_B \in \text{Rules}(B)\}$$

- $\text{Final}(A \cap B) = \{(s_A, s_B) \mid s_A \in \text{Final}(A) \wedge s_B \in \text{Final}(B)\}$

Product Construction: Blow-up

Product Construction: Blow-up

- $|\text{States}(A \times B)| = |\text{States}(A)| \times |\text{States}(B)|$
- $|\text{Rules}(A \times B)| \leq |\text{Rules}(A)| \times |\text{Rules}(B)|$

Blow-up: Quadratic

Closure under Complement: Completion

Definition (Complete tree automaton)

For each $a^{(k)} \in \text{Alphabet}(A)$ and $q_1, \ldots, q_k \in \text{States}(A)$, there is a rule

$$(q_1, \ldots, q_k) \xrightarrow{a} q$$

for some q.

Example (Incomplete (deterministic) Tree Automaton)

Tree automaton A for $\{a(b, b)\}$:

$\varepsilon \xrightarrow{b} q_b$

$(q_b, q_b) \xrightarrow{a} q_a$

with $\text{Final}(A) = q_a$

Closure under Complement: Completion

Example (Incomplete (deterministic) Tree Automaton)

Tree automaton A for $\{a(b,b)\}$:

$\varepsilon \xrightarrow{b} q_b$

$(q_b, q_b) \xrightarrow{a} q_a$

with $\text{Final}(A) = q_a$

Example (Complementing A)

Add a sink state q_s:

$\varepsilon \xrightarrow{b} q_b$ $\quad \varepsilon \xrightarrow{a} q_s$

$(q_b, q_b) \xrightarrow{a} q_a$ $\quad (q_b, q_a) \xrightarrow{a} q_s$ $\quad (q_a, q_b) \xrightarrow{a} q_s$ $(q_a, q_a) \xrightarrow{a} q_s$

$(q_b, q_b) \xrightarrow{b} q_s$ $\quad (q_b, q_a) \xrightarrow{b} q_s$ $\quad (q_a, q_b) \xrightarrow{b} q_s$ $(q_a, q_a) \xrightarrow{b} q_s$

$(q_s, q) \xrightarrow{a,b} q_s$ for all $q \in \{q_a, q_b, q_s\}$

$(q, q_s) \xrightarrow{a,b} q_s$ for all $q \in \{q_a, q_b, q_s\}$

with $\text{Final}(A) = \{q_b, q_s\}$

Closure under Complement

Complementing A

(1) Determinize A

(2) Complete the result

(3) Switch final $\leftrightarrow$ non-final states

Complement construction: Blow-up

- **Determinizing** A: exponential blow-up (states: $2^{\text{States}(A)}$)
- **Completing the result**: blow-up in rules: $|\text{Alphabet}| \times (2^{|\text{States}(A)|})^k$, where k is maximal rank Alphabet(A)
- **Switching final $\leftrightarrow$ non-final states**: linear

Overall: Exponential blow-up

Natural Questions

General questions:

- Are non-deterministic and deterministic ranked tree automata equivalent? **Yes.**
- Are regular tree languages closed under Bool. operations? **Yes.**
- Does it matter whether we read trees top-down or bottom-up?
- Do we have a pumping lemma?
- Can tree automata be minimized?

Complexity questions:

What is the complexity of deciding whether...

- automaton A accepts a tree t?
- automaton A accepts a tree at all?
- automaton A accepts all trees of automaton B?
- a set of automata accept a common tree?

Top-Down Tree Automata: Connection to Strings

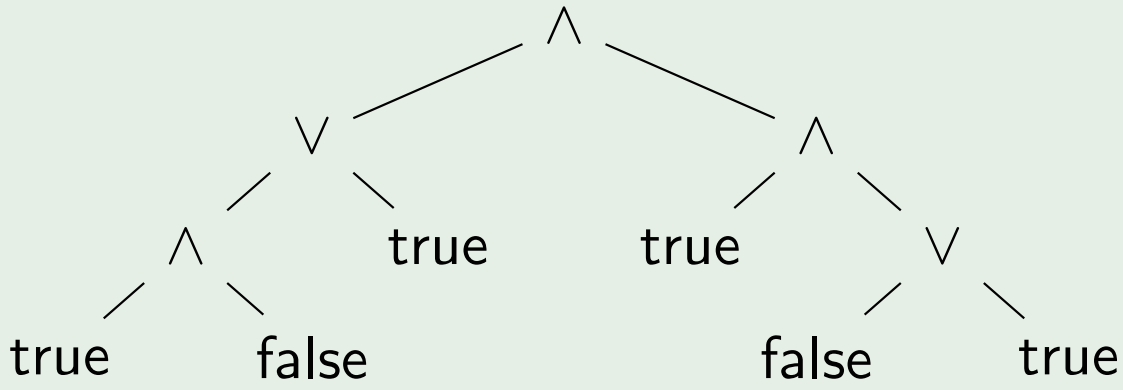
Example (Strings are Unary Trees)

$$\text{abcd} = \text{a(b(c(d)))} = \begin{array}{c} \text{a} \\ | \\ \text{b} \\ | \\ \text{c} \\ | \\ \text{d} \end{array}$$

Reading strings left-to-right
= Reading trees top-down

Top-Down Tree Automata

Example

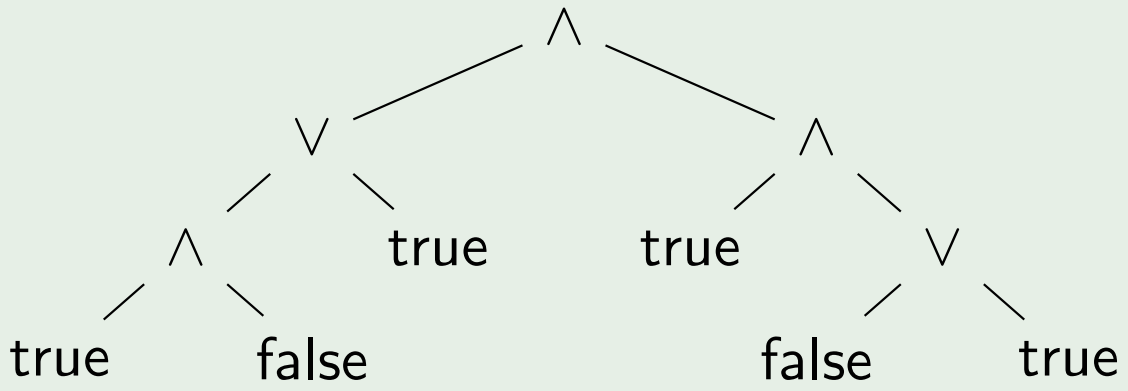


$$
\begin{array}{llll}
\varepsilon & \xrightarrow{\text{true}} & \mathrm{t} & \\
\varepsilon & \xrightarrow{\text{false}} & \mathrm{f} & \\
(t,t) & \xrightarrow{\wedge} & \mathrm{t} & (t,t) \xrightarrow{\vee} \mathrm{t} \\
(t,f) & \xrightarrow{\wedge} & \mathrm{f} & (t,f) \xrightarrow{\vee} \mathrm{t} \\
(f,t) & \xrightarrow{\wedge} & \mathrm{f} & (f,t) \xrightarrow{\vee} \mathrm{t} \\
(f,f) & \xrightarrow{\wedge} & \mathrm{f} & (f,f) \xrightarrow{\vee} \mathrm{f}
\end{array}
$$

We need different rules

Top-Down Tree Automata

Example



$\mathsf{Init}(A) = t$

$$
\begin{array}{lll}
t \xrightarrow{\wedge} (t,t) & \qquad & \mathrm{t} \xrightarrow{\vee} (\mathrm{t},\mathrm{t}) \\
f \xrightarrow{\wedge} (t,f) & & \mathrm{t} \xrightarrow{\vee} (\mathrm{t},\mathrm{f}) \\
f \xrightarrow{\wedge} (f,t) & & \mathrm{t} \xrightarrow{\vee} (\mathrm{f},\mathrm{t}) \\
f \xrightarrow{\wedge} (f,f) & & \mathrm{f} \xrightarrow{\vee} (\mathrm{f},\mathrm{f}) \\
t \xrightarrow{\text{true}} \varepsilon & & \mathrm{f} \xrightarrow{\text{false}} \varepsilon
\end{array}
$$

Top-Down Tree Automata

A top-down tree automaton A consists of

Alphabet(A): finite set of alphabet symbols

States(A): finite set of states

Rules(A): finite set of transition rules

Init(A): finite set of initial states

where

$$\text{Rules}(A) \text{ are of the form } q \xrightarrow{a^{(k)}} (q_1, \ldots, q_k)$$

Top-down tree automata also recognize all regular tree languages

Can Top-Down Tree Automata Be Determinized?

Top-Down Deterministic Tree Automaton

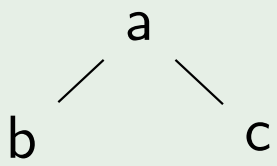
For every $q \in \text{States}(A)$ and $a \in \text{Alphabet}(A)$ there's at most one rule

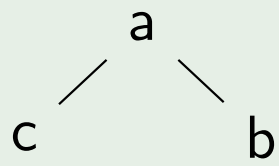
$$q \xrightarrow{a^{(k)}} (q_1, \ldots, q_k)$$

Can Top-Down Tree Automata Be Determinized?

Top-Down TA do not recognize all regular languages!

Example





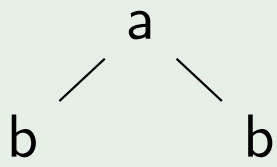
$\mathsf{Init}(A) = q_0$

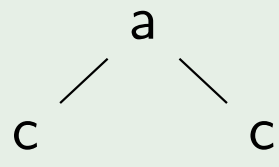
$q_0 \xrightarrow{a} (q, q)$

$q \xrightarrow{b} \varepsilon$

$q \xrightarrow{c} \varepsilon$

also recognizes...





Natural Questions

General questions:

- Are non-deterministic and deterministic ranked tree automata equivalent? **Yes.**
- Are regular tree languages closed under Bool. operations? **Yes.**
- Does it matter whether we read trees top-down or bottom-up? **Yes.**
- Do we have a pumping lemma?
- Can tree automata be minimized?

Complexity questions:

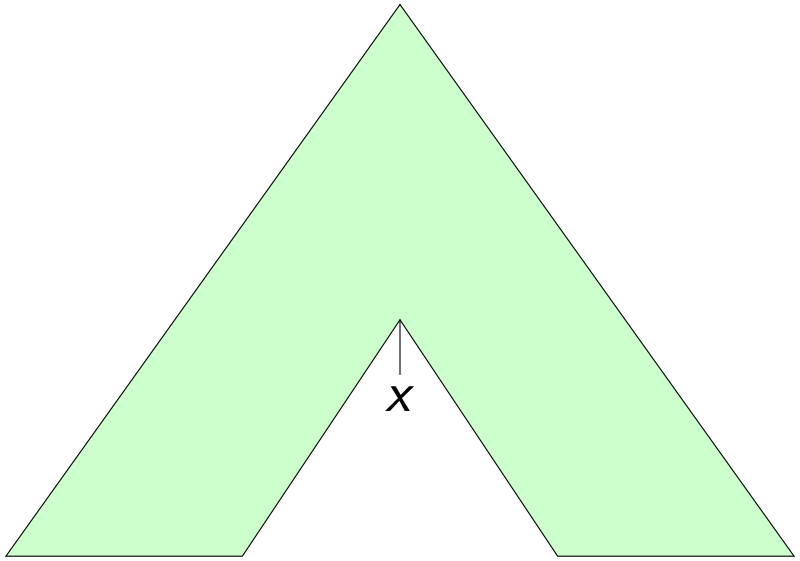
What is the complexity of deciding whether...

- automaton A accepts a tree t?
- automaton A accepts a tree at all?
- automaton A accepts all trees of automaton B?
- a set of automata accept a common tree?

Pumping Lemma

Definition (Context)

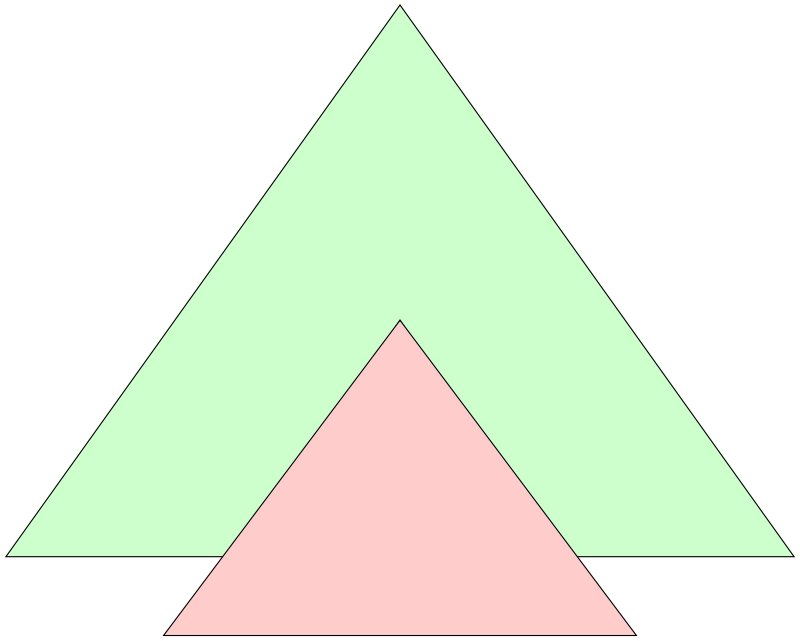
A context is a tree with a hole



Pumping Lemma

Definition (Context Application)

Given the context C and a tree t, $C[t]$ is obtained by plugging t into C.



One can also apply a context to a context: $C_1[C_2]$
The result is a new context

The Pumping Lemma

Lemma (Pumping Lemma)

If L is regular, there is a constant k such that,

- *for every tree $t \in L$ with depth at least k*
- *there is a context C_1, a (non-empty) context C_2, and a small tree t'*

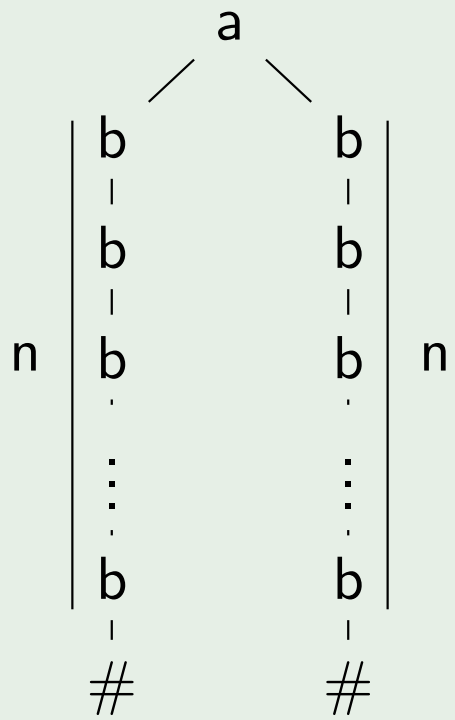
such that

- $t = C_1[C_2'[t']]$ *and*
- *for every $n \in \mathbb{N}$, $C_1[C_2^n[t']]$ is in L.*

What's the use?

Show that languages are not regular.

Example (The Usual Suspect)



Natural Questions

General questions:

- Are non-deterministic and deterministic ranked tree automata equivalent? **Yes.**
- Are regular tree languages closed under Bool. operations? **Yes.**
- Does it matter whether we read trees top-down or bottom-up? **Yes.**
- Do we have a pumping lemma? **Yes.**
- Can tree automata be minimized?

Complexity questions:

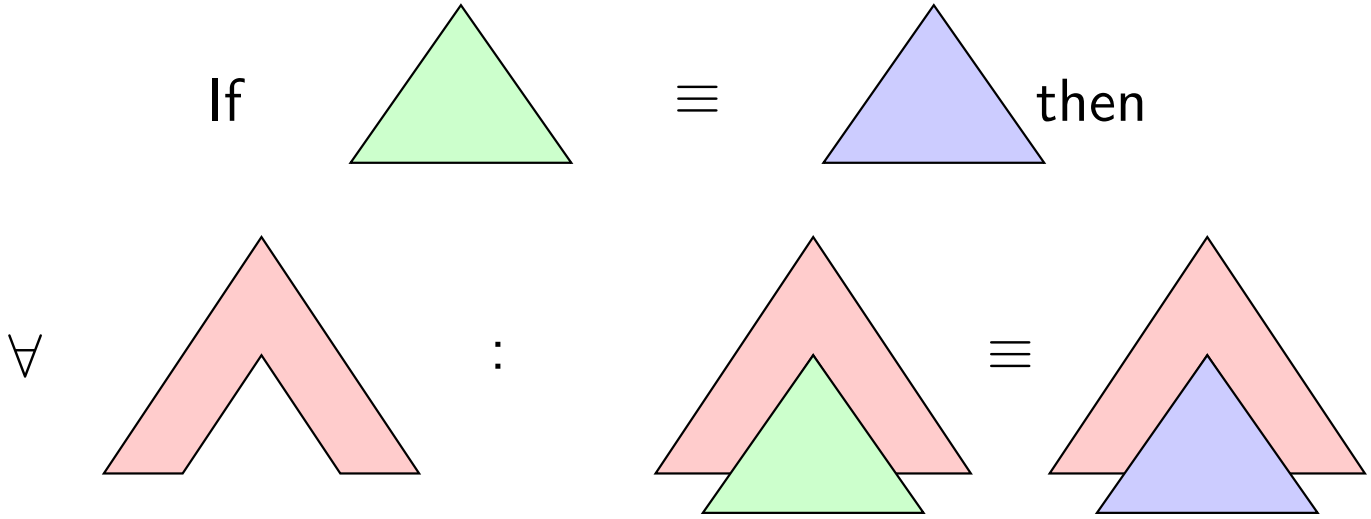
What is the complexity of deciding whether...

- automaton A accepts a tree t?
- automaton A accepts a tree at all?
- automaton A accepts all trees of automaton B?
- a set of automata accept a common tree?

Minimization: Myhill-Nerode Theorem

Definition (Congruence)

A congruence is an equivalence relation on trees closed under context



Minimization: Myhill-Nerode Theorem

Definition (Congruence of a Language)

If L is a tree language then we define

$$t_1 \equiv_L t_2 \text{ if, for all contexts } C\text{: } C[t_1] \in L \Leftrightarrow C[t_2] \in L$$

Minimization: Myhill-Nerode Theorem

Theorem (Myhill-Nerode for Trees)

The following are equivalent:

(a) *L is a regular tree language*

(b) *L is the union of some equivalence classes of finite index*

(c) *the relation $\equiv_L$ is a congruence of finite index*

Minimal automaton A for L follows from this classification:
size of A = number of equivalence classes of $\equiv_L$

Natural Questions

General questions:

- Are non-deterministic and deterministic ranked tree automata equivalent? **Yes.**
- Are regular tree languages closed under Bool. operations? **Yes.**
- Does it matter whether we read trees top-down or bottom-up? **Yes.**
- Do we have a pumping lemma? **Yes.**
- Can tree automata be minimized? **Yes.**

Complexity questions:

What is the complexity of deciding whether...

- automaton A accepts a tree t?
- automaton A accepts a tree at all?
- automaton A accepts all trees of automaton B?
- a set of automata accept a common tree?