

# Foundations of XML Types: Tree Automata

Pierre Genevès  
CNRS

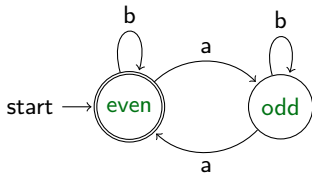
(slides mostly based on slides by W. Martens and T. Schwentick)

University Grenoble Alpes

# Why Tree Automata?

- Foundations of XML type languages (DTD, XML Schema, Relax NG...)
- Provide a general framework for XML type languages
- A tool to define regular tree languages with an operational semantics
- Provide algorithms for efficient validation
- Basic tool for static analysis (proofs, decision procedures in logic)

# Prelude: Word Automata



## Transitions

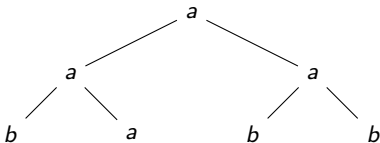
*even*  $\xrightarrow{a}$  *odd*

*odd*  $\xrightarrow{a}$  *even*

...

# From Words to Trees: Binary Trees

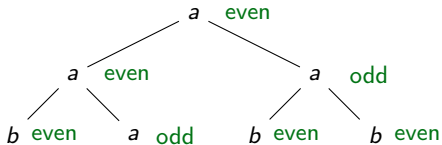
Binary trees with an even number of *a*'s



How to write transitions?

# From Words to Trees: Binary Trees

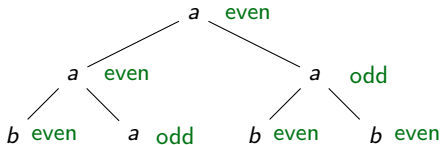
Binary trees with an even number of *a*'s



How to write transitions?

# From Words to Trees: Binary Trees

Binary trees with an even number of *a*'s



How to write transitions?

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

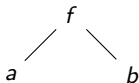
$(\text{even}, \text{even}) \xrightarrow{a} \text{odd}$

*etc.*

# Ranked Trees?

They come from *parse trees* of data (or programs)...

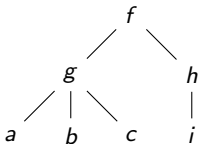
A function call  $f(a, b)$  is a ranked tree



# Ranked Trees?

They come from *parse trees* of data (or programs)...

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





# Ranked Alphabet

*A ranked alphabet symbol is:*

- a formalisation of a function call
- a symbol  $a$  with an integer  $\text{arity}(a)$ 
  - $\text{arity}(a)$  indicates the number of children of  $a$

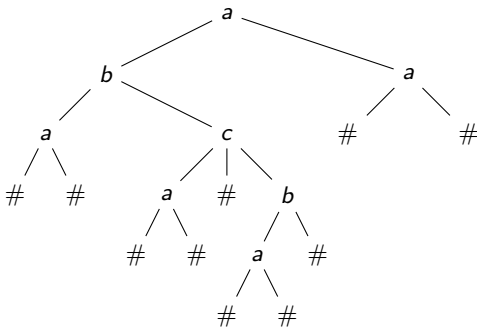
## Notation

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

# Example

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

Possible tree:



# Ranked Tree Automata

A ranked tree automaton  $A$  consists in:

Alphabet( $A$ ): finite alphabet of symbols

States( $A$ ): finite set of states

Rules( $A$ ): finite set of transition rules

Final( $A$ ): finite set of final states ( $\subseteq$  States( $A$ ))

where :

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

if  $k = 0$ , we write  $\epsilon \xrightarrow{a^{(0)}} q$

# Ranked Tree Automata

A ranked tree automaton  $A$  consists in:

Alphabet( $A$ ): finite alphabet of symbols

States( $A$ ): finite set of states

Rules( $A$ ): finite set of transition rules

Final( $A$ ): finite set of final states ( $\subseteq$  States( $A$ ))

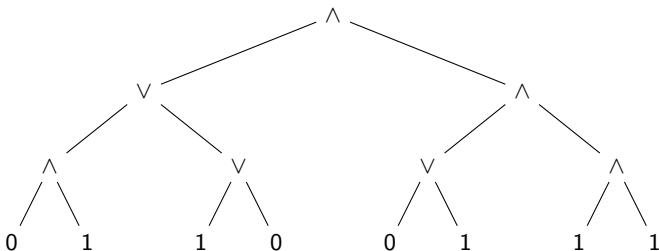
where :

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

if  $k = 0$ , we write  $\epsilon \xrightarrow{a^{(0)}} q$

# How do they work?

## Example: Boolean Expressions



## Principle

- $\text{Alphabet}(A) = \{\wedge, \vee, 0, 1\}$
- $\text{States}(A) = \{q_0, q_1\}$
- 1 accepting state at the root:  
 $\text{Final}(A) = \{q_1\}$

## Rules(A)

$$\epsilon \xrightarrow{0} q_0$$

$$(q_1, q_1) \xrightarrow{\wedge} q_1$$

$$(q_0, q_1) \xrightarrow{\wedge} q_0$$

$$(q_1, q_0) \xrightarrow{\wedge} q_0$$

$$(q_0, q_0) \xrightarrow{\wedge} q_0$$

$$\epsilon \xrightarrow{1} q_1$$

$$(q_0, q_1) \xrightarrow{\vee} q_1$$

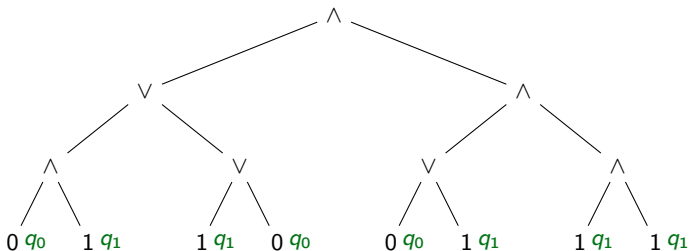
$$(q_1, q_0) \xrightarrow{\vee} q_1$$

$$(q_1, q_1) \xrightarrow{\vee} q_1$$

$$(q_0, q_0) \xrightarrow{\vee} q_0$$

# How do they work?

## Example: Boolean Expressions



## Principle

- $\text{Alphabet}(A) = \{\wedge, \vee, 0, 1\}$
- $\text{States}(A) = \{q_0, q_1\}$
- 1 accepting state at the root:  
 $\text{Final}(A) = \{q_1\}$

## Rules(A)

$$\epsilon \xrightarrow{0} q_0$$

$$(q_1, q_1) \xrightarrow{\wedge} q_1$$

$$(q_0, q_1) \xrightarrow{\wedge} q_0$$

$$(q_1, q_0) \xrightarrow{\wedge} q_0$$

$$(q_0, q_0) \xrightarrow{\wedge} q_0$$

$$\epsilon \xrightarrow{1} q_1$$

$$(q_0, q_1) \xrightarrow{\vee} q_1$$

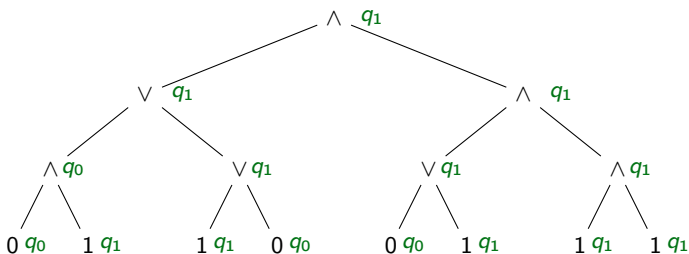
$$(q_1, q_0) \xrightarrow{\vee} q_1$$

$$(q_1, q_1) \xrightarrow{\vee} q_1$$

$$(q_0, q_0) \xrightarrow{\vee} q_0$$

# How do they work?

## Example: Boolean Expressions



## Principle

- $\text{Alphabet}(A) = \{\wedge, \vee, 0, 1\}$
- $\text{States}(A) = \{q_0, q_1\}$
- 1 accepting state at the root:  
 $\text{Final}(A) = \{q_1\}$

## Rules(A)

$$\begin{array}{ll} \epsilon \xrightarrow{0} q_0 & \epsilon \xrightarrow{1} q_1 \\ (q_1, q_1) \xrightarrow{\wedge} q_1 & (q_0, q_1) \xrightarrow{\vee} q_1 \\ (q_0, q_1) \xrightarrow{\wedge} q_0 & (q_1, q_0) \xrightarrow{\vee} q_1 \\ (q_1, q_0) \xrightarrow{\wedge} q_0 & (q_1, q_1) \xrightarrow{\vee} q_1 \\ (q_0, q_0) \xrightarrow{\wedge} q_0 & (q_0, q_0) \xrightarrow{\vee} q_0 \end{array}$$

# Terminology

- $\text{Language}(A)$  : set of trees accepted by  $A$
- For a tree automaton  $A$ ,  $\text{Language}(A)$  is a *regular tree language* [Thatcher and Wright, 1968, Doner, 1970]



# Example

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

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

States( $A$ ) :

Final( $A$ ) :

Rules( $A$ ) :

# Example

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

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

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

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

Rules( $A$ ) :

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

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

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

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

$\epsilon \xrightarrow{\#} \text{even}$

# Outline

1. Can we implement a tree automaton efficiently? (notion of determinism)
  2. Are tree automata closed under set-theoretic operations?
  3. Can we check type inclusion?
  4. Can we build equivalent top-down tree automata?
  5. Nice theory. But... what should I do with my *unranked* XML trees?
- Can we apply this for XML type-checking?

# Deterministic Tree Automata

## Deterministic

does not have two rules of the form:

$$\begin{aligned}(q_1, \dots, q_k) &\xrightarrow{a^{(k)}} q \\ (q_1, \dots, q_k) &\xrightarrow{a^{(k)}} q'\end{aligned}$$

for two different states  $q$  and  $q'$

## Intuition

At most one possible transition at a given node  $\rightarrow$  implementation...

# Can we Make a Tree Automaton Deterministic?

## Theorem (determinization)

From a given non-deterministic (bottom-up) tree automaton we can build a deterministic tree automaton

## Corollary

Non-deterministic and deterministic (bottom-up) tree automata recognize the same languages.

## Complexity

EXPTIME ( $|\text{States}(A_{\text{det}})| = 2^{|\text{States}(A)|}$ )

# Implementing Validation

## Membership Checking

Given a tree automaton  $A$  and a tree  $t$ , does  $t \in \text{Language}(A)$ ?

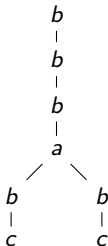
### Remark

We can implement even if  $A$  is non-deterministic...

### Example

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

$$\begin{aligned} \epsilon &\xrightarrow{c} q & q &\xrightarrow{b} q_b & q &\xrightarrow{b} q \\ q_b &\xrightarrow{b} q_f & (q, q) &\xrightarrow{a} q \end{aligned}$$



# Implementing Validation

## Membership Checking

Given a tree automaton  $A$  and a tree  $t$ , does  $t \in \text{Language}(A)$ ?

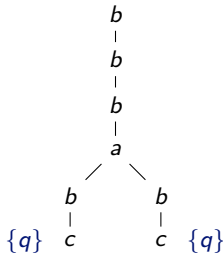
### Remark

We can implement even if  $A$  is non-deterministic...

### Example

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

$$\begin{aligned} \epsilon &\xrightarrow{c} q & q &\xrightarrow{b} q_b & q &\xrightarrow{b} q \\ q_b &\xrightarrow{b} q_f & (q, q) &\xrightarrow{a} q \end{aligned}$$



# Implementing Validation

## Membership Checking

Given a tree automaton  $A$  and a tree  $t$ , does  $t \in \text{Language}(A)$ ?

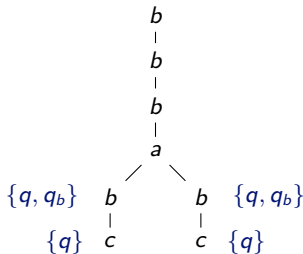
### Remark

We can implement even if  $A$  is non-deterministic...

### Example

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

$$\begin{aligned} \epsilon &\xrightarrow{c} q & q &\xrightarrow{b} q_b & q &\xrightarrow{b} q \\ q_b &\xrightarrow{b} q_f & (q, q) &\xrightarrow{a} q \end{aligned}$$





# Implementing Validation

## Membership Checking

Given a tree automaton  $A$  and a tree  $t$ , does  $t \in \text{Language}(A)$ ?

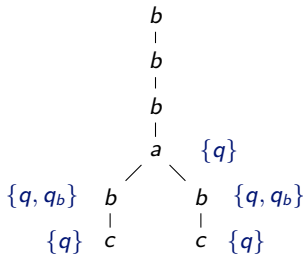
### Remark

We can implement even if  $A$  is non-deterministic...

### Example

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

$$\begin{aligned} \epsilon &\xrightarrow{c} q & q &\xrightarrow{b} q_b & q &\xrightarrow{b} q \\ q_b &\xrightarrow{b} q_f & (q, q) &\xrightarrow{a} q \end{aligned}$$



# Implementing Validation

## Membership Checking

Given a tree automaton  $A$  and a tree  $t$ , does  $t \in \text{Language}(A)$ ?

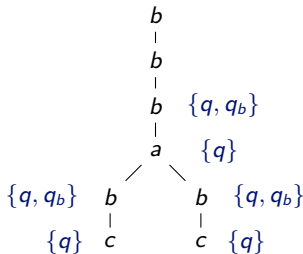
### Remark

We can implement even if  $A$  is non-deterministic...

### Example

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

$$\begin{aligned} \epsilon &\xrightarrow{c} q & q &\xrightarrow{b} q_b & q &\xrightarrow{b} q \\ q_b &\xrightarrow{b} q_f & (q, q) &\xrightarrow{a} q \end{aligned}$$



# Implementing Validation

## Membership Checking

Given a tree automaton  $A$  and a tree  $t$ , does  $t \in \text{Language}(A)$ ?

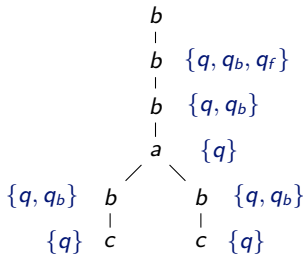
### Remark

We can implement even if  $A$  is non-deterministic...

### Example

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

$$\begin{aligned} \epsilon &\xrightarrow{c} q & q &\xrightarrow{b} q_b & q &\xrightarrow{b} q_f \\ q_b &\xrightarrow{b} q_f & (q, q) &\xrightarrow{a} q \end{aligned}$$



# Implementing Validation

## Membership Checking

Given a tree automaton  $A$  and a tree  $t$ , does  $t \in \text{Language}(A)$ ?

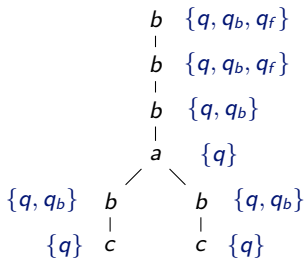
### Remark

We can implement even if  $A$  is non-deterministic...

### Example

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

$$\begin{aligned} \epsilon &\xrightarrow{c} q & q &\xrightarrow{b} q_b & q &\xrightarrow{b} q \\ q_b &\xrightarrow{b} q_f & (q, q) &\xrightarrow{a} q \end{aligned}$$



# Implementing Validation

## Membership Checking

Given a tree automaton  $A$  and a tree  $t$ , does  $t \in \text{Language}(A)$ ?

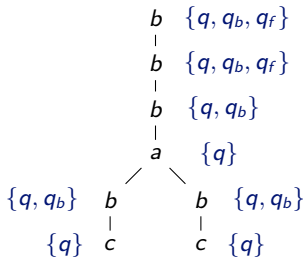
### Remark

We can implement even if  $A$  is non-deterministic...

### Example

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

$$\begin{aligned} \epsilon &\xrightarrow{c} q & q &\xrightarrow{b} q_b & q &\xrightarrow{b} q \\ q_b &\xrightarrow{b} q_f & (q, q) &\xrightarrow{a} q \end{aligned}$$



## Complexity

Membership-Checking is in PTIME (time linear in the size of the tree)

# Set-Theoretic Operations

## Recall

- We have seen that neither local tree grammars nor single-type tree grammars are closed under boolean operations (e.g. union)
- What about tree automata?

# Closure under Union and Intersection...

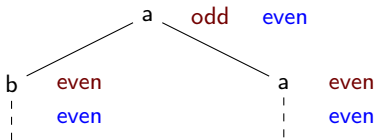
## Example

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

# Closure under Union and Intersection...

## Example

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

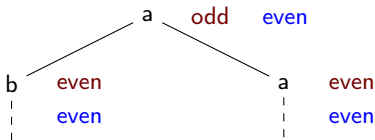




# Closure under Union and Intersection...

## Example

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



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

# Product Construction

Given  $A$  and  $B$ , build  $A \times B$

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

$$\left\{ \left( (q_a^1, q_b^1), \dots, (q_a^k, q_b^k) \right) \xrightarrow{a^{(k)}} (q_a, q_b) \mid \begin{array}{l} q_a^1, \dots, q_a^k \xrightarrow{a^{(k)}} q_a \in \text{Rules}(A) \\ q_b^1, \dots, q_b^k \xrightarrow{a^{(k)}} q_b \in \text{Rules}(B) \end{array} \right\}$$

# Closure under Union

Given  $A$  and  $B$ , build  $A \cup B$

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

$$\left\{ ((q_a^1, q_b^1), \dots, (q_a^k, q_b^k)) \xrightarrow{a^{(k)}} (q_a, q_b) \mid \begin{array}{l} q_a^1, \dots, q_a^k \xrightarrow{a^{(k)}} q_a \in \text{Rules}(A) \\ q_b^1, \dots, q_b^k \xrightarrow{a^{(k)}} q_b \in \text{Rules}(B) \end{array} \right\}$$

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

# Closure under intersection

Given  $A$  and  $B$ , build  $A \cap B$

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

$$\left\{ ((q_a^1, q_b^1), \dots, (q_a^k, q_b^k)) \xrightarrow{a^{(k)}} (q_a, q_b) \mid \begin{array}{l} q_a^1, \dots, q_a^k \xrightarrow{a^{(k)}} q_a \in \text{Rules}(A) \\ q_b^1, \dots, q_b^k \xrightarrow{a^{(k)}} q_b \in \text{Rules}(B) \end{array} \right\}$$

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

# Complexity of the Product

## Size of the Result Automaton

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

Quadratic increase in size

# Closure under Negation: Completion

## Definition : Complete Tree Automaton

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

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

with some  $q$

## Intuition

At least one transition at a given node...

# Example

## Incomplete (deterministic) tree automaton

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

$$\epsilon \xrightarrow{b} q_b$$

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

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

## Completion of $A$

Add a sink state  $q_p$

$$\epsilon \xrightarrow{b} q_b$$

$$\epsilon \xrightarrow{a} q_p$$

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

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

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

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

$$(q_b, q_b) \xrightarrow{b} q_p$$

$$(q_b, q_a) \xrightarrow{b} q_p$$

$$(q_a, q_b) \xrightarrow{b} q_p$$

$$(q_a, q_a) \xrightarrow{b} q_p$$

$$(q_p, q) \xrightarrow{\sigma} q_p \quad \text{for all } q \in \{q_a, q_b, q_p\} \text{ and } \sigma \in \{a, b\}$$

$$(q, q_p) \xrightarrow{\sigma} q_p \quad \text{for all } q \in \{q_a, q_b, q_p\} \text{ and } \sigma \in \{a, b\}$$

$$(q) \xrightarrow{\sigma} q_p \quad \text{for all } q \in \{q_a, q_b, q_p\} \text{ and } \sigma \in \{a, b\}$$

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

# Example

## Incomplete (deterministic) tree automaton

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

$$\epsilon \xrightarrow{b} q_b$$

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

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

## Completion of $A$ , Complementation of $A$

Add a sink state  $q_p$

$$\epsilon \xrightarrow{b} q_b$$

$$\epsilon \xrightarrow{a} q_p$$

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

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

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

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

$$(q_b, q_b) \xrightarrow{b} q_p$$

$$(q_b, q_a) \xrightarrow{b} q_p$$

$$(q_a, q_b) \xrightarrow{b} q_p$$

$$(q_a, q_a) \xrightarrow{b} q_p$$

$$(q_p, q) \xrightarrow{\sigma} q_p \quad \text{for all } q \in \{q_a, q_b, q_p\} \text{ and } \sigma \in \{a, b\}$$

$$(q, q_p) \xrightarrow{\sigma} q_p \quad \text{for all } q \in \{q_a, q_b, q_p\} \text{ and } \sigma \in \{a, b\}$$

$$(q) \xrightarrow{\sigma} q_p \quad \text{for all } q \in \{q_a, q_b, q_p\} \text{ and } \sigma \in \{a, b\}$$

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



# Closure under Negation: Summary

## Building the Complement of $A$

- Make  $A$  deterministic
- Complete the result
- Switch final  $\leftrightarrow$  non-final states

## Complexity

- Determinization of  $A$  : exponential explosion (states:  $2^{\text{States}(A)}$ )
- Completion of the result: exponential explosion of the number of rules:  
 $|\text{Alphabet}(A)| \cdot \left(2^{\text{States}(A)}\right)^k$  where  $k$  is the maximal rank
- Switching final  $\leftrightarrow$  non-final states: linear

Total: exponential explosion

# Emptiness Test

Given a tree automaton  $A$ , is  $\text{Language}(A) \neq \emptyset$ ?

## Principle

Compute the set of reachable states and then see if any of them are in the final set

## Complexity

PTIME (time proportional to  $|A|$ )

# Application for Checking Type Inclusion

## Type Inclusion

Given two tree automata  $A_1$  and  $A_2$ , is  $\text{Language}(A_1) \subseteq \text{Language}(A_2)$  ?

## Theorem

Containment for non-deterministic tree automata can be decided in exponential time

## Principle

- $\text{Language}(A_1 \cap \overline{A_2}) \stackrel{?}{=} \emptyset$
  - For this purpose, we must make  $A_2$  deterministic (size:  $O(2^{|A_2|})$ )
- EXPTIME
- Essentially no better solution [Seidl, 1990]

# Application for Checking Type Inclusion

## Type Inclusion

Given two tree automata  $A_1$  and  $A_2$ , is  $\text{Language}(A_1) \subseteq \text{Language}(A_2)$  ?

## Theorem

Containment for non-deterministic tree automata can be decided in exponential time

## Principle

- $\text{Language}(A_1 \cap \overline{A_2}) \stackrel{?}{=} \emptyset$
- For this purpose, we must make  $A_2$  deterministic (size:  $O(2^{|A_2|})$ )
- EXPTIME
- Essentially no better solution [Seidl, 1990]

# Top-Down Tree Automata

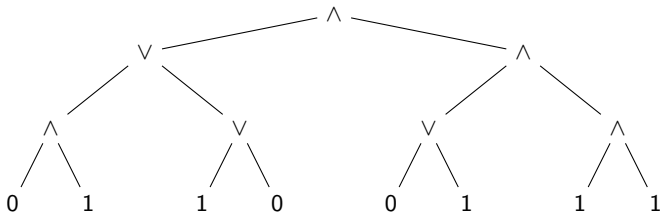
Is that useful?...

## Example: Connection with Strings

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

Reading strings from left to right = reading trees top-down (→ e.g. streaming validation...)

# Top-Down Tree Automata: Example



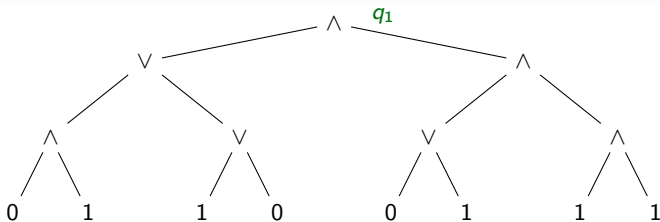
## Principle

- starting from the root, guess correct values
- check at leaves
- 3 states:  $q_0, q_1, acc$
- initial state at the root:  $q_1$
- accepting if all leaves labeled  $acc$

## Transitions

$$\begin{array}{ll} q_1 \xrightarrow{\wedge} (q_1, q_1) & q_1 \xrightarrow{\vee} (q_0, q_1) \\ q_0 \xrightarrow{\wedge} (q_0, q_1) & q_1 \xrightarrow{\vee} (q_1, q_0) \\ q_0 \xrightarrow{\wedge} (q_1, q_0) & q_1 \xrightarrow{\vee} (q_1, q_1) \\ q_0 \xrightarrow{\wedge} (q_0, q_0) & q_0 \xrightarrow{\vee} (q_0, q_0) \\ q_1 \xrightarrow{1} acc & q_0 \xrightarrow{0} acc \end{array}$$

# Top-Down Tree Automata: Example



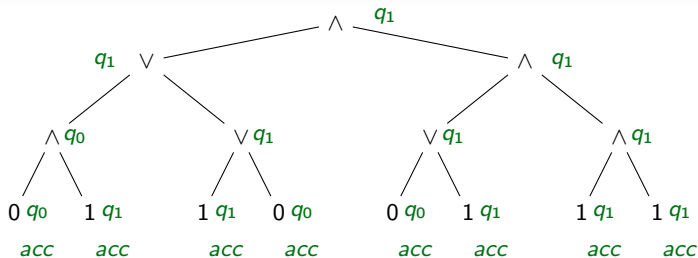
## Principle

- starting from the root, guess correct values
- check at leaves
- 3 states:  $q_0, q_1, acc$
- initial state at the root:  $q_1$
- accepting if all leaves labeled  $acc$

## Transitions

$$\begin{array}{ll} q_1 \xrightarrow{\wedge} (q_1, q_1) & q_1 \xrightarrow{\vee} (q_0, q_1) \\ q_0 \xrightarrow{\wedge} (q_0, q_1) & q_1 \xrightarrow{\vee} (q_1, q_0) \\ q_0 \xrightarrow{\wedge} (q_1, q_0) & q_1 \xrightarrow{\vee} (q_1, q_1) \\ q_0 \xrightarrow{\wedge} (q_0, q_0) & q_0 \xrightarrow{\vee} (q_0, q_0) \\ q_1 \xrightarrow{1} acc & q_0 \xrightarrow{0} acc \end{array}$$

# Top-Down Tree Automata: Example



## Principle

- starting from the root, guess correct values
- check at leaves
- 3 states:  $q_0, q_1, acc$
- initial state at the root:  $q_1$
- accepting if all leaves labeled  $acc$

## Transitions

$$\begin{array}{ll}
 q_1 \xrightarrow{\wedge} (q_1, q_1) & q_1 \xrightarrow{v} (q_0, q_1) \\
 q_0 \xrightarrow{\wedge} (q_0, q_1) & q_1 \xrightarrow{v} (q_1, q_0) \\
 q_0 \xrightarrow{\wedge} (q_1, q_0) & q_1 \xrightarrow{v} (q_1, q_1) \\
 q_0 \xrightarrow{\wedge} (q_0, q_0) & q_0 \xrightarrow{v} (q_0, q_0) \\
 q_1 \xrightarrow{1} acc & q_0 \xrightarrow{0} acc
 \end{array}$$



# Top-Down Tree Automata

A top-down tree automaton  $A$  consists in:

Alphabet( $A$ ): finite alphabet of symbols

States( $A$ ): finite set of states

Rules( $A$ ): finite set of transition rules

Initial( $A$ ): finite set of initial states ( $\subseteq$  States( $A$ ))

where:

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

Top-down tree automata also recognize all regular tree languages

# Top-Down Tree Automata

A top-down tree automaton  $A$  consists in:

Alphabet( $A$ ): finite alphabet of symbols

States( $A$ ): finite set of states

Rules( $A$ ): finite set of transition rules

Initial( $A$ ): finite set of initial states ( $\subseteq$  States( $A$ ))

where:

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

Top-down tree automata also recognize all regular tree languages

# Top-down Determinism

## Deterministic Top-Down Tree Automaton

- for each  $q \in \text{States}(A)$  and  $a \in \text{Alphabet}(A)$  there is at most one rule

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

- there is at most one initial state

# Top-down Determinism

## Deterministic Top-Down Tree Automaton

- for each  $q \in \text{States}(A)$  and  $a \in \text{Alphabet}(A)$  there is at most one rule

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

- there is at most one initial state

## Can We Make Top-Down Automata Deterministic?

- (a) Yes      (b) No      (c) Maybe...

# Can We Make Top-Down Automata Deterministic?

Maybe !

# Can We Make Top-Down Automata Deterministic?

Maybe !

Deterministic top-down tree automata do **not** recognize all regular tree languages

Example



# Can We Make Top-Down Automata Deterministic?

Maybe !

Deterministic top-down tree automata do **not** recognize all regular tree languages

Example



$\text{Initial}(A) = q_0$

# Can We Make Top-Down Automata Deterministic?

Maybe !

Deterministic top-down tree automata do **not** recognize all regular tree languages

Example



$\text{Initial}(A) = q_0$

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



# Can We Make Top-Down Automata Deterministic?

Maybe !

Deterministic top-down tree automata do **not** recognize all regular tree languages

Example



$\text{Initial}(A) = q_0$

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

$q \xrightarrow{b} \epsilon$

# Can We Make Top-Down Automata Deterministic?

Maybe !

Deterministic top-down tree automata do **not** recognize all regular tree languages

## Example



$\text{Initial}(A) = q_0$

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

$q \xrightarrow{b} \epsilon$

$q \xrightarrow{c} \epsilon$

# Can We Make Top-Down Automata Deterministic?

Maybe !

Deterministic top-down tree automata do **not** recognize all regular tree languages

## Example



$\text{Initial}(A) = q_0$

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

$q \xrightarrow{b} \epsilon$

$q \xrightarrow{c} \epsilon$

also accepts...



# Expressive Power of Tree Automata: Summary

## Theorem

The following properties are equivalent for a tree language  $L$ :

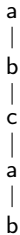
- (a)  $L$  is recognized by a bottom-up non-deterministic tree automaton
- (b)  $L$  is recognized by a bottom-up deterministic tree automaton
- (c)  $L$  is recognized by a top-down non-deterministic tree automaton
- (d)  $L$  is generated by a regular tree grammar

## Proof Idea

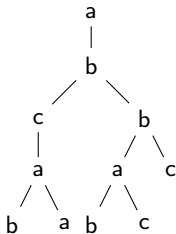
- (a)  $\Rightarrow$  (b): determinization (see [Comon et al., 1997])
- (a)  $\Leftrightarrow$  (c): same thing seen from 2 different ways
- (d)  $\Leftrightarrow$  (a) : ? (horizontal recursion  $a^*$ ?)

# Unranked Trees

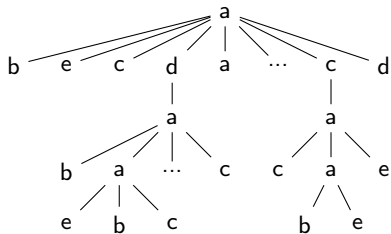
## String as Tree



## Ranked Tree



## Unranked Tree



## Unranked Tree Automata?

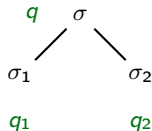
1. either we adapt ranked tree automata
2. or we encode unranked trees as ranked trees...

# Unranked Tree Automata

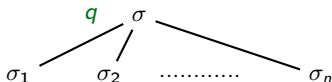
## Ranked Trees

Transitions can be described by finite sets:

$$\delta(\sigma, q) = \{(q_1, q_2), (q_3, q_4), \dots\}$$



## Unranked Trees



$q_1$	$q_2$	.....	$q_n$
-------	-------	-------	-------

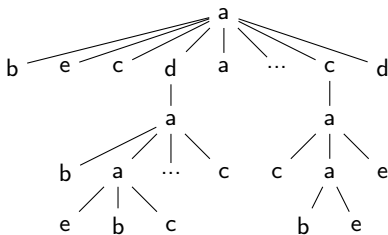
 $\in \delta(\sigma, q)?$

## $\delta(\sigma, q)$

- For unranked trees,  $\delta(\sigma, q)$  is a regular tree language
- $\delta(\sigma, q)$  may be specified by a regular expression or by a finite word automaton [Murata, 1999]

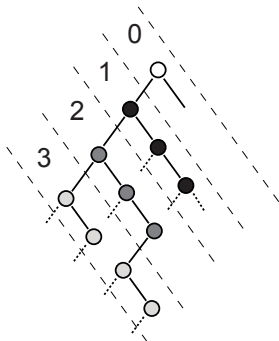
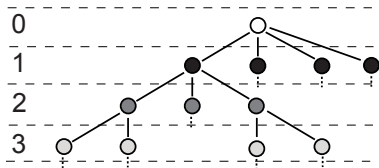
## Second Option

Can we encode unranked trees as ranked trees?



?

# Encoding Unranked Trees As Binary Trees



## Bijection Encoding

- “first child; next sibling” encoding
- Allows to focus on binary trees without loss of generality
- Results for ranked trees hold for unranked trees as well



# Tree Automata: Summary

## Definition

A tree language is regular iff it is recognized by a non-deterministic tree automaton

## Advantages

- Closure, decidable operations
- General tool (theoretical and algorithmic)

## Limitations

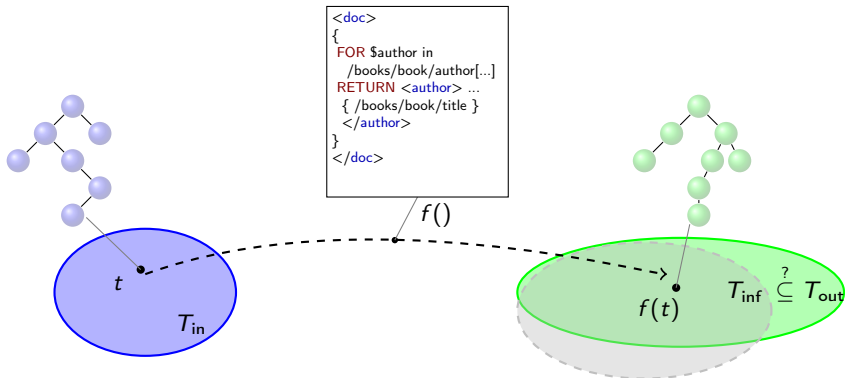
- $a^n b^n$

# Application for Static Type-Checking

## Recall: The XML Static Type-Checking Problem

- Given the following:
  - Source code of some program  $f$  (in e.g. XQuery)
  - A type  $T_{\text{in}}$  for **input** documents
  - A type  $T_{\text{out}}$  for **expected** output documents
- Problem: Does  $f(t) \in T_{\text{out}}$  for all  $t \in T_{\text{in}}$ ?

# Static Type-Checking for XQuery: Principles



Approach in two steps:

1. Compute  $T_{inf} = \{f(t) | t \in T_{in}\}$
2. Check whether  $T_{inf} \subseteq T_{out}$  holds

→ In case  $T_{inf} \subseteq T_{out}$  holds, then we know that for any  $t \in T_{in}$ ,  $f(t) \in T_{out}$

# Static Type-Checking Using Tree Automata

- Many research papers
- XQuery's standard type system inspired from advances in PLs, e.g.:
  - XDuce [Hosoya and Pierce, 2003]: a domain specific language
  - Sound static type system based on an implementation of tree automata containment optimized for practical cases
  - Use pattern-matching for selection (no support for XPath)

## XQuery's standard type system

- As defined in the spec:
    - Sound
    - PTIME
- Sound and more precise type systems in EXPTIME

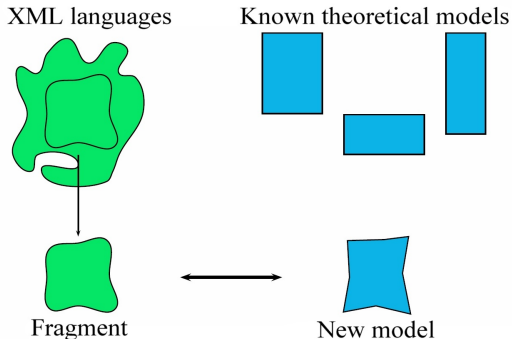
# XQuery and Static Type-Checking

## Open Issues

- Type inference requires a balance between precision and complexity
  - Sound approximations can be more or less imprecise
  - Poor precision (e.g. type approximation) in type inference yields more false alarms
  - XQuery's standard type system is limited:
    - Currently: no or poor support of XPath (e.g. only `child` and `descendant` axes are – poorly – supported)
    - Support for typing XPath has been open for years (in particular for backward axes)
- Can you guess why is that difficult?
- 2015 ICFP paper with elements of answers for this issue (based on advances in tree logics)

# Concluding Remarks on Tree Automata

- Tree automata are part of the theoretical tools that provide the underlying guiding principles for XML (like the relational algebra provides the underlying principles for relational databases)
- Important research challenges still remain



## For those who want to learn more

Pointers to related research results:

- [1] XQuery and Static Typing: Tackling the Problem of Backward Axes.

Pierre Genevès and Nils Gesbert.

In ICFP'15: Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP), Sept. 2015.

<http://tyrex.inria.fr/publications/icfp15.pdf>



Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. (1997).

Tree automata techniques and applications.

Available on: <http://www.grappa.univ-lille3.fr/tata>.  
release October, 1st 2002.



Doner, J. (1970).

Tree acceptors and some of their applications.

*Journal of Computer and System Sciences*, 4:406–451.



Hosoya, H. and Pierce, B. C. (2003).

XDuce: A statically typed XML processing language.

*ACM Trans. Inter. Tech.*, 3(2):117–148.



Murata, M. (1999).

Hedge automata: a formal model for XML schemata.

[http://www.xml.gr.jp/relax/hedge\\_nice.html](http://www.xml.gr.jp/relax/hedge_nice.html).



Seidl, H. (1990).

Deciding equivalence of finite tree automata.

*SIAM J. Comput.*, 19(3):424–437.



Thatcher, J. W. and Wright, J. B. (1968).



Generalized finite automata theory with an application to a decision problem of second-order logic.

*Mathematical Systems Theory*, 2(1):57–81.