

UNIVERSITY OF BAYREUTH

BACHELOR SEMINAR TREE AUTOMATA

---

# Introduction to Ranked Tree Automata

---

*Author:*  
Martin BRAUN

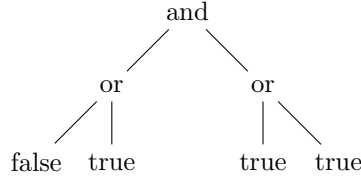
*Supervisor:*  
Prof. Dr. Wim MARTENS

# Introduction to Tree Languages

A good example for a tree language is the one consisting of boolean expressions for which an instance - if formatted in the right way - could look like this:

$and(or(false, true), or(true, true))$

In order to ease understanding, the elements of the language are often represented as a tree in a graphical way:



Just like for "normal" regular languages, it is of interest to know whether a given word (in this case a tree) is part of the (tree-)language. In order to describe an automaton that recognizes tree-languages we have to define what  $\Sigma$ -trees and (regular) tree-languages are, first.

**Definition 1.**  $\Sigma$ -tree [2]

The set of  $\Sigma$ -trees  $T_\Sigma$  over the alphabet  $\Sigma$  is inductively defined as follows:

1. every  $\sigma \in \Sigma$  is a  $\Sigma$ -tree
2.  $\sigma \in T_\Sigma$  and  $t_1, \dots, t_n \in T_\Sigma, n \geq 1 \iff \sigma(t_1, \dots, t_n) \in T_\Sigma$

*Note: Normally, there is no bound for the number of children in a tree (these trees are called unranked), but in this draft we will only take a look at **ranked trees**, which have such a bound.*

**Definition 2.** tree-language [2]

A tree language  $L_{t\Sigma}$  over the alphabet  $\Sigma$  is defined as a subset of  $T_\Sigma$ :

$$L_{t\Sigma} \subseteq T_\Sigma$$

$\Rightarrow T_\Sigma$  is already a tree-language.

We can now define (Non-Deterministic) Finite Tree Automata for tree languages.

*Note: this definition will be expanded with more terms later in this draft and some of the content in this definition will get a specific name assigned to them. But in order to not overcomplicate the definition, these parts are left out for now.*

**Definition 3.** *NFTA [3]*

A (Non-Deterministic) Finite Tree Automaton (NFTA) over the **alphabet**  $\Sigma$  is a tuple  $A = (Q, \Sigma, Q_f, \Delta)$  where  $Q$  is a **finite set of states**,  $Q_f \subseteq Q$  is a **finite set of final states**, and  $\Delta$  is a **finite set of transition rules** of the type:

$$f(q_1, \dots, q_n) \rightarrow q_x$$

where  $n \geq 0, f \in \Sigma, q_x, q_1, \dots, q_n \in Q$

All  $f \in \Sigma$  with  $n = 0$  are called **ground-terms** over  $\Sigma$ . We write these as:

$$a \rightarrow q(a)$$

where  $a \in \Sigma, q \in Q$

Tree automata over  $\Sigma$  run on ground terms over  $\Sigma$ . An automaton starts at the leaves and moves upward, associating along a **run** a state with each subterm inductively while reducing the tree via the transition rules.

For a tree  $t' \in T_{\Sigma \cup Q}$  that is the result of applying a transition rule on a tree  $t \in T_{\Sigma \cup Q}$  we write:

$$t \rightarrow_A t'$$

If more or equal than one transition rules are applied we denote it like this:

$$t \rightarrow_A^* t'$$

$\rightarrow_A^*$  is the reflexive and transitive closure of  $\rightarrow_A$ .

*Note: There is no initial state in an NFTA but the ground-terms (which can be considered to be the "initial rules" of the NFTA) act alike by transitioning constant symbols into a state.*

Our binary-boolean-expression NFTA can now be written as:

*Example 1.* binary-boolean-statement NFTA

$A = (Q, \Sigma, Q_f, \Delta)$

$\Sigma = \{or, and, not, true, false\}$

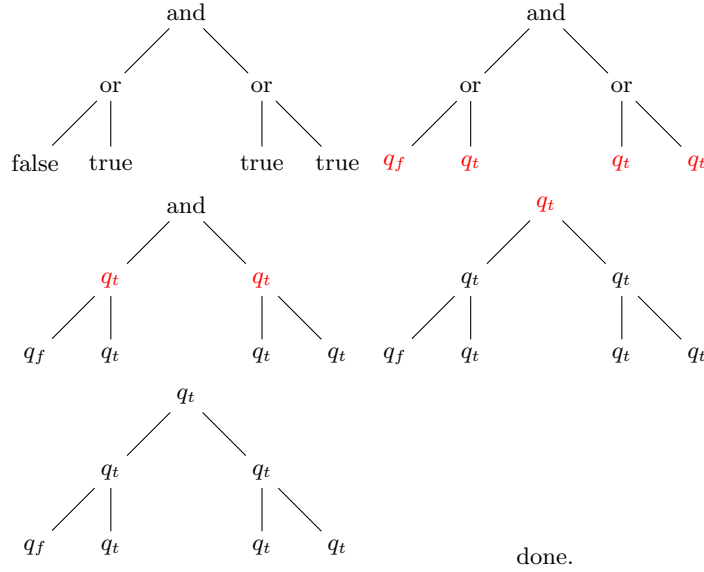
$Q = \{q_f, q_t\}$

$Q_f = \{q_t\}$

$\Delta = \{false \rightarrow q_f, true \rightarrow q_t,$   
 $and(q_t, q_t) \rightarrow q_t, and(q_t, q_f) \rightarrow q_f, and(q_f, q_t) \rightarrow q_f, and(q_f, q_f) \rightarrow q_f,$   
 $or(q_t, q_t) \rightarrow q_t, or(q_t, q_f) \rightarrow q_t, or(q_f, q_t) \rightarrow q_t, or(q_f, q_f) \rightarrow q_f,$   
 $not(q_f) \rightarrow q_t, not(q_t) \rightarrow q_f\}$

A run with the example input from the beginning of this chapter looks like this:

*Example 2.* running a NFTA



Or in written form:

$and(or(false, true), or(true, true)) \rightarrow_A^* and(or(q_f, q_t), or(q_t, q_t))$

$\rightarrow_A^* and(q_t, q_t) \rightarrow_A^* q_t$

$q_t \in Q_f \Rightarrow A$  accepts  $w \Rightarrow w \in L_A$  with  $L_A$  being the language recognized by the automaton.

## Determinization

Our definition of Finite Tree Automata (FTA) is based on Non Deterministic Tree Automata (NFTA) and we will now take a look at how we can determinize these automata as we need our automata to be deterministic later in this draft. But first we have to define what this means in the context of FTAs.

**Definition 4.** *Deterministic Finite Tree Automaton*

*A tree automaton with no two rule of the type:*

$$\begin{aligned} f(q_1, \dots, q_n) &\rightarrow q_x \\ f(q_1, \dots, q_n) &\rightarrow q_y \end{aligned}$$

(Note: this includes the ground-terms)

or

$$\epsilon(q_1, \dots, q_n) \rightarrow q_x$$

with  $n \geq 0, q_x, q_y, q_1, \dots, q_n \in Q, q_x \neq q_y, a \in \Sigma$  is called a **Deterministic Finite Tree Automaton (DFTA)**.

Similar to the algorithm for Determinization in the word case, there exists a power set construction algorithm for determinizing Tree Automata.

**Definition 5.** *Algorithm DET for Tree Automata [3]*

**Data:** NFTA  $A = (Q, \Sigma, Q_f, \Delta)$

$Q_d := \emptyset$

$\Delta_d := \emptyset$

**while**  $\Delta_d$  grew last cycle **do**

$f(q_1, \dots, q_n) \in \Delta$

$s_1, \dots, s_n \in Q_d$

/\* meta-state representing the set of reachable states \*/

$s := \text{state}(\{q \in Q \mid \exists q_1 \in s_1, \dots, q_n \in s_n, f(q_1, \dots, q_n) \rightarrow q \in \Delta\})$

$Q_d := Q_d \cup \{s\}$

$\Delta_d := \Delta_d \cup f(s_1, \dots, s_n) \rightarrow s$

**end**

$Q_{f_d} := \{s \in Q_d \mid \{s\} \cap Q_d \neq \emptyset\}$

**Result:** DFTA  $A_d = (Q_d, \Sigma, Q_{f_d}, \Delta_d)$

It is easy to see that the algorithm produces a deterministic automaton  $A_d$  as we are automatically constructing meta-states for all reachable states and therefor eliminating all possible non-deterministic behaviour.

However we still have to prove that  $L(A) = L(A_d)$ .  
For this, we have to prove that the meta-states  $s \in Q_d$  are built correctly or in more mathematical terms:

$$\text{For any tree } t : t \rightarrow_{A_d}^* s \iff s = \text{state}(\{q \in Q \mid t \rightarrow_A^* q\})$$

*Proof.*  $L(A) = L(A_d)$  (Correctness of DET) [3]

This proof is done via an induction over the structure of terms.

– **Base case:** For any tree  $t = a \in \Sigma$  we take a look at the corresponding ground-term  $a \rightarrow q(a)$ . Because of the way we defined  $s$  as the meta-state representing the set of all reachable states in a given situation this is inherently correct.

– **induction step:**  $t = f(q_1, \dots, q_n)$   
• 1.:  $t \rightarrow_{A_d}^* s \Rightarrow (s = \text{state}(\{q \in Q \mid t \rightarrow_A^* q\}))$

Supposing  $t \rightarrow_{A_d}^* f(s_1, \dots, s_n) \rightarrow_{A_d} s$ , by induction hypothesis, for each  $i \in 1, \dots, n$ , we can see that  $s_i = \text{state}(\{q \in Q \mid q_i \rightarrow_A^* q\})$ .

Since states  $s_i \in Q_d$ , rules  $f(s_1, \dots, s_n) \rightarrow s \in \Delta_d$  are added by the determinization algorithm and  $s := \text{state}(\{q \in Q \mid \exists q_1 \in s_1, \dots, q_n \in s_n, f(q_1, \dots, q_n) \rightarrow q \in \Delta\})$ , we can see that  $s = \text{state}(\{q \in Q \mid t \rightarrow_A^* q\})$ .

• 2.:  $s = \text{state}(\{q \in Q \mid t \rightarrow_A^* q\}) \Rightarrow t \rightarrow_{A_d}^* s$

Considering  $s = \text{state}(\{q \in Q \mid f(q_1, \dots, q_n) \rightarrow_A^* q\})$  with state sets  $S_i$  defined as  $S_i := \{q \in Q \mid q_i \rightarrow_A^* q\}$ , by induction hypothesis for each  $i \in \{1, \dots, n\}$  we know  $q_i \rightarrow_{A_d}^* s_i = \text{state}(S_i)$ . Thus  $s := \text{state}(\{q \in Q \mid \exists q_1 \in s_1, \dots, q_n \in s_n, f(q_1, \dots, q_n) \rightarrow q \in \Delta\})$ .

By the definition of  $\Delta_d$  in the determinization algorithm,  $f(s_1, \dots, s_n) \in \Delta_d$  and thus  $t \rightarrow_{A_d}^* s$ .

*Example 3.* Running the DET algorithm  
consider this NFTA that accepts unordered lists (HTML-style):

$A = (Q, \Sigma, Q_f, \Delta)$   
 $\Sigma = \{ul, li, text, empty\}$   
 $Q = \{q_{ul}, q_{li1}, q_{li2}, q_{text}, q_{empty}\}$   
 $Q_f = \{q_{ul}\}$   
 $\Delta = \{ul(q_{li1}, q_{li2}) \rightarrow q_{ul}, ul(q_{li2}, q_{li1}) \rightarrow q_{ul},$   
 $li(q_{text}, q_{text}) \rightarrow q_{li1}, li(q_{text}, q_{text}) \rightarrow q_{li2}$   
 $text \rightarrow q_{text}, empty \rightarrow q_{empty},$   
 $\epsilon(q_{empty}) \rightarrow q_{text}\}$

## References

1. Wikipedia on XML: [http://de.wikipedia.org/wiki/Extensible\\_Markup\\_Language](http://de.wikipedia.org/wiki/Extensible_Markup_Language)
2. Automata theory for XML researchers Frank Neven University of Limburg frank.neven@luc.ac.be, <http://homepages.inf.ed.ac.uk/libkin/dbtheory/frank.pdf>, 11.03.2015
3. Tree Automata and Techniques, Hubert Comon et. al, Pages 19-39