University of Bayreuth

Bachelor Seminar Tree Automata

Introduction to Ranked Tree Automata

Author:
Martin Braun

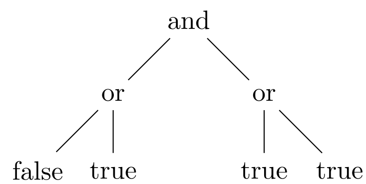
Supervisor:
Prof. Dr. Wim Martens

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 **Σ-trees** and (regular) **tree-languages** are, first.

Definition 1. *Σ-tree [1]*
The set of Σ-trees T_Σ over the alphabet Σ is inductively defined as follows:

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

*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 [1]*
A tree language $L_{t\Sigma}$ over the alphabet Σ is defined as a subset of T_Σ:

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

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

Next, we have to declare some special words in the context of $\Sigma - trees$.

Definition 3. *variables, linear terms, ground-terms [2][3]*
*Let $v \notin \Sigma = \emptyset$ be a constant (symbol with no child). We call v a **variable** in a term over the ranked alphabet Σ if it is a placeholder for any given $\sigma \in \Sigma$ in that term. Terms containing every v at most once are **linear terms**. All terms that don't contain any variables, are called **ground-terms** over Σ.*

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 4. *NFTA [2]*
*A (Non-Deterministic) Finite Tree Automaton (NFTA) over the **alphabet** Σ 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 Δ is a **finite set of transition rules** of the type:*

$$f(q_1, ..., q_n) \to q_x$$
$$\text{where } n \geq 0, f \in \Sigma, q_x, q_1, ..., q_n \in Q$$

For $n = 0$, we write:

$$a \to q(a)$$
$$\text{where } a \in \Sigma, q \in Q$$

*Tree automata over Σ run on ground terms over Σ. 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:

$$t \to_A t'$$

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

$$t \to_A^* t'$$

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

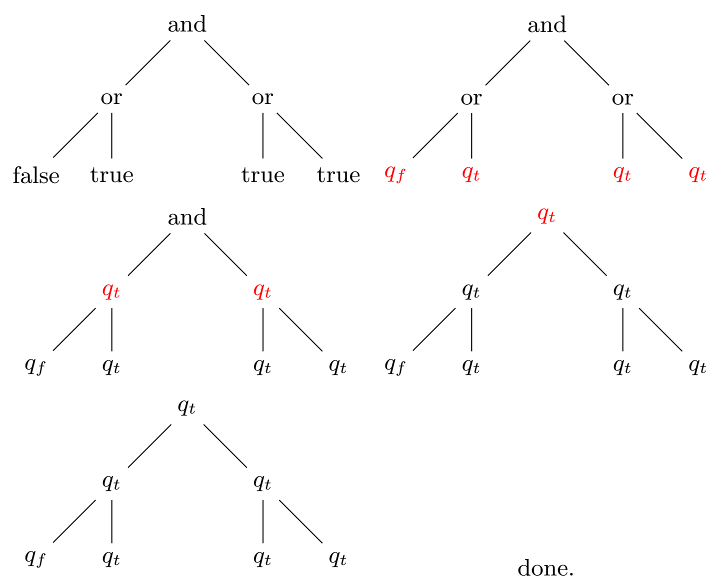
Note: There is no initial state in an NFTA but the ground-terms (which can be considered to be the "initial rules" of the NFTA) act alike by transitioning constant symbols into a state.

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

Example 1. binary-boolean-statement NFTA
$A = (Q, \Sigma, Q_f, \Delta)$
$\Sigma = \{or, and, not, true, false\}$
$Q = \{q_f, q_t\}$
$Q_f = \{q_t\}$
$\Delta = \{false \to q_f, true \to q_t,$
$\quad and(q_t, q_t) \to q_t, and(q_t, q_f) \to q_f, and(q_f, q_t) \to q_f, and(q_f, q_f) \to q_f,$
$\quad or(q_t, q_t) \to q_t, or(q_t, q_f) \to q_t, or(q_f, q_t) \to q_t, or(q_f, q_f) \to q_f,$
$\quad not(q_f) \to q_t, not(q_t) \to q_f\}$

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

Example 2. running a NFTA



Or in written form:
$and(or(false, true), or(true, true)) \to_A^* and(or(q_f, q_t), or(q_t, q_t))$
$\to_A^* and(q_t, q_t) \to_A^* q_t$

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

Determinization

Non Deterministic Finite Tree Automata (NFTA) can be determinized just like Non Determinitistic Automata (NFA) in the word case. By knowing that there exists a DFTA for every NFTA, some algorithms become easier to write and proof. We will now take a look at how this is done. But first we have to define formally, what being deterministic means in the context of FTAs.

Definition 5. *Deterministic Finite Tree Automaton*

A tree automaton with no two rule of the type:

$$f(q_1, ..., q_n) \to q_x$$
$$f(q_1, ..., q_n) \to q_y$$

(Note: this includes the ground-terms)

or

$$\epsilon(q_1, ..., q_n) \to q_x$$

with $n \geq 0, q_x, q_y, q_1, ...q_n \in Q, q_x \neq q_y, a \in \Sigma$ *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 determizing Tree Automata.

Definition 6. ***Algorithm DET*** *for Tree Automata [2]*

Data*: NFTA* $A = (Q, \Sigma, Q_f, \Delta)$
$Q_d := \emptyset$
$\Delta_d := \emptyset$
while Δ_d *grew last cycle* **do**
  $f(q_1, ..., q_n) \in \Delta$
  $s_1, ..., s_n \in Q_d$
  /* *meta-state representing the set of reachable states* */
  $s := state(\{q \in Q \mid \exists q_1 \in s_1, ..., q_n \in s_n, f(q_1, ...q_n) \to q \in \Delta\})$

  $Q_d := Q_d \cup \{s\}$
  $\Delta_d := \Delta_d \cup f(s_1, ..., s_n) \to s$
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 show that the meta-states $s \in Q_d$ are "built correctly", or in formal terms:

$$\textit{For any tree } t : t \rightarrow^*_{A_d} s \iff s = \textit{state}(\{q \in Q \mid t \rightarrow^*_A q\})$$

Proof. $L(A) = L(A_d)$ (Correctness of DET) [2]
This proof is done via an induction over the structure of the symbols in Σ.

– **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:** $t = f(q_1, ..., q_n)$

  - 1.: $t \rightarrow^*_{A_d} s \Rightarrow (s = \textit{state}(\{q \in Q \mid t \rightarrow^*_A q\})$

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

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

  - 2.: $s = \textit{state}(\{q \in Q \mid t \rightarrow^*_A q\}) \Rightarrow t \rightarrow^*_{A_d} s$

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

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

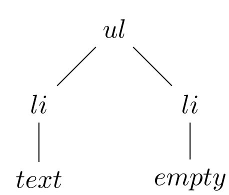
Following is an example of how a NFTA can be determinized with this algorithm.

Example 3. Running the DET algorithm
consider an obviously non deterministic FTA given like this:
$A = (Q, \Sigma, Q_f, \Delta)$
$\Sigma = \{ul, li, text, empty\}$
$Q = \{q_{ul}, q_{li1}, q_{li2}, q_{text}, q_{empty}\}$
$Q_f = \{q_{ul}\}$
$\Delta = \{ul(q_{li1}, q_{li2}) \to q_{ul}, ul(q_{li2}, q_{li1}) \to q_{ul},$
$li(q_{text}, q_{text}) \to q_{li1}, li(q_{text}, q_{text}) \to q_{li2}$
$text \to q_{text}, empty \to q_{empty},$
$\epsilon(q_{empty}) \to q_{text}\}$

This recognizes all trees that represent unordered lists (ul) in HTML notation that contain 2 list items (li) that are either empty or contain text:

```
<ul>
    <li>text</li>
    <li>empty</li>
</ul>
```

Or as a tree input:



If we start determinizing with the ground terms and then go "up in the hierarchy", we get these new rules and states:

$text \to state(\{q_{text}\})$
$empty \to state(\{q_{text}, q_{empty}\})$
$li(state(\{q_{text}\}), state(\{q_{text}\})) \to state(\{q_{li1}, q_{li2}\})$
$li(state(\{q_{text}, q_{empty}\}), state(\{q_{text}, q_{empty}\})) \to state(\{q_{li1}, q_{li2}\})$
$ul(state(\{q_{li1}, q_{li2}\}), state(\{q_{li1}, q_{li2}\})) \to state(\{q_{ul}\})$

And the set of final states is $Q_{f_d} = \{state(\{q_{ul}\})\}$.

Minimization

Definition 7. *Context*
A context $C \in C(\Sigma)$ *is a tree*

References

1. 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, 03/11/2015
2. Tree Automata and Techniques, Hubert Comon et. al, Pages 19-39
3. http://en.wikipedia.org/wiki/Ground_expression, 03/16/2015