Hedge automata and XML schemes

Sebastian Muskalla

June 26, 2017

This document is based on Prof. Roland Meyer's handwritten notes on the topic.
They are available here:

- tcs.cs.tu-bs.de/documents/AutomataTheory_WS_2013/lecturenotes/30_xml_part_1.pdf
- tcs.cs.tu-bs.de/documents/AutomataTheory_WS_2013/lecturenotes/31_xml_part_2.pdf

TU Braunschweig

Summer term 2017

1. Unranked trees

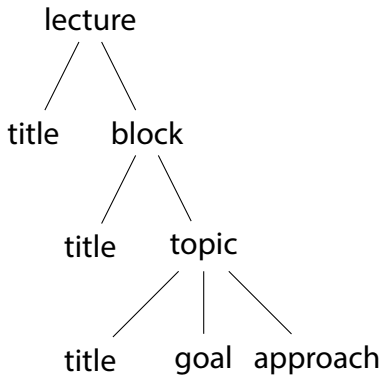
So far, we have only considered **ranked** trees, trees, in which the number of successors of each node is determined by its label. This means we only consider bounded branching: The maximum out-degree of a node in the tree is bounded by the greatest rank of a symbol in the alphabet.

In this lecture, we want to consider so-called **unranked** trees, in which each node can have an arbitrary number of successors.

Consider for example an XML document with the following content.

```
<lecture>
    <title>Algorithmic automata theory</title>
    <block>
        <title>Finite words</title>
        <topic>
            <title>WSMO</title>
            <goal>Satisfiablity</goal>
            <approach>Büchi</approach>
        </topic>
    </block>
</lecture>
```

We can see its structure (i.e. the document without the data) as a tree.



This is an unranked tree: A lecture could consist of an arbitrary number of blocks, a block could consist of an arbitrary number of topics.

2. XML specifications

We want to pose requirements on the structure of XML documents.

In the example, we want to specify that each lecture consists of blocks, and each block consists of topics.

Such requirements specify a language of unranked trees whose nodes are labeled by the **tags** (lecture, title, block, …) of the document. The description of the specification is called **scheme**.

A document is **valid** with respect to a given scheme if it satisfies the specification given by the scheme.

We are interested in the connections to automata theory:

- **Validity**: Is a given document valid wrt. a given scheme?
  ↳ **Membership**: Is the tree representing the structure of the document in the tree language specified by the scheme?
- Is there a document valid for a given scheme? (Sanity check)
  ↳ **Emptiness** of the tree language specified by the scheme?
- Are all documents that are valid wrt. one schema also valid wrt. another scheme? (Needed when merging archives/companies.)
  ↳ **Inclusion** of the tree languages.

There are several standards for XML schemes. Commonly used in practice are nowadays e.g. RELAX NG and XSD. We will consider the older standard **document type definition (DTD)** here.

A **document type definition (DTD)** is essentially a context-free grammar with regular expressions on the right-hand side. Its tree language is the set of derivation trees of terminal words produced by the DTD.

The document type definition corresponding to our example could look as follows.

```
<!DOCTYPE LECTURE [
    <!ELEMENT lecture (title, (block+ | (topic,exercise?)+) ) >
    <!ELEMENT block   (title, (topic,exercise?)+ ) >
    <!ELEMENT topic   (title, goal, problem?, approach) >
    <!ELEMENT title   (#PCDATA) ) >
    ...
]>
```

The right-hand side of a DTD is called **content model**. Here, we use `|` for choice, `,` for sequences (concatenation), `+` for one or more occurrences, and `?` wfor zero or one occurrence. `#PCDATA` represents an arbitrary character sequence, i.e. data in the document.

We can represent a DTD as a context-free grammar. In the example, we obtain the following CFG:

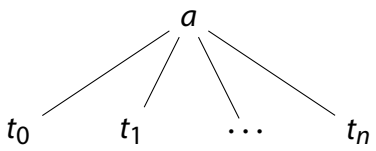
$$\begin{aligned} \text{lecture} &\to \text{title.(block}^+ + \text{(topic.(exercise} + \varepsilon))^+) \\ \text{block} &\to \text{title.(topic.(exercise} + \varepsilon))^+ \\ \text{topic} &\to \text{title.goal.(problem} + \varepsilon).\text{approach} \\ \text{title} &\to \varepsilon \\ &\dots \end{aligned}$$

To formally define the language of a DTD, we need **hedge automata**

3. Hedge automata

Let Σ be an alphabet. We will call it **unranked alphabet** to emphasize that we drop the restrictions from the previous lectures.

We consider Σ-labeled trees $t\colon \mathcal{T} \to \Sigma$. Each node $v \in \mathcal{T}$ can have arbitrarily many, but finitely many children. This means we allow unbounded branching.



We call the subtrees $t_0, \dots, t_n$ a **hedge**.

A hedge automaton will process such a tree bottom-up. Our goal is to obtain similar properties as for ranked trees. The problem is that since the number of children is not fixed, we can not list all possible transitions explicitly. Instead, we will represent an infinite set of transitions symbolically.

3.1 Example
Consider the language of all trees

- of height 2
- with the root labeled by *a*,
- the children labeled by *b*, and
- an even number of children.

A bottom-up automaton should have two states q, q', a transition $\to_b q'$ for the leaves, and transitions $\to_a q, (q', q') \to_a q, (q', q', q', q') \to_a q, \ldots$.

The infinite set of transitions can be represented by

$$(q'q')^* \to_a q\,.$$

More generally, we want to allow transitions of the shape

$$R \to_a q\,,$$

where $R \subseteq Q^*$ is a regular language over the states of the automaton.

3.2 Definition
A **(non-deterministic) hedge automaton (NHA)** A is a tuple

$$A = (\Sigma, Q, \to, Q_F)\,,$$

where

- Σ is a unranked alphabet,
- Q is a finite set of control states,
- $Q_F \subseteq Q$ is the set of final states, and
- $\to \subseteq \mathcal{P}\big(Q^*\big) \times \Sigma \times Q$ is the transition relation.

The left hand sides $R \subseteq Q^*$ of transitions $R \to_a q \in$ are required to be regular, and $\to$ should be finite. The R are called **horizontal languages**.

A **run** of the NHA A over the tree $t\colon \mathcal{T} \to \Sigma$ is a function

$$r\colon \mathcal{T} \to Q$$

so that for all nodes $v \in \mathcal{T}$ with $t(v) = a$, $r(v) = q$, there is a transition $R \to_a q$ such that

$$r(v.0)r(v.1)\ldots r(v.(n\text{ - }1)) \in R\,,$$

where n is the number of successors of v.

We are only allowed to apply a transition $R \to_a q$ to a leaf if $\varepsilon \in R$.

A run is **accepting** if $r(\varepsilon) \in Q_F$.

The language of A is the set of trees that have an accepting run,

$$\mathcal{L}(A) = \{t\colon \mathcal{T} \to \Sigma \mid \text{ there is an accepting run of } A \text{ on } t\}\,.$$

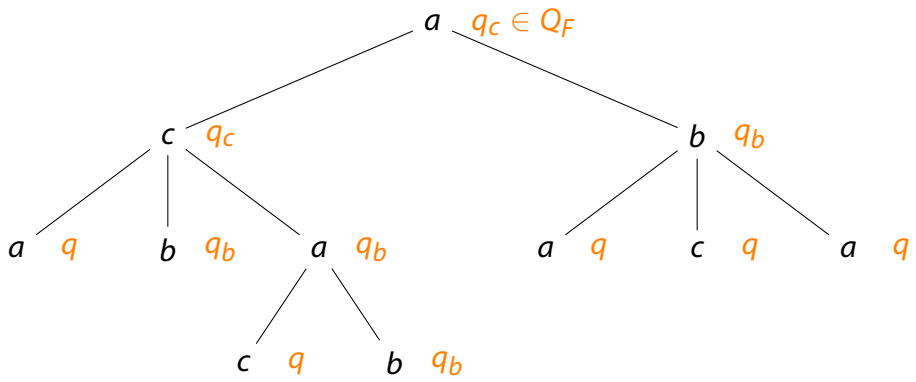
3.3 Example

Consider the alphabet $\Sigma = a, b, c$ and the language of Σ-labeled trees defined by the following condition:
There are two nodes labeled by b whose greatest common ancestor is labeled by c.

This language is regular, it is the language of the NHA $A = (\Sigma, Q, \to, Q_f)$ with

$$\begin{aligned} Q &= \{q, q_a, q_c\} \\ Q_F &= \{q_c\} \\ Q^* &\to_{a/c} q \\ Q^* &\to_{b} q_b \\ Q^* q_b Q^* &\to_{a/c} q_b \\ Q^* q_b Q^* q_b Q^* &\to_{c} q_c \\ Q^* q_c Q^* &\to_{a/b/c} q_c \end{aligned}$$

The automaton works as follows: It assigns q_b to b-labeled nodes and and propagates this states upwards. As soon as a c occurs that has two q_bs as children, it goes to the final state, and propagates it to the root.



4. DTD revisited

4.1 Definition

A **document type definition (DTD)** is a tuple $D = (\Sigma, s, \delta)$, where

- Σ is a finite set of tags,
- $s \in \Sigma$ is the root tag, and
- δ is a function assigning each symbol $a \in \Sigma$ a regular expression $\delta(a)$ over Σ.

Given a DTD D, we define the corresponding hedge automaton $A_D = (\Sigma, Q, \rightarrow, Q_F)$ where

- $Q = \{q_a \mid a \in \Sigma\}$,
- $Q_F = \{q_s\}$, and
- for each symbol $a \in \Sigma$, there is a unique transition

$$\mathcal{L}\big(\delta(a)\big) \rightarrow_a q_a \,.$$

The language of D is defined to be the language of A_D, $\mathcal{L}\big(D\big) = \mathcal{L}\big(A_D\big)$.

Note that in the expression $\mathcal{L}\big(\delta(a)\big)$, we implicitly assume that we convert tags a to their corresponding state q_a.

4.2 Example

Consider the transition

$$\text{lecture} \rightarrow \text{title}.(\text{block}^+ + (\text{topic}.(\text{exercise} + \varepsilon))^+)$$

from our running example.

The corresponding NHA is

$$A_D = (\Sigma, \{q_{lec}, q_{tit}, q_{block}, q_{top}, q_{ex}, \ldots\}, \{q_{lec}\}, \rightarrow)$$

with the transition

$$q_{tit}.(q_{block}^+ + (q_{top}.(q_{ex} + \varepsilon))^+) \rightarrow_{\text{lecture}} q_{lec}$$

Now we could check that our initial example document is indeed valid. There is a run that assigns to each node labeled by tag a the state q_a. In particular, it assigns to the root node labeled by lecture the state $q_{lec} \in Q_F$.

How to do a validity check (membership check) algorithmically?

Validity checking against DTD
Given: DTD D, XML document *doc*
Decide: Is *doc* valid wrt. D?

By the previous discussion, this corresponds to membership checking for NHAs.

5. Membership checking for NHAs

The goal is to provide a fast decision procedure for the following problem.

Membership checking for NHAs
Given: NHA A, tree t
Decide: $t \in \mathcal{L}(A)$?

Here, we assume that the regular horizontal languages are given by non-deterministic finite automat (NFAs).

5.1 Theorem
Membership for NHAs with horizontal languages given by NFAs can be decided in polynomial time (in the total size of the input NHA and the NFAs for the regular languages and the tree).

Towards a proof, we construct an algorithms.

Assume $A = (\Sigma, Q, \rightarrow, Q_F)$ is the given NHA, and $t\colon \mathcal{T} \rightarrow \Sigma$ is the given tree. We construct a mapping

$$\rho\colon \mathcal{T} \rightarrow \mathcal{P}(Q)$$

that assigns each node set of possible states. Namely, it maps each node v to the set of states $\rho(v) \subseteq$ that are **reachable at** v in a run of A on t.

To this end, in each step the algorithms picks a node v whose children $v.0, \ldots, v.(n\text{ - }1)$ are already mapped to $Q_0, \ldots, Q_{n\text{-}1}$. For each transition $R \rightarrow_a q$, where $a = t(v)$ is the label of the node, it checks whether the transition can be applied to a string $q_0 \ldots q_{n\text{-}1}$ with $q_i \in Q_i$ for each i.

This means the algorithm implements a bottom-up on-the-fly powerset construction along the tree.

5.2 Algorithm

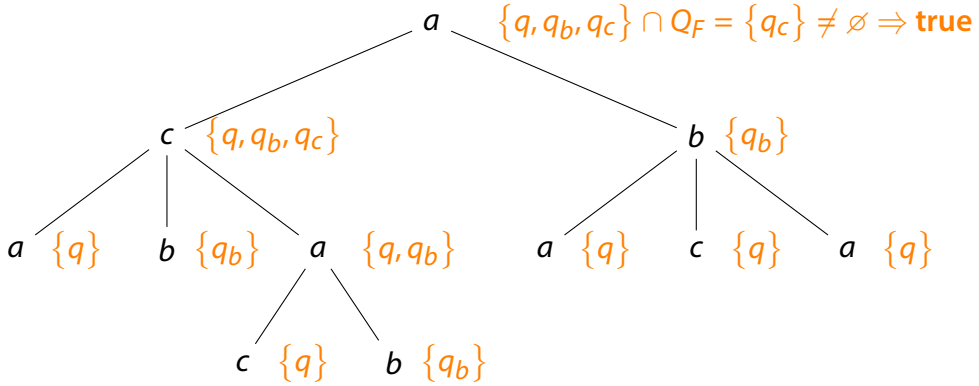
After the algorithm, ρ will be a total function. During the algorithm, we allow ρ to be a partial function, i.e. we allow it to be undefined for some nodes v, $\rho(v) = \bot$.

Input: NHA A, tree t
Output: **true** iff $t \in \mathcal{L}(A)$
Algorithm:

Initialize ρ, $\rho(v) \leftarrow \bot$ for all $v \in \mathcal{T}$.
while $\exists$ node v: $\rho(v) = \bot, \forall$ children $v.i$: $\rho(v.i) \neq \bot$ **do**
  $M \leftarrow \varnothing$
  for transition $R \rightarrow_a q$ with $a = t(v)$ **do**
    if $\exists q_0 \in \rho(v.0), \ldots, \exists q_{n\text{-}1} \in \rho(v.n\text{ - }1)$: $q_0 \ldots q_{n\text{-}1} \in R$ **then**
      $M \leftarrow M \cup \{q\}$
    end if
  end for
  $\rho(v) \leftarrow M$
end while
return true if $\rho(\varepsilon) \cap Q_F \neq \varnothing$, **false** otherwise

5.3 Example

Consider the tree from Example 3.3.



We are not done check: It is not clear how the check if $\exists q_0 \in \rho(v.0), \ldots, \exists q_{n\text{-}1} \in \rho(v.n\text{ - }1)$: $q_0 \ldots q_{n\text{-}1} \in R$ can be implemented in polynomial time.

Naively, we have to iterate over all words $\vec{q} \in Q^n$ with $q_i \in \rho v.i$. This takes in the worst case Q^n, i.e. exponentially many, steps.

It can be implemented in polynomial time as follows:

- Let p_{init} be the initial state of the NFA A_R representing R.

- Collect the set P_0 of states of A_R reachable by symbols (states) in $\rho(v.0)$ from p_0.
- Collect the set P_1 of states of A_R reachable by symbols in $\rho(v.1)$ from a state in P_0.
- etc.
- Collect the set $P_{n\text{-}1}$ of states of A_R reachable by symbols in $\rho(v.n - 1)$ from a state in $P_{n\text{-}1}$. This is the set of states reachable from p_0 by a word $\vec{q}$ with $q_i \in \rho v.i$.
- Check if $P_{n\text{-}1}$ contains a final state of A_R. If yes, the transition can be applied.

The resulting algorithm is a on-the-fly powerset construction along the set of words $\rho v.0. \ldots .\rho v.n - 1$ (in the automaton A_R).

5.4 Algorithm
Input: NFA $A_R = (Q, P, \rightarrow, p_0, P_F)$, $Q_0 = \rho(v.0), \ldots, Q_{n\text{-}1} = \rho(v.n - 1)$
Output: true iff the transition $\mathcal{L}(A_R) \rightarrow_a q$ can be applied
Algorithm:

$P_{\text{-}1} \leftarrow \{p_0\}$
for $i = 0, \ldots, n - 1$ **do**
  $P_i \leftarrow \{p \in P \mid \exists p' \in P_{i\text{-}1} \exists q \in Q_i : p' \rightarrow_q p\}$
end for
return true if $P_{n\text{-}1} \cap P_F \neq \varnothing$, **false** otherwise

Using Algorithm 5.4 to implement the if-check in Algorithm 5.2 yields the desired algorithm for checking membership in polynomial time.

The resulting algorithm does two nested on-the-fly powerset constructions:

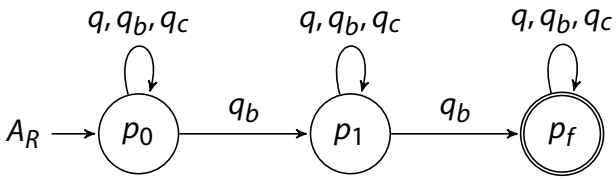
- a bottom-up determinization of A along the given tree, and
- determinizations of the automata A_R for the horizontal languages along the words provided by the states for the child nodes.

5.5 Example
Consider the left child of the root labeled by a. Assume we have already determined the possible sets of states $Q_0 = \{q\}$, $Q_1 = \{q_b\}$, $Q_2 = \{q, q_b\}$ for the child nodes.

We want to check whether the transition

$$Q^* q_b Q^* q_b Q^* \rightarrow_c q_c$$

can be applied. Assume the regular expression is represented by the following automaton.



During the application of Algorithm 5.4, we encounter the following sets of states:

$$P_{\text{-}1} = \{p_0\} \overset{\{q\}}{\longrightarrow} P_0 = \{p_0\} \overset{\{q_b\}}{\longrightarrow} P_1 = \{p_0, p_1\} \overset{\{q,q_b\}}{\longrightarrow} P_2 = \{p_0, p_1, p_f\}\,.$$

Since $p_f \in P_2$, the transition can be applied.