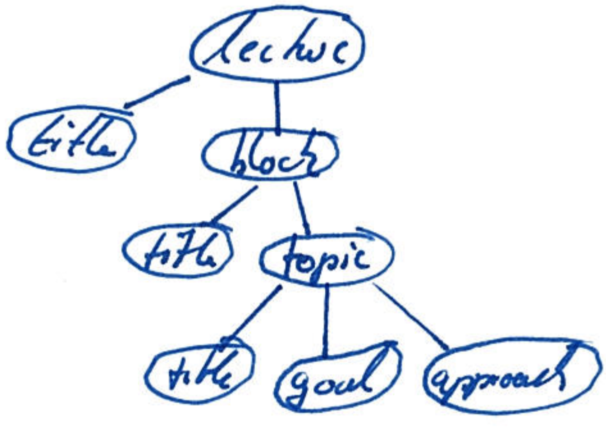
9. XML Schema Languages

↳ XML document

```
<lecture>
    <title> Applied Automata Theory </title>
    <block>
        <title> Finite Words </title>
        <topic>
            <title> WMSO </title>
            <goal> Satisfiability </goal>
            <approach> Buechi </approach>
        </topic>
    </block>
</lecture>
```

yields a tree that

- reflects structure of the document
- without data.



Goal: Pose requirements on the structure of documents

↳ Every lecture is split into blocks

↳ Blocks are divided into topics

Observation: ↳ Requirements describe a tree language over the alphabet of tags

↳ Such a description is called a schema

↳ Document is valid wrt. a schema if it belongs to the tree language defined by the schema

Several XML schema languages exist:

- ↳ Document Type Definitions (here), XML Schema, Relax NG
- ↳ We are interested in
  - ↳ Connection to automata theory
  - (↳ Expressiveness)
  - ↳ Algorithmic problems
    - Is a document valid wrt. a schema? (here)
      (membership in the language)
    - Is there a document that is valid for this schema?
      - → Sanity check
      - → Emptiness in language theory
        (used as subproblem for inclusion)
    - Are all documents valid wrt. one schema
      valid for another schema?
      - → Needed when merging archives/companies
      - → Inclusion in language theory.

9.1 Document Type Definitions and Hedge Automata

- ↳ A document type definition (DTD) is a context-free grammar with regular expressions on the right hand side
- ↳ Tree language of this grammar = all derivation trees.

Example:

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

content model

On the right hand side:

| = choice   + = one or more occurrences

, = sequence   ? = zero or one occurrences

#PCDATA = arbitrary character sequence
(parsed character data, for data inside document)

Corresponding CF-grammar:

lecture → title . (block$^+$ + (topic . (exercise + ε))$^+$)

block → title . (topic . (exercise + ε))$^+$

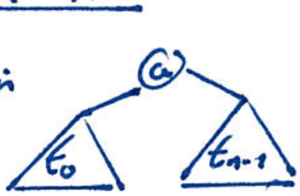
topic → title . goal . (problem + ε) . approach

title → ε

...

To define the tree language described by a DTD, need hedge automata.

9.1.1 Unranked Trees and Hedge Automata

- Drop the restriction of ranked alphabets for trees
  ↳ Remember • Letters in Σ have ranks
  • Trees $t: T \to \Sigma$ obey these ranks
- Consider unranked alphabet Σ
  ↳ Each node has arbitrarily but finitely many children
  ↳ Tree $t: T \to \Sigma$ without further constraints called an unranked tree
- ↳ Call $t_0, \ldots, t_{n-1}$ in  a hedge
- Hedge automata process unranked trees bottom-up
  Goal: Obtain similar properties as for ranked trees
  Problem: Number of successors of a node is not bounded (but finite)
  (unbounded branching)

↳ Transitions cannot be listed

↳ Represent <u>symbolically</u> infinite number of transitions.

<u>Example</u>:

↳ Accept all trees of height 2 where

- root is labelled by "a"
- all children labelled by "b"
- number of children even

↳ Use · two states q, q_b

· transitions $\rightarrow_b q_b$

$\rightarrow_a q$, $(q_b, q_b) \rightarrow_a q$, $(q_b, q_b, q_b, q_b) \rightarrow_a q, \dots$

↳ Infinite set of transitions

can be represented by $(q_b q_b)^* \rightarrow_a q$

More generally,

$$R \rightarrow_a q \quad \text{where } R \subseteq Q^*$$

is a <u>regular language over the states</u> of the automaton.

<u>Definition (Hedge automaton)</u>:

· A <u>(nondeterministic) hedge automaton (NHA) over Σ</u>

is a tuple $A = (Q, \rightarrow, Q_F)$ where

$$\rightarrow \subseteq \mathcal{P}(Q^*) \times \Sigma \times Q$$

with $R \subseteq Q^*$ on lhs of transitions <u>regular</u>.

These R are called <u>horizontal languages</u>.

· A <u>run of A on $t : T \rightarrow \Sigma$</u>

is a function

$$r : T \rightarrow Q$$

so that for all $w \in T$ with $r(w) = q$, $t(w) = a$, and n = # successors of w

we have a transition $R \rightarrow_a q$ with $r(w.0) \dots r(w.(n-1)) \in R$.

To apply a transition $R \to_a q$ at a leaf, we require $\varepsilon \in R$.

- Run is <u>accepting</u> if $r(\varepsilon) \in Q_F$

<u>Language</u> of A is

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

<u>Example:</u>

- There are two nodes labelled "b" whose greatest common ancestor is a "c". (longest common prefix)
- $\Sigma = \{a, b, c\}$

$A = (Q, \to, Q_F)$, $Q = \{q, q_b, q_c\}$, $Q_F = \{q_c\}$

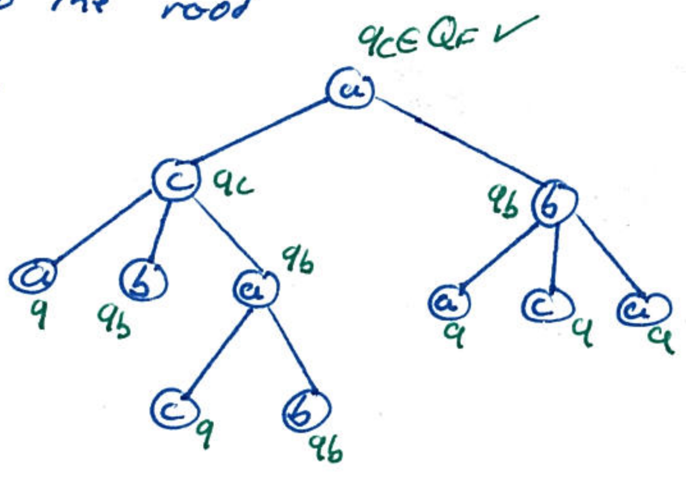
Transitions:

$Q^* \to_{a/c} q$ $\quad Q^* q_b Q^* \to_{a/c} q_b$ $\quad Q^* q_c Q^* \to_{a/b/c} q_c$

$Q^* \to_b q_b$ $\quad Q^* q_b Q^* q_b Q^* \to_c q_c$

- Label b nodes by q_b
- As long as condition has not been satisfied, pass q_b upwards
- Until you find a c node with two q_b children
- Label this node by final state q_c
- Pass to the root

Run on



9.1.2 Document Type Definitions

Definition (DTD):

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

- start symbol s
- function δ assigning each element $a \in \Sigma$ a regular expression $\delta(a)$ over Σ.

To define the language of a DTD, understand it as a hedge automaton

$$A_D := (Q, \rightarrow, Q_F)$$

with

$Q := \{ q_a \mid a \in \Sigma \}$ // State for each letter (tag)

$Q_F := \{ q_s \}$

To define transitions, understand

$\mathcal{L}(\delta(a)) \subseteq \Sigma^*$ as subset of Q^*

(by taking $a_1 \ldots a_n$ as $q_{a_1} \ldots q_{a_n}$).

Then

$\mathcal{L}(\delta(a)) \rightarrow_a q_a$ f.a. $a \in \Sigma$

The language of the DTD is $\mathcal{L}(D) := \mathcal{L}(A_D)$.

Note that A_D is deterministic in the following sense:

for all $R_1 \rightarrow_a q_1$ and $R_2 \rightarrow_a q_2$ we have

$$R_1 \cap R_2 \neq \emptyset \text{ implies } q_1 = q_2 .$$

Example:

Reconsider lecture $\rightarrow$ title. (block* + (topic. (exercise + ϵ))$^+$)

...

The corresponding hedge automaton is

$$A_{LEC} = (Q, \rightarrow, \{q_{lec}\})$$

with

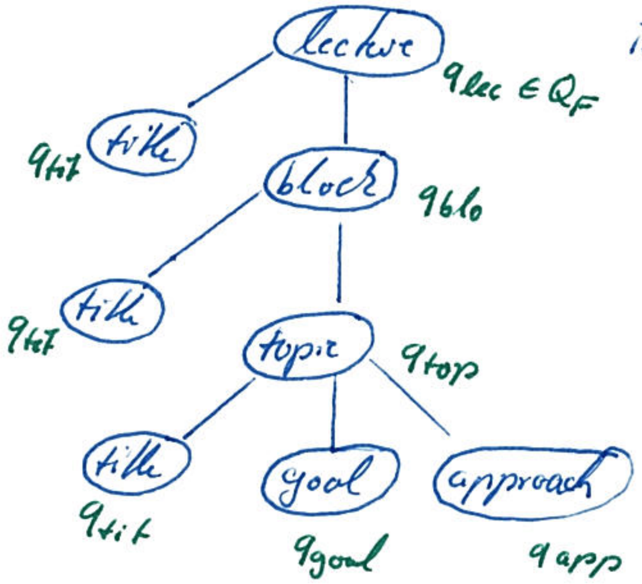
$$Q = \{q_{lec}, q_{tit}, q_{blo}, q_{ex}, q_{top}, q_{goal}, q_{prob}, q_{app}\},$$

and

$$q_{tit} \cdot (q_{blo}^{+} + (q_{top} \cdot (q_{ex} + \varepsilon))^{+}) \rightarrow_{lec} q_{lec}.$$

...

Check that



is valid: ✓

How to check validity systematically?

↳ Algorithm

↳ With low complexity.