XML
Automata and languages

Mirian Halfeld Ferrari

European Master's Program - Information Technologies for Business Intelligence

17 février 2015

Bibliographic notes

References

- **Foundations of XML Processing : The tree-automata approach**. Haruo Hosoya, 2011, Cambridge University Press.
- **Web Data Management**. Serge Abiteboul, Ioana Manolescou, Philippe Rigaux, Marie-Christine Rousset, Pierre Senellart, 2011, Cambridge University Press.
- **Introduction to Automata Theory, Languages and Computation**. J. Hopcroft, R. Motwani and J. Ullman, Addsion Wesley, 2003.
- **Tata Book - Tree Automata Techniques and Applications**. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi. Available on : `http://www.grappa.univ-lille3.fr/tata`

Bibliographic notes

Papers

- M. Murata, D. Lee and M. Mani, **Taxonomy of XML Schema Language using Formal Language Theory**, Extreme Markup Language, Montreal, Canada, 2001.
- Béatrice Bouchou, Ahmed Cheriat, Mírian Halfeld Ferrari, Dominique Laurent, Maria-Adriana Lima and M. Musicante, **Efficient Constraint Validation for Updated XML Databases**, Informatica, volume 31, number 3, pages 285-310, 2007.
- Jacques Chabin, Mirian Halfeld Ferrari Alves, Martin A. Musicante and Pierre Réty, **Minimal Tree Language Extensions : A Keystone of XML Type Compatibility and Evolution** ; Theoretical Aspects of Computing - ICTAC 2010. Pages 60-75.

Contact

Mirian Halfeld Ferrari
LIFO - Batiment IIIA
Rue Léonard de Vinci - BP6759
45067 Orléans Cedex 2
email : mirian@univ-orleans.fr

Schedule

Course subjects

- Automatas : on words, on trees.
- Classes of schema languages.
- Validation process.
- Integrity Constraint Verification (general ideas).

Exercises

Exercises to be done after a course lecture (at home).

Sequences

- In our course, we often consider sequences of various kind...
- For a given set Σ (we call it an alphabet), we write a **sequence** of elements from Σ (we also say a **word** or a **string**) by simply listing them.
- **Length of a string** : the number of positions for symbols in the string.
- Examples :
  - 01101 and 111 are strings from the binary alphabet $\Sigma = \{0, 1\}$. The length $\mid 01101 \mid = 5$ and $\mid 111 \mid = 3$.
  - *abcbbbcc* and *bcabca* are strings from alphabet $\Sigma = \{a, b, c\}$.
- We write the sequence of length 0 by ϵ and we call it the **empty string or sequence**

Concatenation

Concatenation

The **concatenation** of two sequences s and t is written $s.t$ (or st).
Example : Let $x = 01101$ and $y = 110$.
Then $x.y = 01101110$ and $y.x = 11001101$

Prefix and suffix

When sequence s can be written as $t.u$, we say that t is a **prefix** of s and that u is a **suffix** of s.

Powers of an alphabet

Σ^k : the set of strings of length k, each of whose is in Σ.
Examples : Σ^0 : $\{\epsilon\}$, regardless of what Σ is.
If $\Sigma = \{0, 1\}$, then : $\Sigma^1 = \{0, 1\}$ and $\Sigma^2 = \{00, 01, 10, 11\}$ and $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

Kleene Star

Kleene Star

- The set of all sequences of elements in Σ is denoted Σ^*.
  Examples : $\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$
- The symbol $*$ is called **Kleene star** and is named after the mathematician and logician Stephen Cole Kleene.
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots$ Thus : $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$

Order on Σ^*

When a **strict total order** $<$ is defined on Σ, the **lexicographic order** $\preceq$ on Σ^* can be defined : $s \preceq t$ IF
• EITHER $t = s.s'$ for some s' (*i.e.*, s is a **prefix** of t)
• OR ELSE $s = u.a.s'$ and $t = u.b.t'$ for some sequences u, s', t' and some elements a, b with $a < b$
Examples : $11 \preceq 112$, $112 \preceq 121$

Languages

- If Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a (formal) **language** over Σ. In other words, **a (possibly infinite) set of strings all of which are chosen from some Σ^***.
- Examples :
  - English or French.
  - The language of all strings consisting of n 0s followed by n 1s ($n \geq 0$) : $\{\epsilon, 01, 0011, 000111, \ldots\}$.
  - The set of strings of 0s and 1s with an equal number of each : $\{\epsilon, 01, 10, 0011, 0101, 1001, \ldots\}$.
  - Σ^* is a language for any alphabet Σ.
  - $\emptyset$, the empty language, is a language over any alphabet.
  - $\{\epsilon\}$, the language consisting of only the empty string, is also a language over any alphabet.
    IMPORTANT : $\emptyset \neq \{\epsilon\}$ since $\emptyset$ has no strings and $\{\epsilon\}$ has one
  - $\{w \mid w$ consists of an equal number of 0 and 1$\}$.

Regular Expressions

A set of **regular expressions** over Σ, ranged over by r, is defined by the following grammar :

$$
\begin{array}{lll}
r & ::= & \epsilon \\
 & & a \\
 & & r_1.r_2 \\
 & & r_1 \mid r_2 \\
 & & r^*
\end{array}
$$

That is :

- ϵ and any element $a \in \Sigma$ are regular expressions ;
- when r_1 and r_2 are regular expressions, so are $r_1.r_2$ and $r_1 \mid r_2$.
- when r is a regular expression so is r^*.

Example : Let $\Sigma = \{a, b\}$. Thus $(a \mid \epsilon).b$ and $(a.b)^*$ are regular expressions.

Regular Expressions (semantics)

The semantics of a regular expression is usually defined by interpreting it as a language (*i.e.*, set of strings (words) from Σ).

Formally the **language** $L(r)$ of a regular expression r is defined as follows :

$$\begin{array}{lll} L(\epsilon) = & \{\epsilon\} & \\ L(\mathbf{a}) = & \{a\} & \\ L(r_1.r_2) = & L(r_1).L(r_2) = & \{s_1.s_2 \mid s_1 \in L(r_1), s_2 \in L((r_2)\} \\ L(r_1 \mid r_2) = & L(r_1) \cup L(r_2) & \\ L(r^*) = & (L(r))^* = & \{s_1.s_2 \ldots s_n \mid s_1 \ldots s_n \in L(r), n \geq 0\} \end{array}$$

Example : $L((a \mid \epsilon).b) = \{ab, b\}$ and $L((a \mid b)^*) = \{\epsilon, a, b, ab, ba, aa, bb, aaa, bbb, \ldots\}$

String automata

A non-deterministic string automaton

A **finite state automaton (FSA)** on Σ is a 5-tuple $A = (Q, \Sigma \cup \{\epsilon\}, \delta, q_0, F)$

1. A finite set of **states**, commonly denoted by Q.
2. A finite set of **input symbols**, commonly denoted by Σ.
3. A set of transition rules of the form $a, q \to q'$ where $q, q' \in Q$ and $a \in \Sigma \cup \{\epsilon\}$.
4. A **start state**, $q_0 \in Q$
5. A set of **final or accepting** states F ($F \subseteq Q$)

String automata - semantics

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a string automaton (FSA).

- The semantics of the string automata is defined in terms of runs.
- A run on a string $s = a_1 \dots a_n$ is a sequence $q_1 \dots q_n$ of states from Q such that :
  1. $q_1 = q_0$ is the initial state of the automaton A and
  2. $a, q_i \to q_{i+1} \in \delta$ for each $i = 1, \dots, n-1$.
- Such a run **accepts** s when $q_n \in F$. In this case we say that **the automaton accepts the string (word)**.

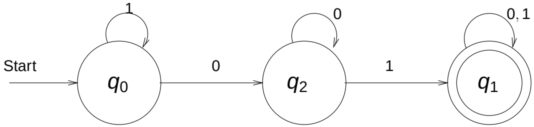
Example

An automaton accepting strings with a sub-string 01 :

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$$

where the set of rules δ is given by the *transition table* :

		0	1
$\rightarrow$	q_0	q_2	q_0
$\star$	q_1	q_1	q_1
	q_2	q_2	q_1



Regular languages

- The set of words accepted by an automaton A is called **the language of** A and written $L(A)$.
- The class of languages each accepted by an FSA is called the class of **regular string languages**. These languages can alternatively be described by regular expressions.

Deterministic and non-deterministic automaton

- A string automaton $A = (Q, \Sigma, \delta, q_0, F)$ is **deterministic** when :
  1. it has no ϵ-transition and
  2. for each state $q \in Q$ and label $a \in \Sigma$ there is at most one transition $a, q \to q' \in \delta$.
- Alternatively, we can define δ as a transition function from $(\Sigma \cup \{\epsilon\}) \times Q$ to 2^Q. Then, the only difference between a NFA and a DFA is the result that δ returns : deterministic automata return a singleton.

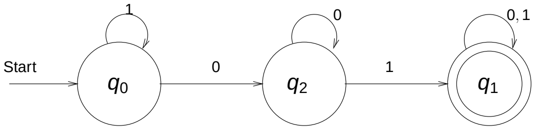
Example (1)

A DFA accepting strings with a sub-string 01 :

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$$

where δ is given by the *transition table* :

		0	1
$\rightarrow$	q_0	q_2	q_0
$\star$	q_1	q_1	q_1
	q_2	q_2	q_1



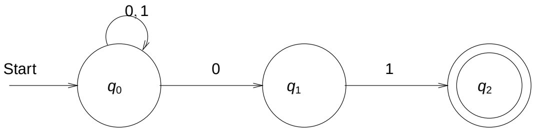
Example (2)

Non-deterministic automaton that accepts all and only the strings of 0s and 1s that end in 01.

$$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$$

where δ is given by the table below. Notice that for q_0 and 0, one obtains $\{q_0, q_1\}$, *i.e.*, transitions $0, q_0 \rightarrow q_0$ and $0, q_0 \rightarrow q_1$ are in δ.

		0	1
$\rightarrow$	q_0	$\{q_0, q_1\}$	$\{q_0\}$
	q_1	$\emptyset$	$\{q_2\}$
$\star$	q_2	$\emptyset$	$\emptyset$



Example (3)

A NFA is allowed to make a transition spontaneously, without receiving an input symbol. This capability does not expand the class of languages that can be accepted by finite automata, but it does give us some added programming convenience.

$$A = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_2\})$$

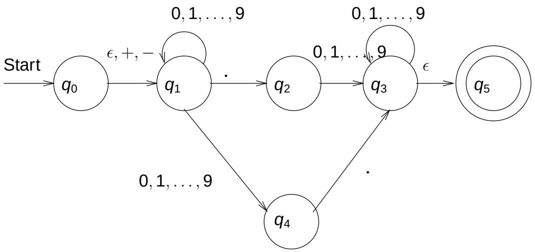
where δ is given by the transition table (notice the ϵ-transition).

		a	b	ϵ
$\rightarrow$	q_0	$\{q_0, q_1\}$	$\emptyset$	$\emptyset$
	q_1	$\emptyset$	$\{q_0\}$	$\{q_2\}$
$\star$	q_2	$\emptyset$	$\emptyset$	$\{q_3\}$
	q_3	$\emptyset$	$\emptyset$	$\{q_3\}$

$\boxed{\text{Draw it !}}$

Example (4)

NFA that accepts decimal numbers consisting of : an optional $+$ or $-$ sign ; a string of digits ; a decimal point and another string of digits. Either the first or the second string of bits can be empty, but at least one of the two strings must be non-empty.



Important results on string automata

- For any string automaton with ϵ-transitions, there is a string automaton without ϵ-transitions that accepts the same language.
- For each non-deterministic string automaton (NFA), there is a deterministic string automaton (DFA) that accepts the same language.
- Given a NFA, we can construct a DFA which, in practice has about as many states as the NFA, although it has more transitions. However, **in the worst case**, the smallest DFA can have 2^n (for a smallest NFA with n state).
- For any regular expression there is a string automaton that accepts the same language.
- There is no FSA accepting the language $\{a^i.b^i \mid i \geq 0\}$

Important results on regular string languages

Closure Properties of Regular Languages

Let L and M be regular languages. Then the following languages are all regular :

- Union : $L \cup M$.
- Intersection : $L \cap M$.
- Complement : $\overline{L}$ where $\overline{L} = \Sigma^* \setminus L$.
- Difference : $L \setminus M$.
- Reversal : $L^R = w^R : w \in L$.
- Closure : L^*.
- Concatenation : $L.M$.

Introducing Tree Automata

- FSA (on words) are used to define word languages, *i.e.*, subsets of Σ^*.
- It is possible to define **tree automata** whose propose is to define subsets of the set of all trees.
- Tree automata provide a finite-state acceptor model for trees that naturally **corresponds to schemas**.
- With FSA (on words) there is no difference between left-to-right (one that reads a word from left to right) and a right-to-left (one that reads a word from right to left) automata.
- **Trees can be read top-down or bottom up**. However, there is a difference between top-down and bottom-up tree automata.

Ranked and Unranked Tree Automata

There are two types of tree automata :

Ranked

Trees are ranked, *i.e.*, the number of children per node is bounded.

- A lot of work has already been done in this field, easier to specify.
- XML trees can be seen as binary trees. Each node has two pointers : a left one pointing to the first child and a right one pointing to the next sibling.

Unranked

Trees are unranked, *i.e.*, there is no prior bound on the number of children of a node.

- Extension of ranked automata, a better adaptation to XML world.

Binary tree automata

Bottom-up non-deterministic tree automata

A non-deterministic bottom-up tree automata is a 4-tuple $\mathcal{A} = (\Sigma, Q, F, \Delta)$ where

- Σ is an alphabet. We usually distinguish between two disjoint alphabets : a leaf alphabet (Σ_{leaf}) and an internal one ($\Sigma_{internal}$).
- Q is a set of states.
- F is a set of accepting states $F \subseteq Q$.
- Δ is a set of transition rules having one of the forms :

$$l \rightarrow q \text{ when } l \in \Sigma_{leaf}$$
$$a(q_1, q_2) \rightarrow q \text{ when } a \in \Sigma_{internal}$$

Binary tree automata - semantics

Let $\mathcal{A} = (\Sigma, Q, F, \Delta)$ be a bottom-up non-deterministic binary tree automaton.

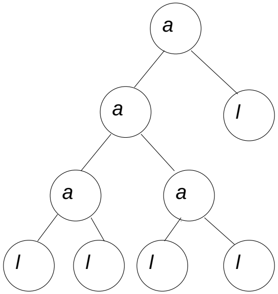
- The semantics of $\mathcal{A}$ is described in terms of runs.
- Given a ranked tree t, a **run** r of $\mathcal{A}$ on t is a mapping from $dom(t)$ to Q such that for each $p \in dom(t)$ we have $r(p) = q \in Q$ where :
  - if p is a leaf position then there exists a transition $a \to q \in \Delta$ or
  - if p is an internal position then there exists a transition $a(q_1, q_2) \to q \in \Delta$ with $r(p.0) = q_1$ and $r(p.1) = q_2$.

  A bottom-up run is successful if $r(\epsilon) \in F$.

Example

Let $\mathcal{A} = (\{a, l\}, \{q_0, q_1\}, \{q_0\}, \Delta)$ where

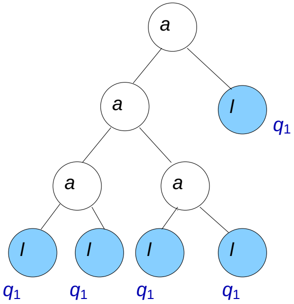
$$\Delta = \begin{cases} a(q_1, q_1) & \to q_0 \\ a(q_0, q_0) & \to q_1 \\ l & \to q_1 \end{cases}$$



Example

Let $\mathcal{A} = (\{a, l\}, \{q_0, q_1\}, \{q_0\}, \Delta)$ where

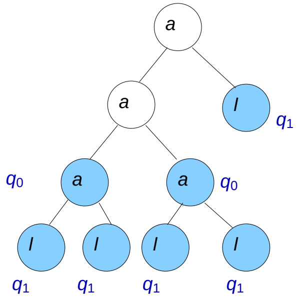
$$\Delta = \begin{cases} a(q_1, q_1) & \to q_0 \\ a(q_0, q_0) & \to q_1 \\ l & \to q_1 \end{cases}$$



Example

Let $\mathcal{A} = (\{a, l\}, \{q_0, q_1\}, \{q_0\}, \Delta)$ where

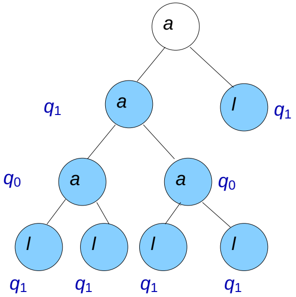
$$\Delta = \begin{cases} a(q_1, q_1) & \rightarrow q_0 \\ a(q_0, q_0) & \rightarrow q_1 \\ l & \rightarrow q_1 \end{cases}$$



Example

Let $\mathcal{A} = (\{a, l\}, \{q_0, q_1\}, \{q_0\}, \Delta)$ where

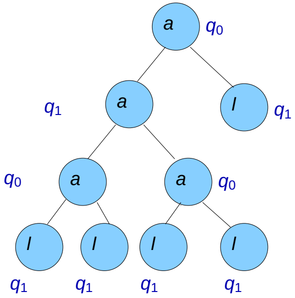
$$\Delta = \begin{cases} a(q_1, q_1) & \to q_0 \\ a(q_0, q_0) & \to q_1 \\ l & \to q_1 \end{cases}$$



Example

Let $\mathcal{A} = (\{a, l\}, \{q_0, q_1\}, \{q_0\}, \Delta)$ where

$$\Delta = \begin{cases} a(q_1, q_1) & \to q_0 \\ a(q_0, q_0) & \to q_1 \\ l & \to q_1 \end{cases}$$



Remarks

- The definition can be extended to ranked trees with symbols of arbitrary arity.
- A bottom-up tree automata with ϵ-transition may contain rules of the form $\epsilon, q \to q'$ where $q, q' \in Q$, in addition to the labeled transition rules. This means that if a node is in state q, then it may move to state q'.
- It is also possible to define (and build) **deterministic bottom-up tree automata** by forbidding ϵ-transitions and transitions having the same left-hand side (with different right-hand side).

Definitions

- We define the language $L(\mathcal{A})$ to be the set of trees accepted by $\mathcal{A}$.
- **A language accepted by a bottom-up tree automaton is called a regular tree language.**

Top-down tree automata

Binary top-down tree automata

A non-deterministic top-down tree automata is a 5-tuple $\mathcal{A} = (\Sigma, Q, I, F, \Delta)$ where

- Σ is an alphabet.
- Q is a set of states.
- $I \subseteq Q$ is a set of initial states.
- F is a set of accepting states $F \subseteq Q$.
- Δ is a set of transition rules having the form :

  $$q \rightarrow a(q_1, q_2).$$

  where $a \in \Sigma$ and $q, q_1, q_2 \in Q$

Top down tree automata - semantics

Run

- A run of top-down automaton $\mathcal{A} = (\Sigma, Q, I, F, \Delta)$ on a binary tree t is a mapping $r : dom(t) \rightarrow Q$ such that
  - $r(\epsilon) \in I$;
  - for each node p with label a, rule $r(p) \rightarrow a(r(p.0), r(p.1))$ is in Δ.
- A run is accepting if for all leaves p we have $r(p) \in F$.

Deterministic binary top-down automata

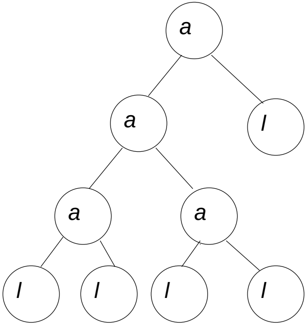
We say that a binary tree automaton is (top-down) deterministic **if I is a singleton and for each $a \in \Sigma$ and $q \in Q$ there is *at most* one transition rule of the form $q \rightarrow a(q_1, q_2)$.**

Example

Let $\mathcal{A} = (\{a, l\}, \{q_0, q_1\}, \{q_0\}, \{q_1\}, \Delta)$ where

$$\Delta = \begin{cases} q_0 & \rightarrow a(q_1, q_1) \\ q_1 & \rightarrow a(q_0, q_0) \end{cases}$$

Note : In general this tree automaton accepts trees in which every leaf is at an even depth.

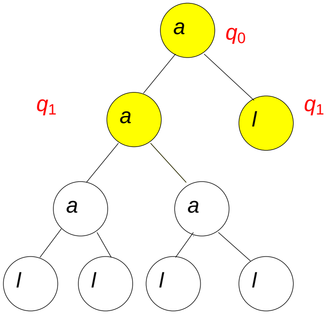


Example

Let $\mathcal{A} = (\{a, l\}, \{q_0, q_1\}, \{q_0\}, \{q_1\}, \Delta)$ where

$$\Delta = \begin{cases} q_0 & \rightarrow a(q_1, q_1) \\ q_1 & \rightarrow a(q_0, q_0) \end{cases}$$

Note : In general this tree automaton accepts trees in which every leaf is at an even depth.

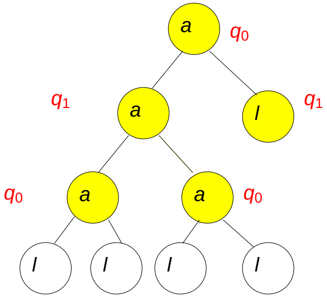


Example

Let $\mathcal{A} = (\{a, l\}, \{q_0, q_1\}, \{q_0\}, \{q_1\}, \Delta)$ where

$$\Delta = \begin{cases} q_0 & \rightarrow a(q_1, q_1) \\ q_1 & \rightarrow a(q_0, q_0) \end{cases}$$

Note : In general this tree automaton accepts trees in which every leaf is at an even depth.

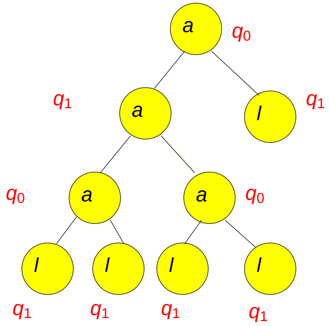


Example

Let $\mathcal{A} = (\{a, l\}, \{q_0, q_1\}, \{q_0\}, \{q_1\}, \Delta)$ where

$$\Delta = \begin{cases} q_0 & \to a(q_1, q_1) \\ q_1 & \to a(q_0, q_0) \end{cases}$$

Note : In general this tree automaton accepts trees in which every leaf is at an even depth.



Deterministic top-down automata are weaker(1)

- Consider the language $L_1 = \{a(b(I,I), c(I,I)), a(c(I,I), b(I,I))\}$.
- Let $\mathcal{A} = (\Sigma, Q, I, F, \Delta)$ be a top-down tree automaton where

$$\begin{array}{ll} Q = & \{q_0, q_1, q_2, q_3\} \\ I = & \{q_0\} \\ F = & \{q_3\} \\ \Delta = & \left\{ \begin{array}{ll} q_0 & \rightarrow a(q_1, q_2) \\ q_0 & \rightarrow a(q_2, q_1) \\ q_1 & \rightarrow b(q_3, q_3) \\ q_2 & \rightarrow c(q_3, q_3) \end{array} \right. \end{array}$$

- Clearly $\mathcal{A}$ accepts L but $\mathcal{A}$ is not a top-down deterministic tree automaton.
- Can we build a top-down deterministic tree automaton that accepts L_1 ? We will prove that this is NOT possible.

Deterministic top-down automata are weaker(2)

- We suppose that the language L_1 is accepted by a top-down deterministic tree automata $\mathcal{A}' = (\Sigma, Q', I', F', \Delta')$.
- By definition, I' must be a singleton set $\{q'_0\}$ and since L_1 contains a tree with root labeled a, there must be a transition rule $q'_0 \rightarrow a(q'_1, q'_2) \in \Delta$ for some $q'_1, q'_2 \in Q$ and no other transition for q'_0 with label a.
- Since $\mathcal{A}'$ accepts both trees in L_1 in state q'_0 there must also be the following transitions in Δ

$$\begin{aligned} q'_1 &\rightarrow b(q'_{11}, q'_{12}) \\ q'_2 &\rightarrow c(q'_{21}, q'_{22}) \\ q'_1 &\rightarrow c(q'_{13}, q'_{14}) \\ q'_2 &\rightarrow b(q'_{23}, q'_{24}) \end{aligned}$$

  where all $q_{ij} \in F$.
- But this implies that A' must accept also trees $a(b(l,l), b(l,l))$ and $a(c(l,l), c(l,l))$. A contradiction !

Regular tree languages

The following are equivalent :

- L is a regular tree language.
- L is accepted by a non-deterministic bottom-up automaton.
- L is accepted by a deterministic bottom-up automaton.
- L is accepted by a non-deterministic top-down automaton.

Important results

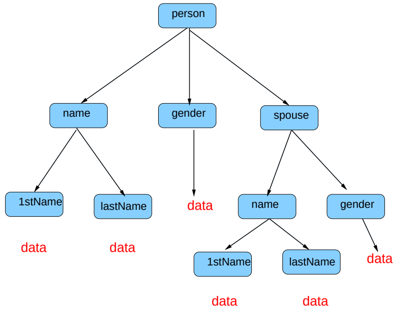
Generally speaking, one can show for regular tree languages the same results as for regular string languages (sometimes the complexity is higher). In particular :

- Given a tree automaton, one can find an equivalent (bottom-up only) automaton that is deterministic (with possibly an exponential blow-up).
- Regular tree languages are closed under complement, intersection and union.

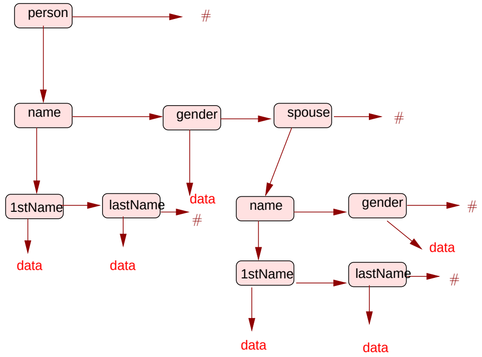
Ranked tree automata for XML

Binarization technique

- One can represent any unranked tree by a binary tree where the left child of a node represents the first child and the right child its next sibling in the original unranked tree.
- This is called the first-child and next-sibling encoding. Others encodings exist.



person
name
gender
spouse
1stName
lastName
data
name
gender
1stName
lastName
data
data
data
data
data



person
#
name
gender
spouse
#
1stName
lastName
data
#
name
gender
#
data
data
data
1stName
lastName
#
data
data

Ranked tree automata for XML

Relation between ranked and unranked tree automata

- Let cod_{fcns} be the one-to-one mapping that encodes an unranked tree t into $cod_{fcns}(t)$, the binary tree with first-child and next-sibling.
- Let $decod_{fcns}$ be its inverse mapping that decodes the binary tree thereby encoded.
- One can prove that :
  - for each unranked tree automaton $\mathcal{A}_{unrank}$, there exists a ranked tree automaton accepting $cod_{fcns}(L(\mathcal{A}_{unrank}))$ and conversely,
  - for each ranked tree automaton $\mathcal{A}_{rank}$, there is an unranked tree automaton accepting $decod_{fcns}(L(\mathcal{A}_{rank}))$.

Unranked bottom-up tree automata

Non-deterministic bottom-up tree automata

A non-deterministic bottom-up tree automaton is a 4-tuple $\mathcal{A} = (\Sigma, Q, F, \Delta)$ where Σ is an alphabet, Q is a set of states, $F \subseteq Q$ is a set of final states an Δ is a set of transition rules of the form

$$a[E] \to q$$

where $a \in \Sigma$, E is a regular expression over Q and $q \in Q$

Unranked tree automata - semantics

Let $\mathcal{A} = (\Sigma, Q, F, \Delta)$ be an unranked tree automata.

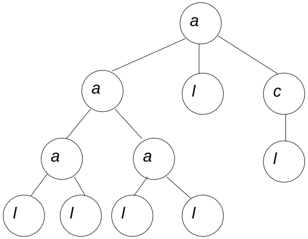
- The semantics of $\mathcal{A}$ is described in terms of runs
- Given an **unranked tree** t, a *run* of $\mathcal{A}$ on t is a mapping from $dom(t)$ to Q where, for each position p whose children are at positions $p0, \ldots, p(n-1)$ (with $n \geq 0$), we have $r(p) = q$ if all the following conditions hold :
  - $t(p) = a \in \Sigma$,
  - the mapping r is already defined for the children of p, *i.e.*, $r(p.0) = q_0, \ldots, r(p.(n-1)) = q_{n-1}$ and
  - the word $q_0.q_1 \ldots q_{n-1}$ is in $L(E)$.
- A run r is *successful* if $r(\epsilon)$ is a final state.

Example

Let $\mathcal{A} = (\{a, l\}, \{q_a, q_c, q_l\}, \{q_a\}, \Delta)$ where

$$\Delta = \begin{cases} a\ [q_a^*.q_l^*.(q_c \mid \epsilon)] & \to q_a \\ c\ [q_l] & \to q_c \\ l\ [\epsilon] & \to q_l \end{cases}$$

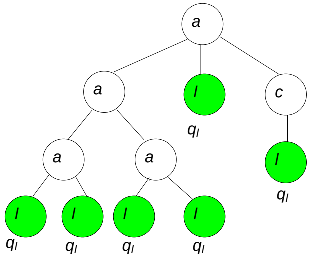
Special rule for leaves



Example

Let $\mathcal{A} = (\{a, l\}, \{q_a, q_c, q_l\}, \{q_a\}, \Delta)$ where

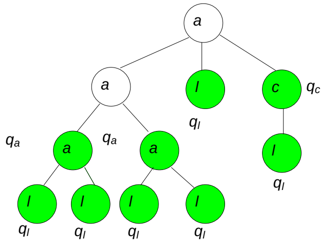
$$\Delta = \begin{cases} a\ [q_a^*.q_l^*.(q_c \mid \epsilon)] & \rightarrow q_a \\ c\ [q_l] & \rightarrow q_c \\ l\ [\epsilon] & \rightarrow q_l \end{cases}$$ Special rule for leaves



Example

Let $\mathcal{A} = (\{a, l\}, \{q_a, q_c, q_l\}, \{q_a\}, \Delta)$ where

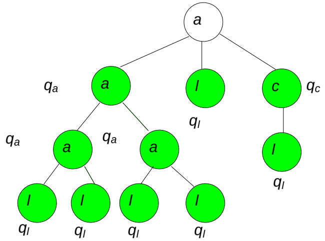
$$\Delta = \begin{cases} a\ [q_a^*.q_l^*.(q_c \mid \epsilon)] & \rightarrow q_a \\ c\ [q_l] & \rightarrow q_c \\ l\ [\epsilon] & \rightarrow q_l \end{cases}$$ Special rule for leaves



Example

Let $\mathcal{A} = (\{a, l\}, \{q_a, q_c, q_l\}, \{q_a\}, \Delta)$ where

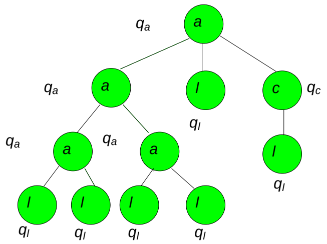
$$\Delta = \begin{cases} a\ [q_a^*.q_l^*.(q_c \mid \epsilon)] & \to q_a \\ c\ [q_l] & \to q_c \\ l\ [\epsilon] & \to q_l \end{cases}$$ Special rule for leaves



Example

Let $\mathcal{A} = (\{a, l\}, \{q_a, q_c, q_l\}, \{q_a\}, \Delta)$ where

$$\Delta = \begin{cases} a\ [q_a^*.q_l^*.(q_c \mid \epsilon)] & \to q_a \\ c\ [q_l] & \to q_c \\ l\ [\epsilon] & \to q_l \end{cases}$$ Special rule for leaves



Deterministic bottom-up automata

Determinism

An unranked tree automata $\mathcal{A}$ is **deterministic** when there is no possibility that it reaches two different states for the same input. In other words, for any two transition rules having the form $a[E_1] \rightarrow q_1$ and $a[E_2] \rightarrow q_2$ in Δ, we have $L(E_1) \cap L(E_2) = \emptyset$.

Important results and comments

Closure properties

As unranked tree automaton can be associated to a ranked tree automaton recognizing the same language (through encoding)... **one can prove that unranked tree automata are closed under union, intersection and complement.**

Top-down automata

- Defined in a similar way of the ranked version. We do not discuss them here.
- Deterministic top-down automata are weaker (similar as their ranked counter part)

Schemas and tree grammar

Regular Tree Grammar (RTG)

A *regular tree grammar* (RTG) is a 4-tuple $G = (N, T, S, P)$, where :

- N is a finite set of *non-terminal symbols* ;
- T is a finite set of *terminal symbols* ;
- S is a set of *start symbols*, where $S \subseteq N$ and
- P is a finite set of *production rules* of the form $X \rightarrow a\,[R]$, where $X \in N$, $a \in T$, and R is a regular expression over N.

(We say that, for a production rule, X is the left-hand side, $a\,R$ is the right-hand side, and R is the content model.)

Schemas

Schemas for XML documents can be formally expressed by RTG.

Example

P_1
$Dir \rightarrow directory[Person^*]$
$Person \rightarrow student[DirA \mid DirB])$
$Person \rightarrow professor[DirB]$
$DirA \rightarrow direction[Name.Number?.Add?]$
$DirB \rightarrow direction[Name.Add?.Phone^*]$

Classes of Schemas (some definitions)

Grammar Context : Competing Non-Terminals

Two different non-terminals A and B (of the same grammar G) are said to be **competing with each other** if
(i) a production rule has A in the left-hand side,
(ii) another production rule has B in the left-hand side, and
(iii) these two production rules share the same terminal symbol in the right-hand side.

Automaton Context : Competing States

Let $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$ be a tree automaton. Two different states q_1 and q_2 in Q are **competing** if Δ contains different transition rules ($a[E_1] \rightarrow q_1$ and $a[E_2] \rightarrow q_2$) which share the same label a.
Remark : we assume that no two transition rules have the same state in the right-hand side and the same label in the left-hand side, since two rules of this kind can be written as a single one.

Example

P_1	Δ_1
$Dir \rightarrow directory[Person^*]$	$directory[q_{person}^*] \rightarrow q_{dir}$
$Person \rightarrow student[DirA \mid DirB])$	$student[q_{dirA} \mid q_{dirB}] \rightarrow q_{person}$
$Person \rightarrow professor[DirB]$	$professor[q_{dirB}] \rightarrow q_{person}$
$DirA \rightarrow direction[Name.Number?.Add?]$	$direction[q_{name}.q_{number}?.q_{add}?] \rightarrow q_{dirA}$
$DirB \rightarrow direction[Name.Add?.Phone^*]$	$direction[q_{name}.q_{add}?.q_{phone}^*] \rightarrow q_{dirB}$

Classes of Schemas (Local Tree Grammar)

Local Tree Grammar

(LTG) A *local tree grammar* is a regular tree grammar that does not have competing non-terminals.

Local Tree Language

(LTL) A *local tree language* (LTL) is a language that can be generated by at least one LTG, or, a LTL is a regular tree language accepted by a tree automaton that does not have competing states.

Example

P_3	Δ_3
$Dir \rightarrow directory[Student^*.Professor^*]$	$directory[q_{stud}^*.q_{prof}^*] \rightarrow q_{dir}$
$Student \rightarrow student[Name.Number?.Add?]$	$student[q_{name}.q_{number}?.q_{add}?] \rightarrow q_{stud}$
$Professor \rightarrow professor[Name.Add?.Phone^*]$	$professor[q_{name}.q_{add}?.q_{phone}^*] \rightarrow q_{prof}$

Classes of Schemas (Single-Type Tree Grammar)

Single-Type Tree Grammar

(STTG) A **single-type tree grammar** (STTG) is a regular tree grammar in normal form, where
(*i*) for each production rule, non terminals in its regular expression do not compete with each other, and
(*ii*) starts symbols do not compete with each other.

Single-Type Tree Language

(STTL) A **single-type tree language** (STTL) is a language that can be generated by at least one STTG, or, a STTL is a regular tree language accepted by a tree automaton having the following characteristics :
(i) For each transition rule, the states in its regular expression do not compete with each other and
(ii) the set Q_f is a singleton.

Example

P_2	Δ_2
$Dir \to directory[Person^*]$	$directory[q_{person}^*] \to q_{dir}$
$Person \to student[DirA])$	$student[q_{dirA}] \to q_{person}$
$Person \to professor[DirB]$	$professor[q_{dirB}] \to q_{person}$
$DirA \to direction[Name.Number?.Add?]$	$direction[q_{name}.q_{number}?.q_{add}?] \to q_{dirA}$
$DirB \to direction[Name.Add?.Phone^*]$	$direction[q_{name}.q_{add}?.q_{phone}^*] \to q_{dirB}$

Classes of regular languages

Expression power

LTL $\subset$ STTL $\subset$ RTL

Properties

The LTL and STTL are closed under intersection but not under union ; while the RTL are closed under union, intersection and difference.

XML Schema Languages

Grammar	XML Schema Language
LTG	DTD
STTG	XSD
RTG	RELAX

A general validation algorithm

- Let $\mathcal{A}$ be a general tree automaton and t an XML tree.
- In general, we can have many successful runs of $\mathcal{A}$ over t.
- In this context, we can define a general validation algorithm where a run of $\mathcal{A}$ over t associates **a set of states** $\mathcal{Q}_p$ to each position $p \in dom(t)$ where $\mathcal{Q}_p$ is composed by all the states q such that :
  1. There exist a transition rule $a[E] \rightarrow q$ in $\mathcal{A}$.
  2. $t(p) = a$.
  3. There is a word $w = q_1, \ldots, q_n$ in $L(E)$ such that $q_1 \in \mathcal{Q}_1, \ldots, q_n \in \mathcal{Q}_n$, where $\mathcal{Q}_1 \ldots \mathcal{Q}_n$ are the set of states associated to each element child of p.
- A run r is *successful* if $r(\epsilon)$ is a set containing at least one final state.

Simplified versions of validation algorithm

Restricted forms of schema languages permit simplified versions of a validation algorithm

LTG

- The sets $\mathcal{Q}_p$ are always singleton.
- There is only one rule associated to each label.

STTG

Although it is possible to have competing states, the result of a successful run of such an automaton can consider just a single type (state) for each node of the tree.

Simplified versions of validation algorithm

Building simplified versions

- While reading an XML document, we can store information useful to avoid the possible ambiguity of state assignment, expressed by the transition rules.
- For instance, in the implementation of the run of a tree automaton for XSD, each tree node can always be associated to one single state.
- This state is obtained by intersecting a set of “expected” states (computed during the sequential reading of the document so far) and the set of states obtained by the bottom-up application of the rules of the automaton.