M1 MPRI : Automates et Applications

Lecture 2
Regular Tree Languages

kn@lri.fr

January 23, 2024

Outline

Previously, in *Automates et applications*

We have recapped definitions on finite *word* automata and their usefulness. They provide an execution model, that is, they are an efficient virtual machine for particular programs : regular expressions.
Regexps have a practical utility : check the well-formedness of strings, search for patterns in a text, ...

Previously, in *Automates et applications*

We have recapped definitions on finite *word* automata and their usefulness. They provide an execution model, that is, they are an efficient virtual machine for particular programs : regular expressions.

Regexps have a practical utility : check the well-formedness of strings, search for patterns in a text, . . .

What is a word on an alphabet Σ ?

Previously, in *Automates et applications*

We have recapped definitions on finite *word* automata and their usefulness. They provide an execution model, that is, they are an efficient virtual machine for particular programs : regular expressions.

Regexps have a practical utility : check the well-formedness of strings, search for patterns in a text, ...

What is a word on an alphabet Σ ? A sequence of symbols $w = abc \dots$

Previously, in *Automates et applications*

We have recapped definitions on finite *word* automata and their usefulness. They provide an execution model, that is, they are an efficient virtual machine for particular programs : regular expressions.

Regexps have a practical utility : check the well-formedness of strings, search for patterns in a text, ...

What is a word on an alphabet Σ ? A sequence of symbols $w = abc\ldots$

Can we generalize this concept ?

Previously, in *Automates et applications*

We have recapped definitions on finite *word* automata and their usefulness. They provide an execution model, that is, they are an efficient virtual machine for particular programs : regular expressions.

Regexps have a practical utility : check the well-formedness of strings, search for patterns in a text, . . .

What is a word on an alphabet Σ ? A sequence of symbols

$w = abc \ldots$

Can we generalize this concept ?

$$a \longrightarrow b \longrightarrow c \to \cdots \qquad \rightsquigarrow \qquad \begin{tikzcd} & b \arrow[r] & a \arrow[r] & \cdots \\ a \arrow[ur] \arrow[dr] & & d \arrow[r] & \cdots \\ & c \arrow[ur] \arrow[r] \arrow[dr] & a \arrow[r] & \cdots \\ & & c \arrow[r] & \cdots \end{tikzcd}$$

Previously, in *Automates et applications*

We have recapped definitions on finite *word* automata and their usefulness. They provide an execution model, that is, they are an efficient virtual machine for particular programs : regular expressions.

Regexps have a practical utility : check the well-formedness of strings, search for patterns in a text, ...

What is a word on an alphabet Σ ? A sequence of symbols $w = abc \ldots$

Can we generalize this concept ?

$$a \longrightarrow b \longrightarrow c \rightarrow \cdots \quad \rightsquigarrow \quad \begin{tikzcd} & b \arrow[r] & a \arrow[r] & \cdots \\ a \arrow[ur] \arrow[dr] & & d \arrow[r] & \cdots \\ & c \arrow[ur] \arrow[r] \arrow[dr] & a \arrow[r] & \cdots \\ & & c \arrow[r] & \cdots \end{tikzcd}$$

If we allow a symbol to have *several* successors, we get a *tree*.

Trees ?

- What is the formal definition of a tree ?
- What is the "high-level" programming language for trees, what "real life problem™" can we solve ?
- What is the VM (the computation model) ?

Outline

Ranked alphabet

Definition (Ranked alphabet)

A ranked alphabet is a pair $(\Sigma, |\cdot|)$ *where* Σ *is a set of symbols and* $|\cdot| : \Sigma \to \mathbb{N}$ *is a function called arity.*

The notation $|\cdot|$ means that one calls the function by putting its argument in place of the dot. For example $|a| = 0$, $|f| = 4$, etc. We write $\Sigma_k = \{a \in \Sigma \mid |a| = k\}$.
In what follows, we shall talk of a ranked alphabet Σ and leave the arity function implicit, when clear from the context. We write $\Sigma = \{f^k, g^n, a^0\}$ to denote that $|f| = k$, $|g| = n$ et $|a| = 0$.

n-ary tree

Definition

A tree t is a function $t : S \to \Sigma$ where, Σ is a ranked alphabet and $S \subseteq \mathbb{N}_1^*$ is a set of integer sequences called paths such that:

Empty path: $\epsilon \in S$

Prefix-closure: $i_0, \cdots i_{n-1} i_n \in S, \Rightarrow i_0 \cdots i_{n-1} \in S$

Well-formedness: $\forall p \in S, i \in \mathbb{N}, pi \in S \Rightarrow i \leq |t(p)|$

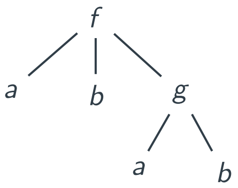
We write $Dom(t)$ the domain of a function t.

Warning: $\mathbb{N}_1$ denotes the set of strictly positive natural numbers, and $\mathbb{N}_1^*$ represents the sequences of such numbers (* is the Kleene star).

Example

Consider the “tree” drawn below.

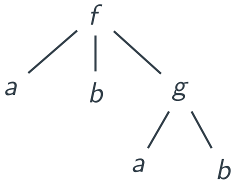
What’s its formal definition ?



Example

Consider the "tree" drawn below.

What's its formal definition ?



$$t : \begin{array}{lll} \epsilon & \mapsto & f \\ 1 & \mapsto & a \\ 2 & \mapsto & b \\ 3 & \mapsto & g \\ 31 & \mapsto & a \\ 32 & \mapsto & b \end{array}$$

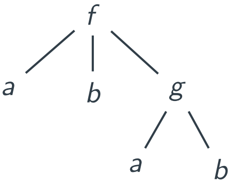
Children are ordered. The path ϵ is called the root. Any path p such that $|t(p)| = 0$ is called a *leaf*.

Example

Consider the “tree” drawn below.

What’s its formal definition ?

How can we represent it compactly ?



$$t : \begin{array}{lcl} \epsilon & \mapsto & f \\ 1 & \mapsto & a \\ 2 & \mapsto & b \\ 3 & \mapsto & g \\ 31 & \mapsto & a \\ 32 & \mapsto & b \end{array}$$

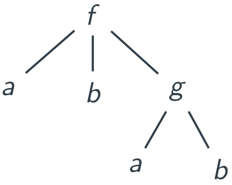
Children are ordered. The path ϵ is called the root. Any path p such that $|t(p)| = 0$ is called a *leaf*.

Example

Consider the “tree” drawn below.

What’s its formal definition ?

How can we represent it compactly ?



$$t : \begin{array}{lll} \epsilon & \mapsto & f \\ 1 & \mapsto & a \\ 2 & \mapsto & b \\ 3 & \mapsto & g \\ 31 & \mapsto & a \\ 32 & \mapsto & b \end{array}$$

$$f(a, b, g(a, b))$$

Children are ordered. The path ϵ is called the root. Any path p such that $|t(p)| = 0$ is called a *leaf*.

Why is this formalism useful ?

Define the following concepts:

- The children of a path p

Why is this formalism useful ?

Define the following concepts:

- The children of a path p : $\mathrm{ch}(p) = \{pi \mid 1 \leq i \leq |t(p)|\}$ (this set is empty if p is a leaf).
- The parent of a path p

Why is this formalism useful ?

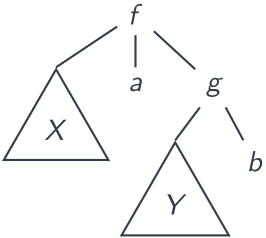
Define the following concepts:

- The children of a path p : $\mathrm{ch}(p) = \{pi \mid 1 \leq i \leq |t(p)|\}$ (this set is empty if p is a leaf).
- The parent of a path p : $\mathrm{par}(p)\{q \mid \exists i \in \mathbb{N}_1, qi = p\}$
  This set is
  - empty if $p = \epsilon$
  - a singleton otherwise
- All path ending in a: $\mathrm{lab}_a(t) = \{p \mid t(p) = a\}$

Trees with "holes" ?

In the case of words (e.g. in the pumping lemma), on can conveniently write $w = xyz$ to say that w has a prefix x a substring y and a suffix z.

The need also arises for trees, but it's a bit more complex:



Set of all trees

Definition (Set of all trees)

Let Σ be a ranked alphabet and $\mathcal{X}$ a set of 0-ary symbols called variables, such that $\Sigma \cap \mathcal{X} = \varnothing$. We call $\mathcal{T}(\Sigma, \mathcal{X})$ the set defined inductively by:

- $\forall a \in \Sigma_0, a \in \mathcal{T}(\Sigma, \mathcal{X})$
- $\forall x \in \mathcal{X}, x \in \mathcal{T}(\Sigma, \mathcal{X})$
- $\forall f \in \Sigma, \forall t_1, \ldots, t_{|f|} \in \mathcal{T}(\Sigma, \mathcal{X}), f(t_1, \ldots, t_{|f|}) \in \mathcal{T}(\Sigma, \mathcal{X})$

We shall write $\mathcal{T}(\Sigma)$ for the set $\mathcal{T}(\Sigma, \varnothing)$

Vocabulary

- a tree with variables is often called a term
- a tree with at most one occurrence of each variable is said to be *linear*
- a linear tree from $\mathcal{T}(\Sigma, \{\Box\})$ is called a *context* (it's a tree with a unique hole, a “place-holder” marked by $\Box$)

Subtree

An important notion is the one of subtree:

Definition (Subtree)

Let $t \in \mathcal{T}(\Sigma, \mathcal{X})$. For all $p \in Dom(t)$, we define the subtree of t rooted in p and we write $t|_p$ the tree defined by:

- $Dom(t|_p) = \{u \mid pu \in Dom(t)\}$
- $\forall u \in Dom(t|_p), t|_p(u) = t(pu)$

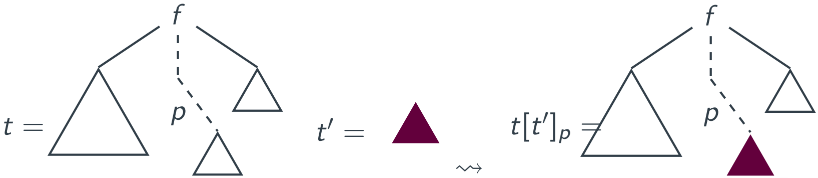
Subtree substitution

A common operation consists in replacing the subtree at a given path with another tree.

Definition (Subtree substitution)

Let $t, t' \in \mathcal{T}(\Sigma, \mathcal{X})$. *The tree* t *in which the subtree in* p *is replaced by* t', *written* $t[t']_p$ *is defined by:*

- $Dom(t[t']_p) = (Dom(t) \setminus \{pu \mid pu \in Dom(t)\}) \cup \{pu \mid u \in Dom(t')\}$
- $\forall q \in Dom(t[t']_p), t[t']_p(q) = \begin{cases} t'(u) & \forall u \text{ s.t. } q = pu \\ t(q) & \text{otherwise} \end{cases}$



Outline

Non-deterministic tree automaton

Definition (Non-deterministic tree automaton)

A non-deterministic bottom-up tree automaton or NTA is a $4-tuple$ $\mathcal{A} = (Q, \Sigma, \delta, F)$ *defined by*

- *a set of states* Q
- *a ranked alphabet* Σ
- *a set of accepting states* F
- *a transition function* $\delta : \mathcal{T}(\Sigma, Q) \rightarrow \mathcal{P}(Q)$ *of the form:*

$$f(q_{i_1}, \ldots, q_{i_k}) \mapsto q$$

with $f \in \Sigma$ *and* $|f| = k$*.*

Until otherwise specified, we will consider all automata to be bottom-up or ascending. We will come back to this aspects later.

Non-deterministic tree automaton(2)

The transition function δ takes as argument a symbol of Σ and a sequence of states which must hold on the corresponding children and returns a state for the current path. If δ always returns a singleton, the automaton is deterministic.

Run of a tree automaton

Definition

The run of an NTA $\mathcal{A} = (Q, \Sigma, \delta, F)$ *for a tree* $t \in \mathcal{T}(\Sigma)$ *is a function* $r : dom(t) \to Q$ *such that* $\forall p \in dom(t),\; r(p) \in \delta(t(p)(r(p1), \ldots, r(pk)))$*, with* $|t(p)| = k$*.*

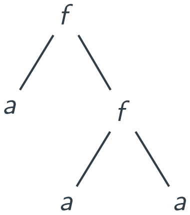
A run is accepting if $r(\epsilon) \in F$*.*

A set $L \subseteq \mathcal{T}(\Sigma)$ is a *regular tree language* if there exists a tree automaton that recognizes L.

Example

Consider the automaton $\mathcal{A} = (\{q_0, q_1\}, \{f^2, a\}, \delta, \{q_1\})$ which recognizes the "combs", that is, linear binary trees with leaf a and internal nodes f where each left subtree is a. The run for $t = f(a, f(a, a))$ is:

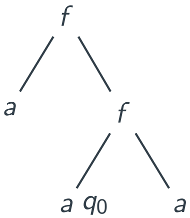
$$\delta : \quad \begin{array}{rcl} a & \mapsto & \{q_0\} \\ f(q_0, q_0) & \mapsto & \{q_1\} \\ f(q_0, q_1) & \mapsto & \{q_1\} \end{array}$$



Example

Consider the automaton $\mathcal{A} = (\{q_0, q_1\}, \{f^2, a\}, \delta, \{q_1\})$ which recognizes the “combs”, that is, linear binary trees with leaf a and internal nodes f where each left subtree is a. The run for $t = f(a, f(a, a))$ is:

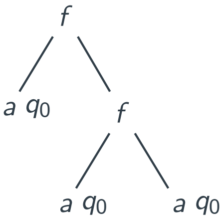
- $$\delta : \begin{array}{rcl} a & \mapsto & \{q_0\} \\ f(q_0, q_0) & \mapsto & \{q_1\} \\ f(q_0, q_1) & \mapsto & \{q_1\} \end{array}$$



Example

Consider the automaton $\mathcal{A} = (\{q_0, q_1\}, \{f^2, a\}, \delta, \{q_1\})$ which recognizes the "combs", that is, linear binary trees with leaf a and internal nodes f where each left subtree is a. The run for $t = f(a, f(a, a))$ is:

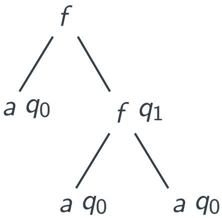
- $$\delta : \quad \begin{array}{rcl} a & \mapsto & \{q_0\} \\ f(q_0, q_0) & \mapsto & \{q_1\} \\ f(q_0, q_1) & \mapsto & \{q_1\} \end{array}$$



Example

Consider the automaton $\mathcal{A} = (\{q_0, q_1\}, \{f^2, a\}, \delta, \{q_1\})$ which recognizes the “combs”, that is, linear binary trees with leaf a and internal nodes f where each left subtree is a. The run for $t = f(a, f(a, a))$ is:

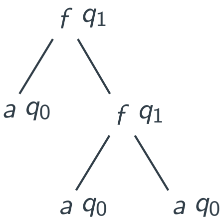
$$\delta : \quad \begin{array}{rcl} a & \mapsto & \{q_0\} \\ \blacktriangleright \quad f(q_0, q_0) & \mapsto & \{q_1\} \\ f(q_0, q_1) & \mapsto & \{q_1\} \end{array}$$



Example

Consider the automaton $\mathcal{A} = (\{q_0, q_1\}, \{f^2, a\}, \delta, \{q_1\})$ which recognizes the "combs", that is, linear binary trees with leaf a and internal nodes f where each left subtree is a. The run for $t = f(a, f(a, a))$ is:

$$\delta : \quad \begin{array}{rcl} a & \mapsto & \{q_0\} \\ f(q_0, q_0) & \mapsto & \{q_1\} \\ f(q_0, q_1) & \mapsto & \{q_1\} \end{array}$$



Example (2)

Tree automata "start" their run at the leaves. Thus, the transitions for leaves represent "initial states".

In the previous example, the automaton is incomplete. We can complete it as in the case of words:

$$\delta : \begin{array}{rcl} a & \mapsto & \{q_0\} \\ f(q_0, q_0) & \mapsto & \{q_1\} \\ f(q_0, q_1) & \mapsto & \{q_1\} \\ f(q_1, q_0) & \mapsto & \{q_\perp\} \end{array} \quad \begin{array}{rcl} f(q_0, q_\perp) & \mapsto & \{q_\perp\} \\ f(q_\perp, q_0) & \mapsto & \{q_\perp\} \\ f(q_1, q_\perp) & \mapsto & \{q_\perp\} \\ f(q_\perp, q_1) & \mapsto & \{q_\perp\} \\ f(q_\perp, q_\perp) & \mapsto & \{q_\perp\} \end{array}$$

Since the number of states to add is polynomial (need to consider all $f(\ldots, q_\perp, \ldots)$), we will often give incomplete automata.

Example (3)

Consider the NTA $\mathcal{A}_{\text{prop}} = (\{q_0, q_1\}, \{\vee^2, \wedge^2, \neg^1, \mathsf{F}, \mathsf{T}\}, \delta, \{q_1\})$ recognizing true statements in propositional logic:

$$\delta : \begin{array}{rcl} \mathsf{F} & \mapsto & \{q_0\} \\ \mathsf{T} & \mapsto & \{q_1\} \\ \neg(q_0) & \mapsto & \{q_1\} \\ \neg(q_1) & \mapsto & \{q_0\} \\ \wedge(q_0, q_0) & \mapsto & \{q_0\} \\ \wedge(q_0, q_1) & \mapsto & \{q_0\} \end{array} \qquad \begin{array}{rcl} \wedge(q_1, q_0) & \mapsto & \{q_0\} \\ \wedge(q_1, q_1) & \mapsto & \{q_1\} \\ \vee(q_1, q_1) & \mapsto & \{q_1\} \\ \vee(q_0, q_0) & \mapsto & \{q_0\} \\ \vee(q_1, q_0) & \mapsto & \{q_1\} \\ \vee(q_0, q_1) & \mapsto & \{q_1\} \end{array}$$

Example of non-determinism

The previous examples happen to be deterministic. We can adapt non-deterministic word automata to give some examples. Consider the automaton that recognizes trees on $\Sigma = \{f^2, g^2, \#\}$ whose root is $f(f(x,y),z)$ or $f(z,f(y,z))$, for arbitrary x, y, z in $\mathcal{T}(\Sigma)$: $(\{q_0, q_1, q_2\}, \Sigma, \delta, \{q_2\})$.

$$\delta : \begin{array}{rcl} \# & \mapsto & \{q_0\} \\ f(q_0, q_0) & \mapsto & \{q_0, q_1\} \\ g(q_0, q_0) & \mapsto & \{q_0\} \end{array} \quad \begin{array}{rcl} f(q_1, q_0) & \mapsto & \{q_2\} \\ f(q_0, q_1) & \mapsto & \{q_2\} \end{array}$$

Determinisation

Theorem

Let $\mathcal{A}$ be a non-deterministic tree automaton. There exists a deterministic tree automaton $\mathcal{A}_{det}$ which recognizes the same language.

The proof is the same as for the word case: states of the deterministic are sets of states of the NTA.

Determinisation (2)

université PARIS-SACLAY

Let $\mathcal{A} = (Q, \Sigma, \delta, F)$, define $\mathcal{A}_{\text{det}} = (Q_{\text{det}}, \Sigma, \delta_{\text{det}}, F_{\text{det}})$ where:

- $Q_{\text{det}} = \mathcal{P}(Q)$
- $F_{\text{det}} = \{s \in Q_{\text{det}} \mid s \cap F \neq \varnothing\}$
- $\delta_{\text{det}} : f(s_1, \ldots, s_k) \mapsto \{q \in \delta(f(q_1, \ldots, q_k)) \mid q_1 \in s_1, \ldots, q_k \in s_k \in\}$

Minimization

Theorem (Myhill-Nerode)

Let $\mathcal{A}$ a ***déterministic*** *tree automaton. There exists $\mathcal{A}_{min}$ with $L_{\mathcal{A}_{min}} = L_{\mathcal{A}}$ such that for all $\mathcal{A}'$, $|\mathcal{A}'| < |\mathcal{A}_{min}| \Rightarrow L_{\mathcal{A}'} \neq L_{\mathcal{A}_{min}}$*

Same algorithm as for words (Partition refinement).

Closure properties

Regular tree languages are closed under union, intersection and complement.

- Union: just merge both automata (union of set of states, union of final states, union of δs);
- Intersection: we can use the product construction, as for words.
- Complement: determinization, followed by completion, and invert accepting and non-accepting states.

Pumping lemma for trees

universite PARIS-SACLAY

Lemma (Pumping lemma)

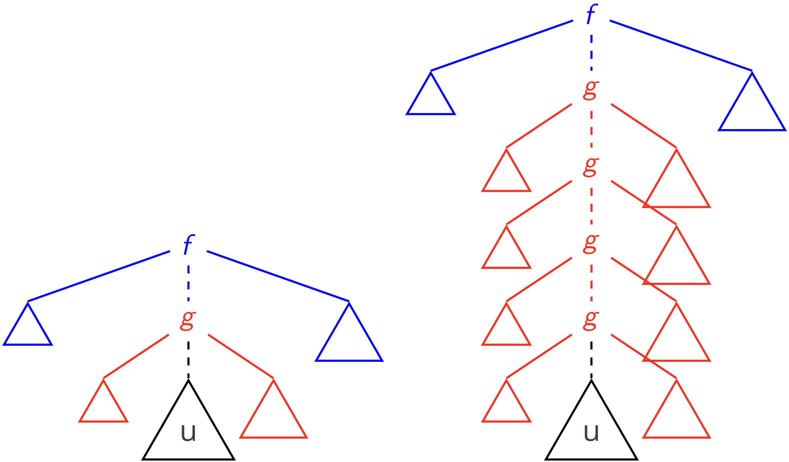
Let T be a regular tree language Σ. There exists $p \geq 1$ (pumping height) such that for all $t \in T$ such that $\mathsf{height}(t) \geq p$, there exists $C \in \mathcal{T}(\Sigma, \{\Box\})$, a non trivial context $C' \in \mathcal{T}(\Sigma, \{\Box\})$ and a tree $u \in \mathcal{T}(\Sigma)$ such that $t = C[C'[u]]$ and for all $n \geq 0$, $C[\underbrace{C'[\ldots[C'[u]]]}_{n \text{ times}}] \in T$.

The function $\mathsf{height} : \mathcal{T}(\Sigma) \to \mathbb{N}$ is defined inductively on $t \in \mathcal{T}(\Sigma)$ by:

- $\mathsf{height}(x) = 1, \forall x \in \Sigma_0$ (leafs have height 0)
- $\mathsf{height}(f(t_1, \ldots, t_k)) = 1 + \max\{\mathsf{hauteur}(t_i) \mid 1 \leq i \leq |f|\}$

The trivial context is $\Box$.

Pumping lemma in pictures



$C[\ ]$: context from the root

$C'[\ ]$: intermediary context $\Rightarrow$ part of the tree that can be pumped, while remaining in the language.

Outline

Tree traversal

In the word case, it seems natural to start from the begining of the word when computing a run. But we could go backward, starting from the last letter in an accepting state and computing the pre-image(s) of δ and require to finish the run in an initial state. In the case of tree, starting at the leaf does not seem natural. Can we “execute” the automaton while starting at the root?

Top-down tree automaton

Definition (Top-down tree automaton)

A non-deterministic top-down tree automaton, NTDTA is a 4-tuple $\mathcal{A} = (Q, \Sigma, \delta, I)$ defined by:

- *a set of states Q*
- *a ranked alphabet Σ*
- *a set of initial states I*
- *a transition function $\delta : Q \times \Sigma \to \mathcal{P}_f(Q^*)$ of the form:*

$$q, f \mapsto \{(q_{1_1}, \dots, q_{1_k}), \dots, (q_{m_1}, \dots, q_{m_k})\}$$

with $f \in \Sigma$ and $|f| = k$.

Top-down tree automaton (2)

The transition function δ take as argument a state and a symbol from Σ and tels in which states to continue the computation for every child of the current path.

If δ always returns a singleton, the automaton is deterministic.

Run of a top-down tree automaton

Definition

A run of a top-down tree automaton $\mathcal{A} = (Q, \Sigma, \delta, I)$ *for a tree* $t \in \mathcal{T}(\Sigma)$ *is a function* $r : dom(t) \to Q$ *such that*

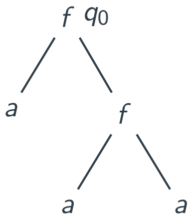
$\forall p \in dom(t),\ (r(p1), \ldots, r(pk)) \in \delta(r(p), t(p))$*, with* $|t(p)| = k$.

A run is accepting if $r(\epsilon) \in I$.

Example

Consider the NTDTA $\mathcal{A} = (\{q_0, q_1\}, \{f^2, a\}, \delta, \{q_0\})$ which (again) recognizes the "right combs" $\{f^2, a\}$. The run of $\mathcal{A}$ on $t = f(a, f(a, a))$ is:

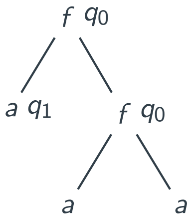
$$
\delta : \quad \begin{array}{rcl}
q_0, f & \mapsto & \{(q_1, q_0)\} \\
q_0, a & \mapsto & \{\} \\
q_1, a & \mapsto & \{\}
\end{array}
$$



Example

Consider the NTDTA $\mathcal{A} = (\{q_0, q_1\}, \{f^2, a\}, \delta, \{q_0\})$ which (again) recognizes the "right combs" $\{f^2, a\}$. The run of $\mathcal{A}$ on $t = f(a, f(a, a))$ is:

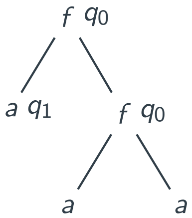
$$\delta : \begin{array}{lcl} q_0, f & \mapsto & \{(q_1, q_0)\} \\ q_0, a & \mapsto & \{\} \\ q_1, a & \mapsto & \{\} \end{array}$$



Example

Consider the NTDTA $\mathcal{A} = (\{q_0, q_1\}, \{f^2, a\}, \delta, \{q_0\})$ which (again) recognizes the "right combs" $\{f^2, a\}$. The run of $\mathcal{A}$ on $t = f(a, f(a, a))$ is:

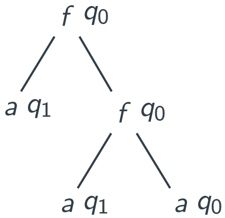
$$\delta : \begin{array}{rrcl} & q_0, f & \mapsto & \{(q_1, q_0)\} \\ & q_0, a & \mapsto & \{\} \\ \blacktriangleright & q_1, a & \mapsto & \{\} \end{array}$$



Example

Consider the NTDTA $\mathcal{A} = (\{q_0, q_1\}, \{f^2, a\}, \delta, \{q_0\})$ which (again) recognizes the “right combs” $\{f^2, a\}$. The run of $\mathcal{A}$ on $t = f(a, f(a, a))$ is:

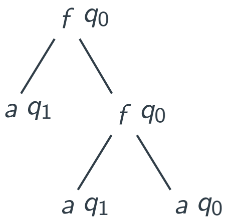
- $$\delta : \begin{array}{rcl} q_0, f & \mapsto & \{(q_1, q_0)\} \\ q_0, a & \mapsto & \{\} \\ q_1, a & \mapsto & \{\} \end{array}$$



Example

Consider the NTDTA $\mathcal{A} = (\{q_0, q_1\}, \{f^2, a\}, \delta, \{q_0\})$ which (again) recognizes the "right combs" $\{f^2, a\}$. The run of $\mathcal{A}$ on $t = f(a, f(a, a))$ is:

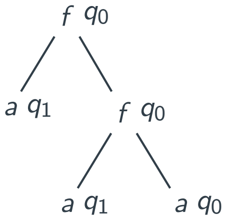
$$
\delta : \quad
\begin{array}{lcl}
q_0, f & \mapsto & \{(q_1, q_0)\} \\
q_0, a & \mapsto & \{\} \\
q_1, a & \mapsto & \{\}
\end{array}
$$



Example

Consider the NTDTA $\mathcal{A} = (\{q_0, q_1\}, \{f^2, a\}, \delta, \{q_0\})$ which (again) recognizes the “right combs” $\{f^2, a\}$. The run of $\mathcal{A}$ on $t = f(a, f(a, a))$ is:

$$\delta : \quad \blacktriangleright \quad \begin{array}{lcl} q_0, f & \mapsto & \{(q_1, q_0)\} \\ q_0, a & \mapsto & \{\} \\ q_1, a & \mapsto & \{\} \end{array}$$



Bottom-up vs Top-down

Theorem (Equivalence NTDTA-NTA)

The set of languages recognized by non-deterministict bottom-up tree automata is exactly the set of languages recognized by non-deterministic top-down tree automata.

Proof: it is easy to construct one from the other. Switch accepting and initial states, and read the transitions backwards.

Deterministic top-down tree automata

université PARIS-SACLAY

Theorem (Non equivalence of DTDTA and NTA)

The set of languages recognized by deterministic top-down tree automata is a strict subset of regular tree languages.

in other words: $TD_{\text{det}} \subsetneq TD_{\text{nondet}} = BU_{\text{nondet}} = BU_{\text{det}}$

Counter example

Consider $\Sigma = \{f^2, a, b\}$. The language $\{f(a, b), f(b, a)\}$ is **not** recognizable by a deterministic top-down tree automaton. Indeed, if we start the automaton in some state q_0 on a symbol f, it must recognize the left subtree in some state q_1 and the right subtree in some state q_2. Therefore, q_1 must recognize a or b (since the automaton is deterministic, $q_0, f \mapsto \{(q_1, q_2)\}$ is the only choice). Likewise, q_2 must recognize a or b. But in that case, the automaton also accepts $f(a, a)$ and $f(b, b)$ which are not in the initial language.

Intuitively, a deterministic top-down tree automaton must decide in which state to explore the subtree with only the knowledge of the path taken up to the current node.

(partial) conclusion

We have seen a new formalism, tree automata.

They recognize regular tree languages.

Most results transfer from the word case (except the determinization, which does not hold for NTDTA).

We still need a high-level language (next week)!