Language, Automata and Logic for Finite Trees

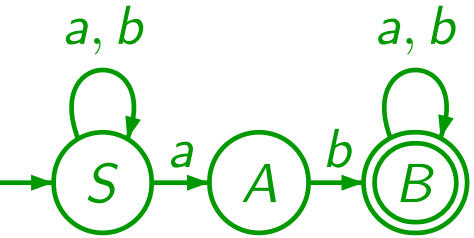
Olivier Gauwin

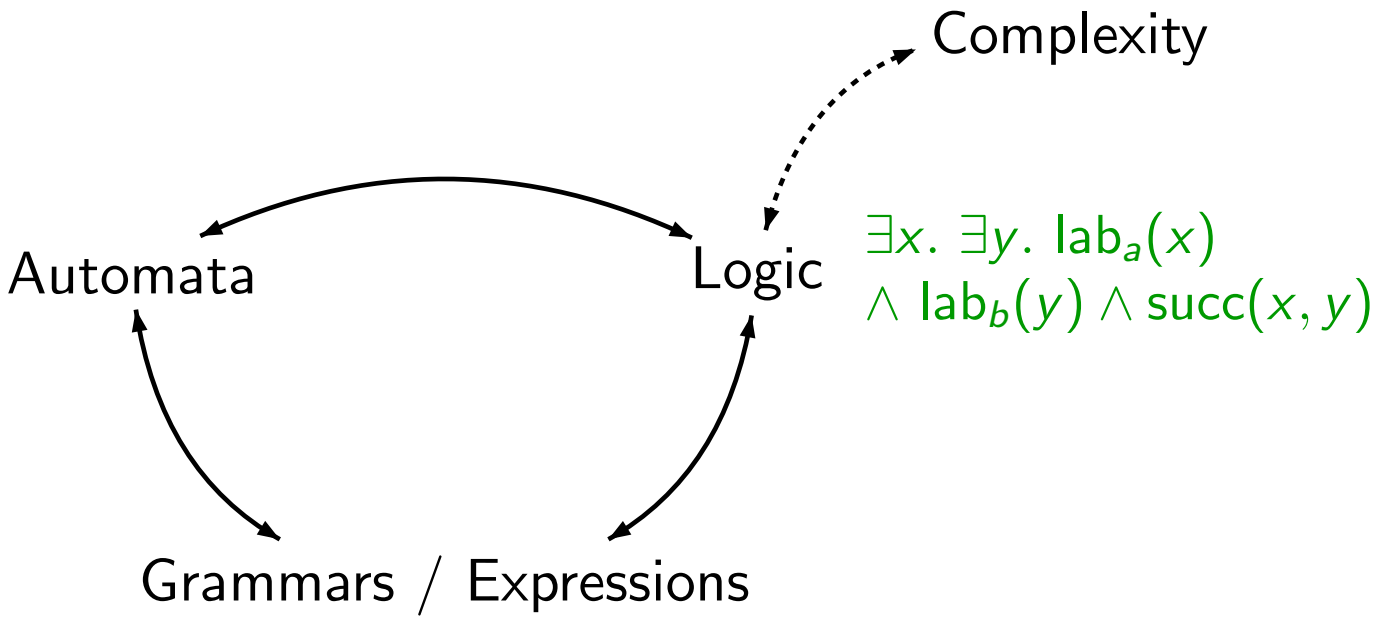
UMons

Feb/March 2010

Languages, Automata, Logic

Example for regular word languages: $\Sigma^*.a.b.\Sigma^*$





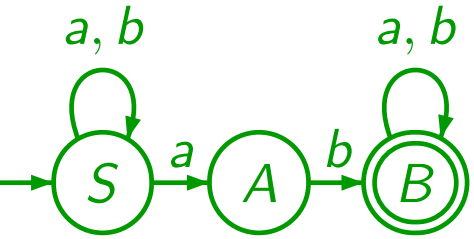
$\exists x.\ \exists y.\ \mathrm{lab}_a(x) \wedge \mathrm{lab}_b(y) \wedge \mathrm{succ}(x, y)$

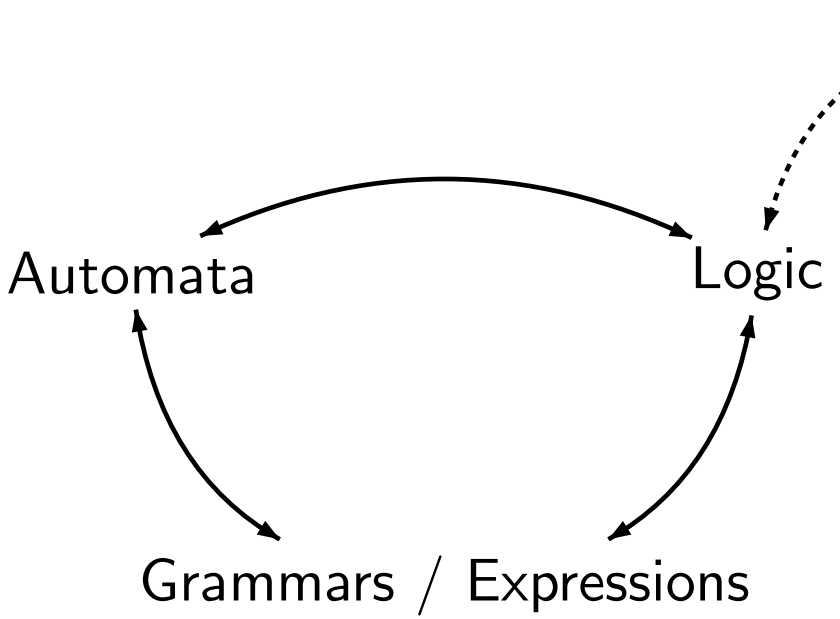
$$\begin{cases} S \to aS & S \to aB & X \to aX & X \to \epsilon \\ S \to bS & B \to bX & X \to bX & \end{cases}$$

Languages, Automata, Logic

Example for regular word languages: $\Sigma^*.a.b.\Sigma^*$

Example for regular tree languages: all trees with an a-node having a b-child





$\exists x.\ \exists y.\ \mathrm{lab}_a(x) \wedge \mathrm{lab}_b(y) \wedge \mathrm{succ}(x, y)$

$\exists x.\ \exists y.\ \mathrm{lab}_a(x) \wedge \mathrm{lab}_b(y) \wedge \mathrm{child}(x, y)$

$$\begin{cases} \_(S, X) \to S \\ \_(X, S) \to S \\ a(B, X) \to S \\ a(X, B) \to S \\ \dots \\ b \to B \\ \dots \end{cases}$$

$$\begin{cases} S \to aS & S \to aB & X \to aX & X \to \epsilon \\ S \to bS & B \to bX & X \to bX \end{cases}$$

$$\begin{cases} S \to \_(S, X) & S \to a(B, X) & B \to b(X, X) & X \to \_(X, X) \\ S \to \_(X, S) & S \to a(X, B) & B \to b & X \to \_ \end{cases}$$

References

The main reference for this talk is the *TATA* book [CDG+07]:

Tree Automata, Techniques and Applications

by

Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugier, Sophie Tison, Marc Tommasi.

Other references will be mentionned progressively.

1. Ranked Trees
   - Trees on Ranked Alphabet
   - Tree Automata
   - Tree Grammars
   - Logic

2. Unranked Trees
   - Unranked Trees
   - Automata
   - Logic

Trees on Ranked Alphabet

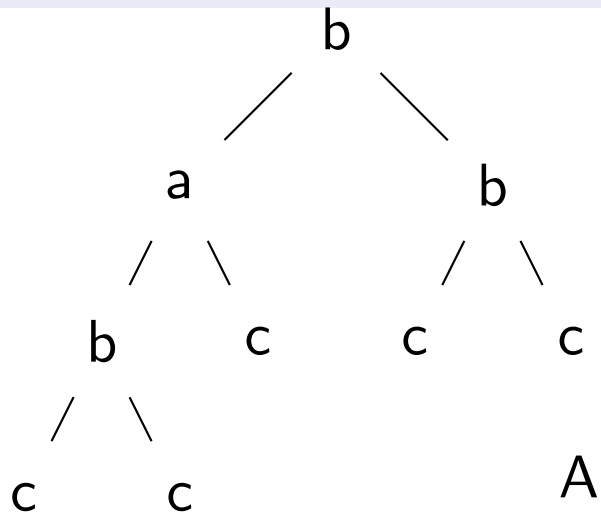
Ranked alphabet

Ranked alphabet Σ_r = finite alphabet $\Sigma = \{a, b, c\}$ + arity function $\mathrm{ar} : \Sigma \to \mathbb{N}$

$\mathrm{ar}(a) = 2$
$\mathrm{ar}(b) = 2$
$\mathrm{ar}(c) = 0$

Ranked trees over Σ_r

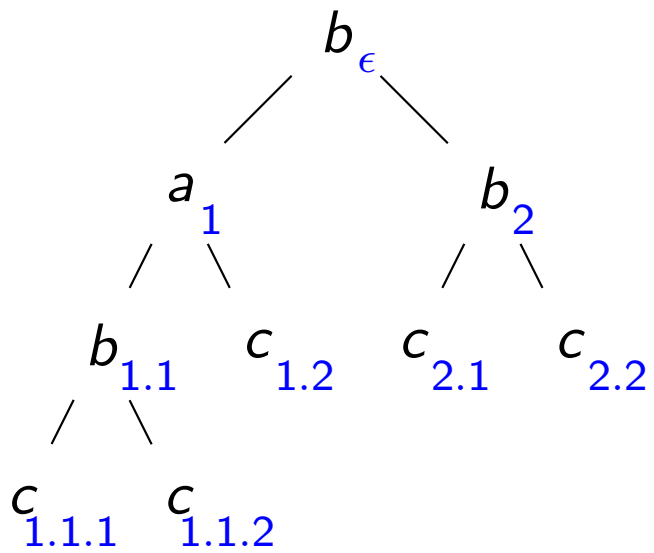
$\mathcal{T}_{\Sigma_r}$, the set of ranked trees, is the smallest set of terms $f(t_1, \ldots, t_k)$ such that: $f \in \Sigma_r$, $k = \mathrm{ar}(f)$, and $t_i \in \mathcal{T}_{\Sigma_r}$ for all $1 \leq i \leq k$.



A tree language T is a set of trees: $T \subseteq \mathcal{T}_{\Sigma_r}$.

Trees as relational structures

We will sometimes consider a ranked tree t as a relational structure $(\text{nodes}^t, \{\text{lab}_a^t, \text{lab}_b^t, \text{lab}_c^t, ch_1^t, ch_2^t\})$.



- $\text{lab}_a^t = \{1\}$
- $\text{lab}_b^t = \{\epsilon, 1.1, 2\}$
- $\text{lab}_c^t = \{1.1.1, 1.1.2, 1.2, 2.1, 2.2\}$
- $ch_1^t = \{(\epsilon, 1), (1, 1.1), (2, 2.1) \dots\}$
- $ch_2^t = \{(\epsilon, 2), (1, 1.2), (2, 2.2) \dots\}$

For convenience we write $\text{lab}(\pi)$ for the label of node π.

Tree Automata

over Ranked Trees

Definition

A tree automaton (TA) over $\Sigma_r = (\Sigma, \mathrm{ar})$ is a tuple $A = (Q, F, \Delta, \Sigma_r)$ where:

- Q is a finite set of states,
- $F \subseteq Q$ a set of final states,
- Δ are rules of type: $a(q_1, \ldots, q_k) \to q$

  with $a \in \Sigma$, $k = \mathrm{ar}(a)$ and $q, q_1, \ldots, q_k \in Q$

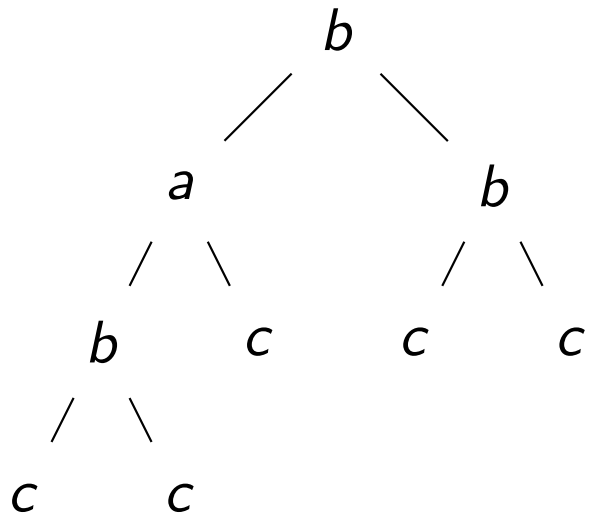
Runs

A run ρ of A on t is a function $\rho : \mathrm{nodes}^t \to Q$ such that:
if $\pi \in \mathrm{nodes}^t$ with children $\pi_1, \ldots, \pi_k$ and label a then
$a(\rho(\pi_1), \ldots, \rho(\pi_k)) \to \rho(\pi) \in \Delta$

Bottom-up vs top-down

Bottom-up view

$a(q_1, \ldots, q_k) \to q$ is a bottom-up point of view:



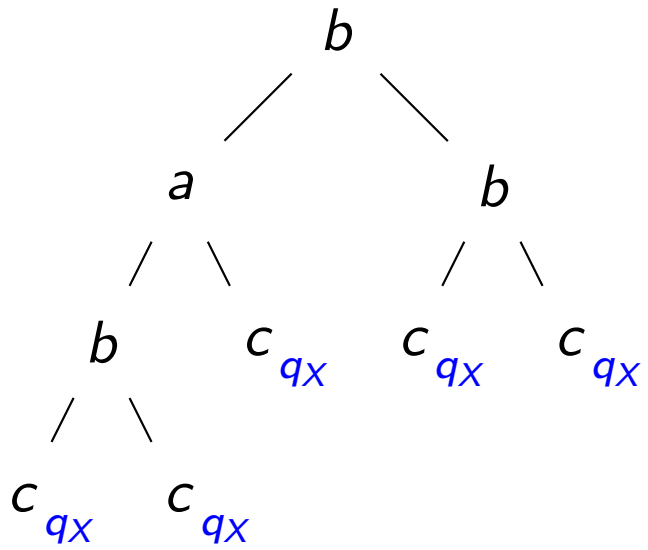
Rules:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

Bottom-up vs top-down

Bottom-up view

$a(q_1, \ldots, q_k) \to q$ is a bottom-up point of view:



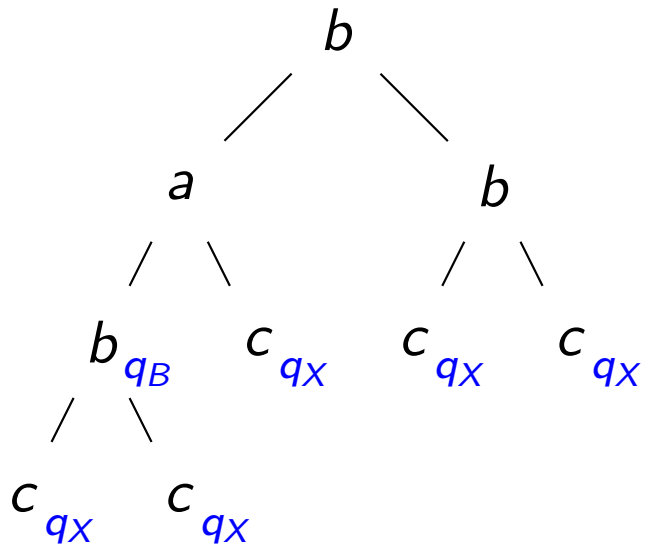
Rules:

$$
\begin{cases}
a(q_S, q_X) \to q_S \\
a(q_X, q_S) \to q_S \\
b(q_S, q_X) \to q_S \\
b(q_X, q_S) \to q_S \\
a(q_B, q_X) \to q_S \\
a(q_X, q_B) \to q_S \\
b(q_X, q_X) \to q_B \\
a(q_X, q_X) \to q_X \\
b(q_X, q_X) \to q_X \\
c \to q_X
\end{cases}
$$

Bottom-up vs top-down

Bottom-up view

$a(q_1, \ldots, q_k) \to q$ is a bottom-up point of view:



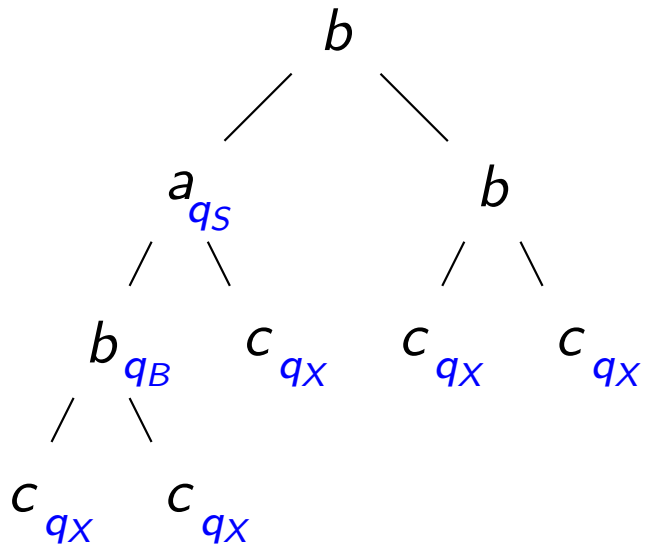
Rules:

$$
\begin{cases}
a(q_S, q_X) \to q_S \\
a(q_X, q_S) \to q_S \\
b(q_S, q_X) \to q_S \\
b(q_X, q_S) \to q_S \\
a(q_B, q_X) \to q_S \\
a(q_X, q_B) \to q_S \\
b(q_X, q_X) \to q_B \\
a(q_X, q_X) \to q_X \\
b(q_X, q_X) \to q_X \\
c \to q_X
\end{cases}
$$

Bottom-up vs top-down

Bottom-up view

$a(q_1, \ldots, q_k) \to q$ is a bottom-up point of view:



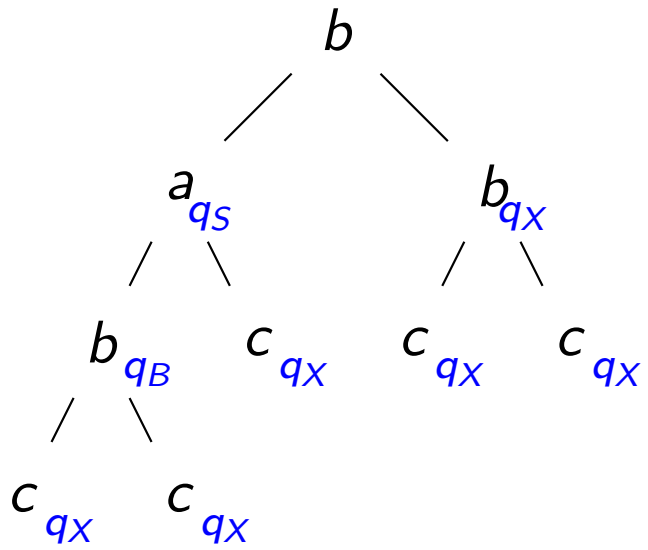
Rules:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

Bottom-up vs top-down

Bottom-up view

$a(q_1, \ldots, q_k) \to q$ is a bottom-up point of view:



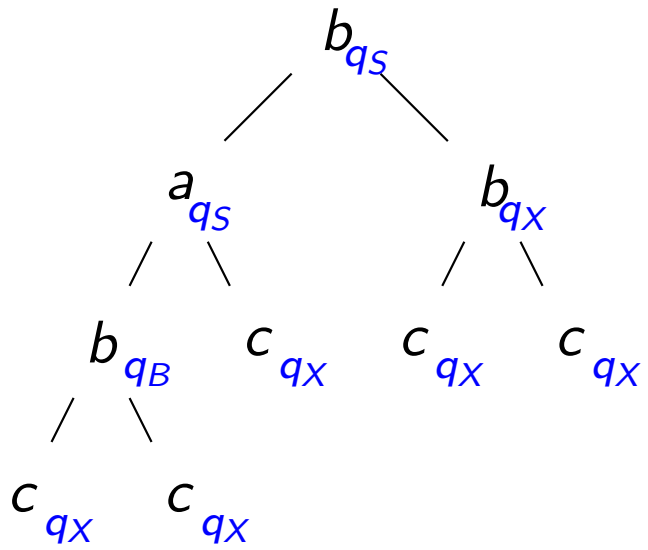
Rules:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

Bottom-up vs top-down

Bottom-up view

$a(q_1, \ldots, q_k) \to q$ is a bottom-up point of view:



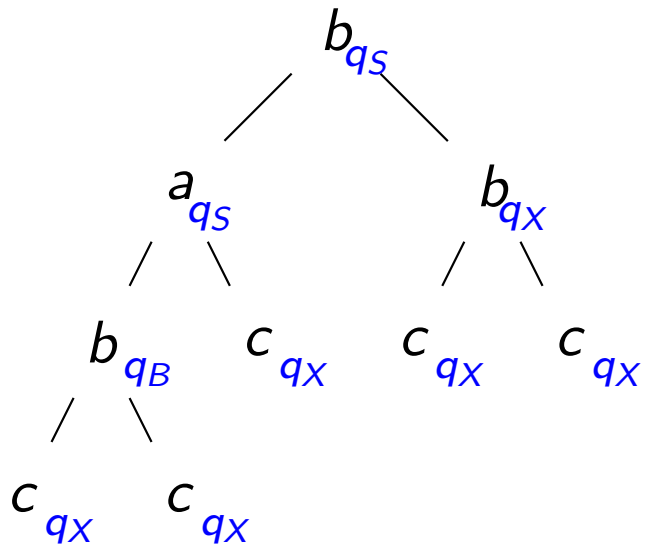
Rules:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

Bottom-up vs top-down

Bottom-up view

$a(q_1, \ldots, q_k) \to q$ is a bottom-up point of view:



Rules:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

A run of A on t is accepting if $\rho(\epsilon) \in F$.

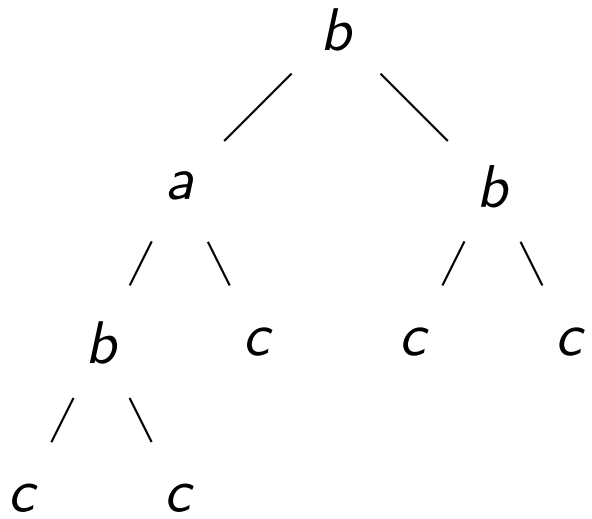
$\mathcal{L}(A) = \{t \mid \text{there exists an accepting run } \rho \text{ of } A \text{ on } t\}$

Bottom-up vs top-down

Top-down view

We could have written rules this way: $a(q) \to (q_1, \ldots, q_k)$, and name F the initial states.

This corresponds to a top-down definition:



Initial: $\{q_S\}$

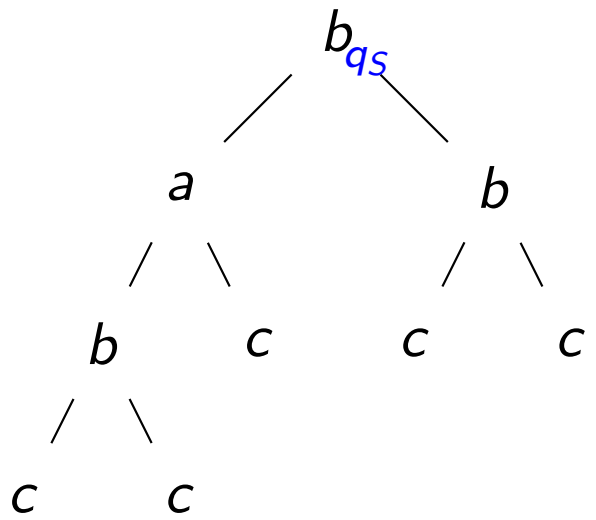
Rules:
$$\begin{cases} a(q_S) \to (q_S, q_X) \\ a(q_S) \to (q_X, q_S) \\ b(q_S) \to (q_S, q_X) \\ b(q_S) \to (q_X, q_S) \\ a(q_S) \to (q_B, q_X) \\ a(q_S) \to (q_X, q_B) \\ b(q_B) \to (q_X, q_X) \\ a(q_X) \to (q_X, q_X) \\ b(q_X) \to (q_X, q_X) \\ c(q_X) \end{cases}$$

Bottom-up vs top-down

Top-down view

We could have written rules this way: $a(q) \to (q_1, \ldots, q_k)$, and name F the initial states.

This corresponds to a top-down definition:



Initial: $\{q_S\}$

Rules:

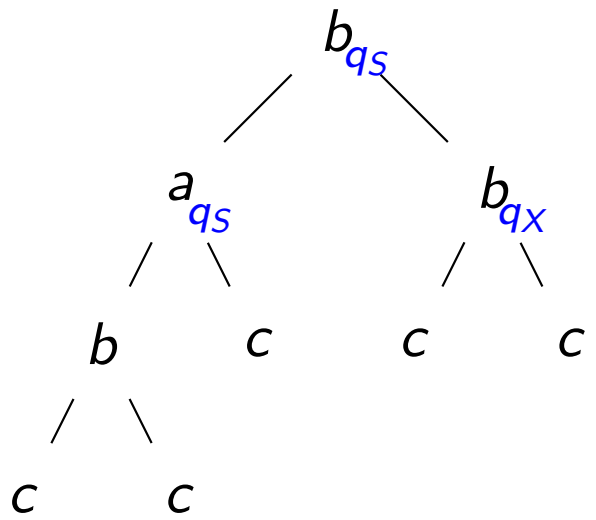
$$\begin{cases} a(q_S) \to (q_S, q_X) \\ a(q_S) \to (q_X, q_S) \\ b(q_S) \to (q_S, q_X) \\ b(q_S) \to (q_X, q_S) \\ a(q_S) \to (q_B, q_X) \\ a(q_S) \to (q_X, q_B) \\ b(q_B) \to (q_X, q_X) \\ a(q_X) \to (q_X, q_X) \\ b(q_X) \to (q_X, q_X) \\ c(q_X) \end{cases}$$

Bottom-up vs top-down

Top-down view

We could have written rules this way: $a(q) \to (q_1, \ldots, q_k)$, and name F the initial states.

This corresponds to a top-down definition:



Initial: $\{q_S\}$

Rules:

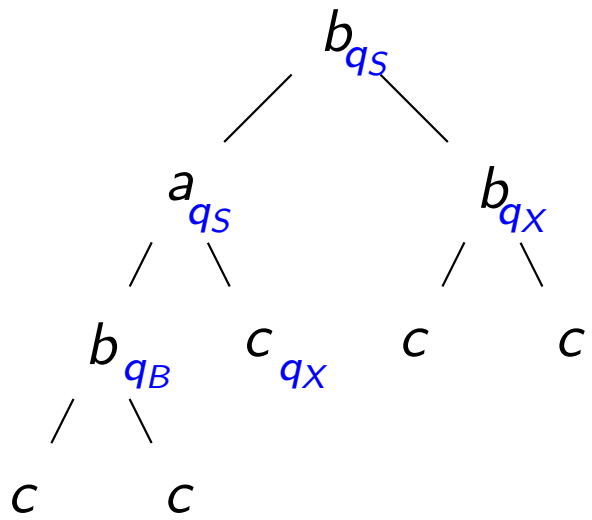
$$\begin{cases} a(q_S) \to (q_S, q_X) \\ a(q_S) \to (q_X, q_S) \\ b(q_S) \to (q_S, q_X) \\ b(q_S) \to (q_X, q_S) \\ a(q_S) \to (q_B, q_X) \\ a(q_S) \to (q_X, q_B) \\ b(q_B) \to (q_X, q_X) \\ a(q_X) \to (q_X, q_X) \\ b(q_X) \to (q_X, q_X) \\ c(q_X) \end{cases}$$

Bottom-up vs top-down

Top-down view

We could have written rules this way: $a(q) \to (q_1, \ldots, q_k)$, and name F the initial states.

This corresponds to a top-down definition:



Initial: $\{q_S\}$

Rules:

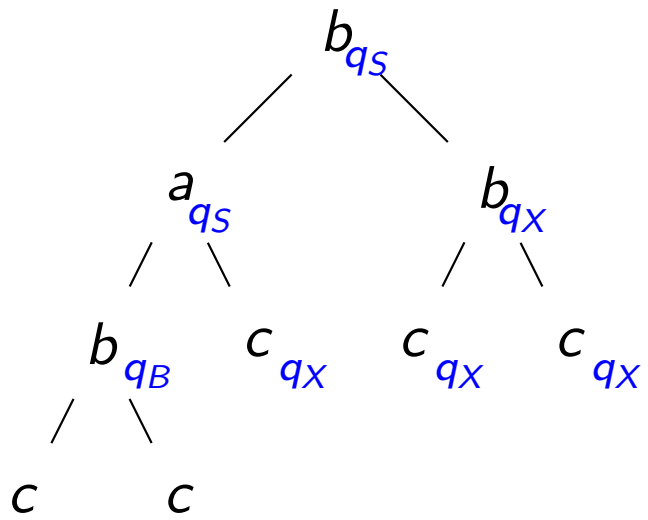
$$\begin{cases} a(q_S) \to (q_S, q_X) \\ a(q_S) \to (q_X, q_S) \\ b(q_S) \to (q_S, q_X) \\ b(q_S) \to (q_X, q_S) \\ a(q_S) \to (q_B, q_X) \\ a(q_S) \to (q_X, q_B) \\ b(q_B) \to (q_X, q_X) \\ a(q_X) \to (q_X, q_X) \\ b(q_X) \to (q_X, q_X) \\ c(q_X) \end{cases}$$

Bottom-up vs top-down

Top-down view

We could have written rules this way: $a(q) \to (q_1, \ldots, q_k)$, and name F the initial states.

This corresponds to a top-down definition:



Initial: $\{q_S\}$

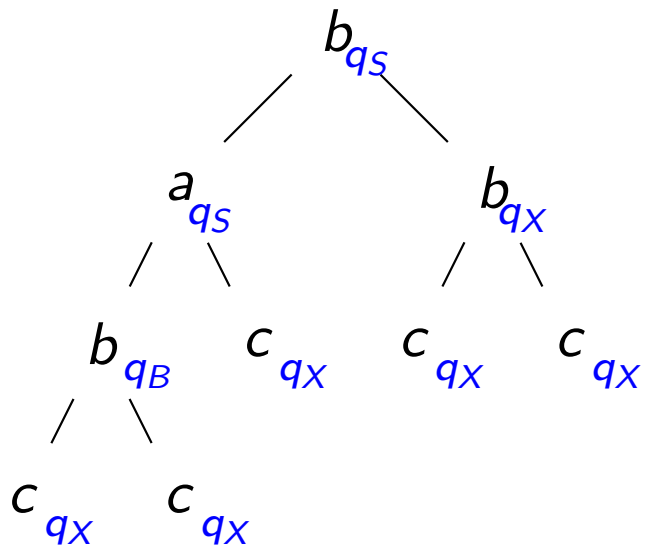
Rules:
$$\begin{cases} a(q_S) \to (q_S, q_X) \\ a(q_S) \to (q_X, q_S) \\ b(q_S) \to (q_S, q_X) \\ b(q_S) \to (q_X, q_S) \\ a(q_S) \to (q_B, q_X) \\ a(q_S) \to (q_X, q_B) \\ b(q_B) \to (q_X, q_X) \\ a(q_X) \to (q_X, q_X) \\ b(q_X) \to (q_X, q_X) \\ c(q_X) \end{cases}$$

Bottom-up vs top-down

Top-down view

We could have written rules this way: $a(q) \to (q_1, \ldots, q_k)$, and name F the initial states.

This corresponds to a top-down definition:



Initial: $\{q_S\}$

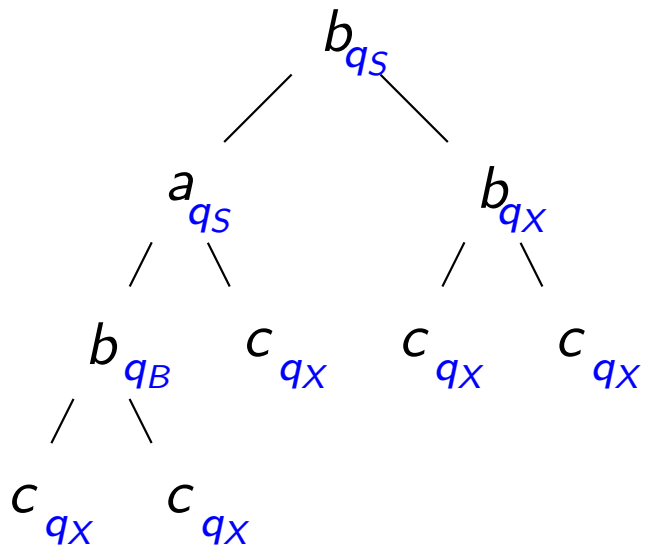
Rules:
$$\begin{cases} a(q_S) \to (q_S, q_X) \\ a(q_S) \to (q_X, q_S) \\ b(q_S) \to (q_S, q_X) \\ b(q_S) \to (q_X, q_S) \\ a(q_S) \to (q_B, q_X) \\ a(q_S) \to (q_X, q_B) \\ b(q_B) \to (q_X, q_X) \\ a(q_X) \to (q_X, q_X) \\ b(q_X) \to (q_X, q_X) \\ c(q_X) \end{cases}$$

Bottom-up vs top-down

Top-down view

We could have written rules this way: $a(q) \to (q_1, \ldots, q_k)$, and name F the initial states.

This corresponds to a top-down definition:



Initial: $\{q_S\}$

Rules:
$$\begin{cases} a(q_S) \to (q_S, q_X) \\ a(q_S) \to (q_X, q_S) \\ b(q_S) \to (q_S, q_X) \\ b(q_S) \to (q_X, q_S) \\ a(q_S) \to (q_B, q_X) \\ a(q_S) \to (q_X, q_B) \\ b(q_B) \to (q_X, q_X) \\ a(q_X) \to (q_X, q_X) \\ b(q_X) \to (q_X, q_X) \\ c(q_X) \end{cases}$$

Bottom-up vs top-down

Comparison

These definitions of runs coincide: a bottom-up run exists iff a top-down run exists, and they are strictly the same:

$$\text{bottom-up TA } (\uparrow\text{TA}) = \text{top-down TA } (\downarrow\text{TA}) = \text{“TA”}$$

Bottom-up vs top-down

Comparison

These definitions of runs coincide: a bottom-up run exists iff a top-down run exists, and they are strictly the same:

$$\text{bottom-up TA } (\uparrow\text{TA}) = \text{top-down TA } (\downarrow\text{TA}) = \text{“TA”}$$

★★★

However, notions of determinism differ!

$$\text{det. } \downarrow\text{TA } (\text{d}\downarrow\text{TA }) \neq \text{det. } \uparrow\text{TA } (\text{d}\uparrow\text{TA })$$

For instance: $\{f(a,b), f(b,a)\}$ can be recognized by a det. bottom-up TA, but any det. top-down TA accepting $f(a,b)$ and $f(b,a)$ would recognize $f(a,a)$.

Determinization

of ↑TA

Proposition

For every ↑TA A, there exists a d↑TA A_d such that $\mathcal{L}(A) = \mathcal{L}(A_d)$.

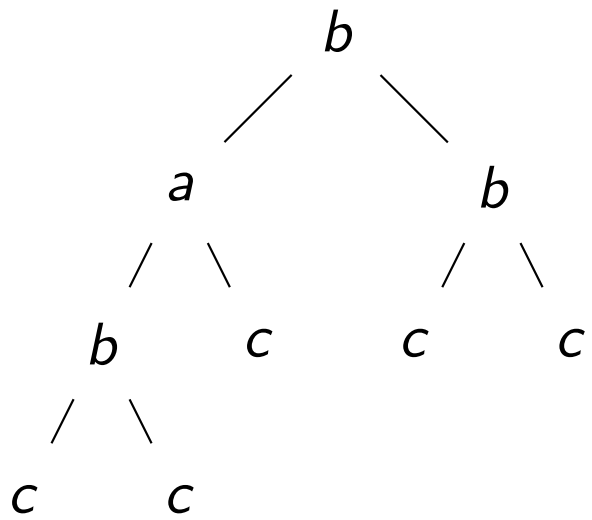
a subset construction, very similar to the determinization of NFAs:

- $Q_d = 2^Q$
- $\Delta_d = \{a(s_1, \ldots, s_n) \to s \mid$
  $s = \{ q \in Q \mid \exists q_1 \in s_1, \ldots, \exists q_n \in s_n,\ f(q_1, \ldots, q_n) \to q \in \Delta\}\}$
- $F_d = \{S \mid S \subseteq Q \text{ and } S \cap F \neq \emptyset\}$

This simulates all runs of A.

Determinization

Example run



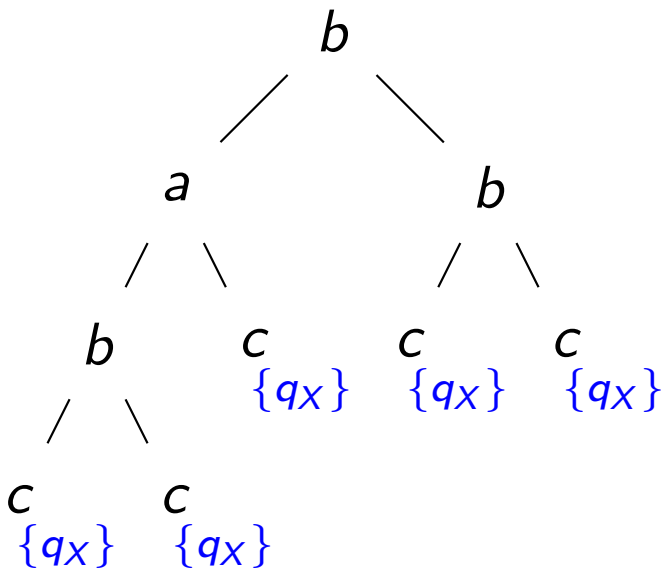
$$F = \{q_S\}$$

Rules of $\uparrow$TA:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

Determinization

Example run



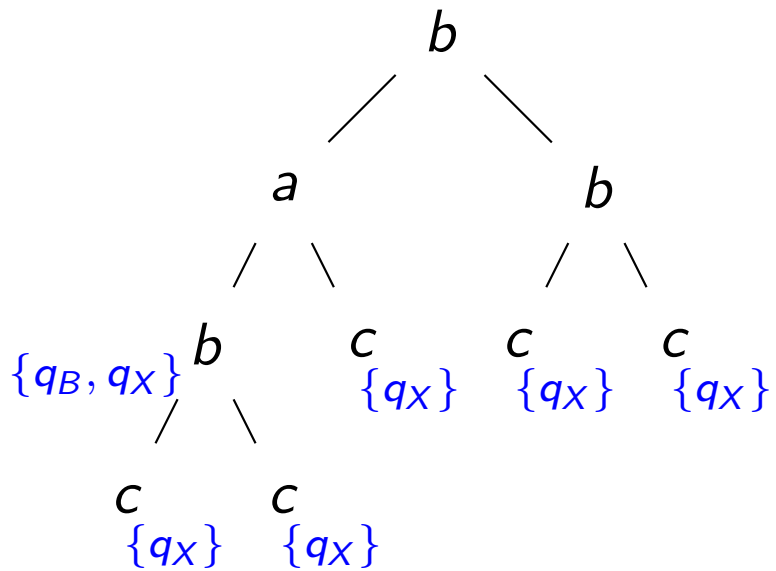
$F = \{q_S\}$

Rules of $\uparrow$TA:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

Determinization

Example run



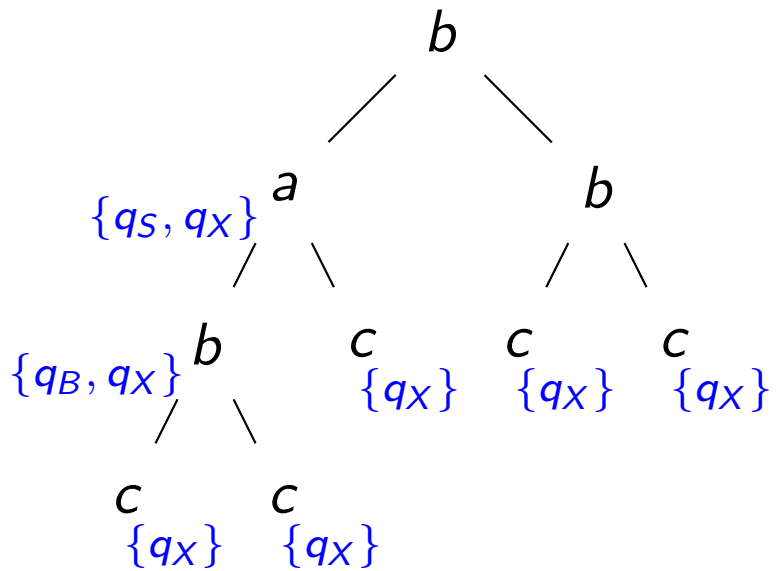
$F = \{q_S\}$

Rules of $\uparrow$TA:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

Determinization

Example run



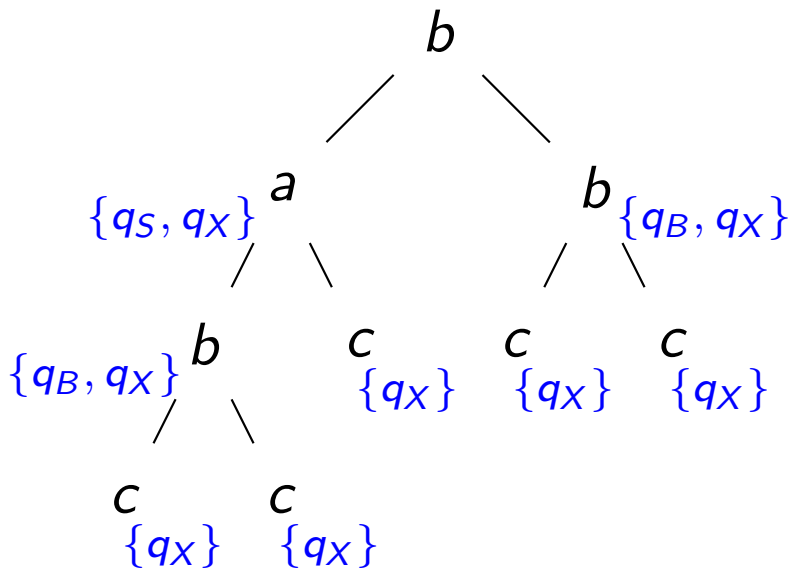
$F = \{q_S\}$

Rules of $\uparrow$TA:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

Determinization

Example run



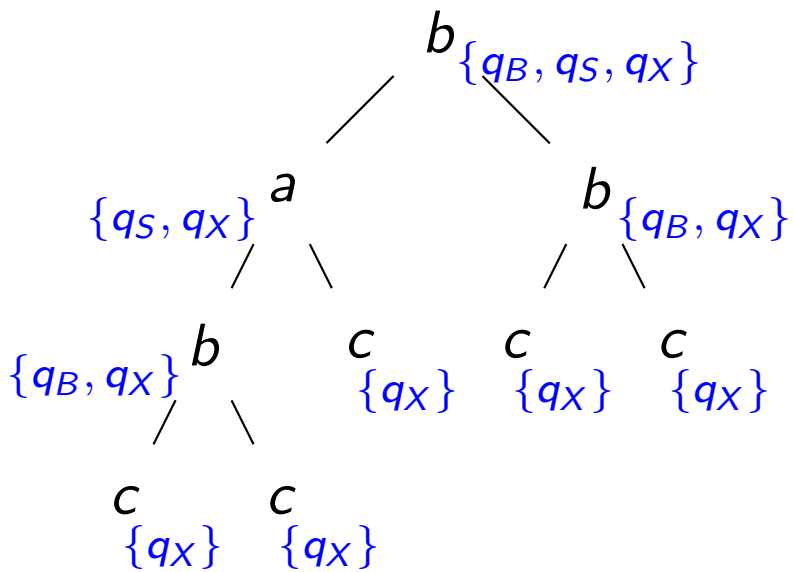
$F = \{q_S\}$

Rules of $\uparrow$TA:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

Determinization

Example run



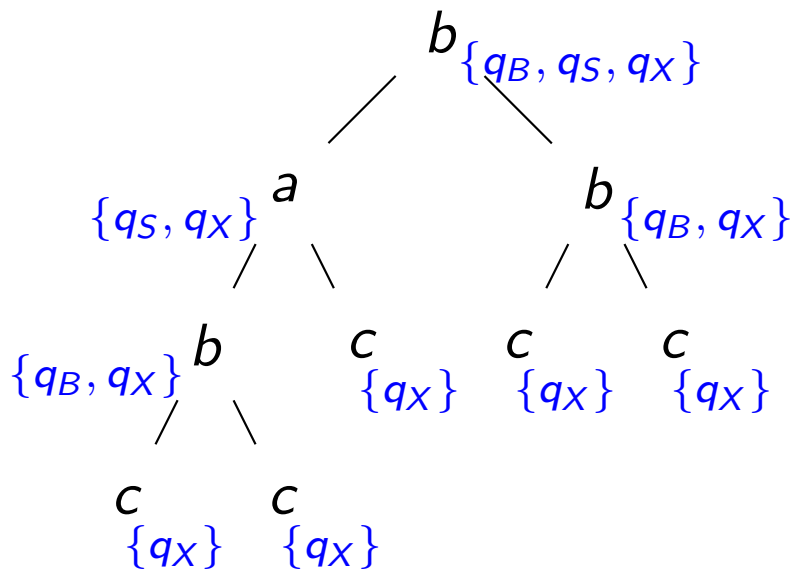
$F = \{q_S\}$

Rules of $\uparrow$TA:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

Determinization

Example run



$F = \{q_S\}$

Rules of $\uparrow$TA:

$$\begin{cases} a(q_S, q_X) \to q_S \\ a(q_X, q_S) \to q_S \\ b(q_S, q_X) \to q_S \\ b(q_X, q_S) \to q_S \\ a(q_B, q_X) \to q_S \\ a(q_X, q_B) \to q_S \\ b(q_X, q_X) \to q_B \\ a(q_X, q_X) \to q_X \\ b(q_X, q_X) \to q_X \\ c \to q_X \end{cases}$$

$\{q_B, q_S, q_X\} \cap F \neq \emptyset$ so this tree is accepted by A_d

TA classes

$$\mathsf{d}{\uparrow}\mathsf{TA} = {\uparrow}\mathsf{TA}$$

TA classes

$$\mathsf{d}{\uparrow}\mathsf{TA} = {\uparrow}\mathsf{TA} = {\downarrow}\mathsf{TA}$$

TA classes

$$\mathsf{d}{\downarrow}\mathsf{TA} \subsetneq \mathsf{d}{\uparrow}\mathsf{TA} = {\uparrow}\mathsf{TA} = {\downarrow}\mathsf{TA}$$

TA classes

$$\mathrm{d}{\downarrow}\mathrm{TA} \subsetneq \mathrm{d}{\uparrow}\mathrm{TA} = {\uparrow}\mathrm{TA} = {\downarrow}\mathrm{TA}$$

Definition

A ranked tree language $L \subseteq \mathcal{T}_{\Sigma_r}$ is recognizable if there is a $\uparrow$TA recognizing L.

Closure Properties

of recognizable tree languages

If L_1 and L_2 are recognizable tree languages, then:

$\overline{L_1}$ is recognizable

if $A = (Q, F, \Delta, \Sigma_r)$ is a complete $\uparrow$TA, then $A' = (Q, Q \setminus F, \Delta, \Sigma_r)$ recognizes $\overline{\mathcal{L}(A)}$.
Completing is easy, by adding a sink state.

$L_1 \cup L_2$ is recognizable

$L_1 \cap L_2$ is recognizable

Closure Properties

of recognizable tree languages

If L_1 and L_2 are recognizable tree languages, then:

$\overline{L_1}$ is recognizable

$L_1 \cup L_2$ is recognizable

Let $\begin{cases} A_1 = (Q_1, F_1, \Delta_1, \Sigma_r) \text{ be a complete } \uparrow\text{TA recognizing } L_1 \\ A_2 = (Q_2, F_2, \Delta_2, \Sigma_r) \text{ be a complete } \uparrow\text{TA recognizing } L_2 \end{cases}$

We build the product automaton

$A_1 \times A_2 = (Q_1 \times Q_2, F_1 \times Q_2 \cup Q_1 \times F_2, \Delta', \Sigma_r)$ with

$$\frac{a(q_1, \ldots, q_n) \to q \in \Delta_1 \qquad a(q'_1, \ldots, q'_n) \to q' \in \Delta_2}{a((q_1, q'_1), \ldots, (q_n, q'_n)) \to (q, q') \in \Delta'}$$

Then $\mathcal{L}(A_1 \times A_2) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$.

$L_1 \cap L_2$ is recognizable

Closure Properties

of recognizable tree languages

If L_1 and L_2 are recognizable tree languages, then:

$\overline{L_1}$ is recognizable

$L_1 \cup L_2$ is recognizable

$L_1 \cap L_2$ is recognizable

$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ (but more efficient using another product construction)

Pumping Lemma

Context $=$ tree over $\Sigma_r \uplus \{\Box\}$ where $\begin{cases} \mathrm{ar}(\Box) = 0 \text{ and} \\ \Box \text{ appears exactly once.} \end{cases}$

Intuitively, *context = tree with a hole*.

(context C) $+$ (tree t) $=$ a new tree $C[t]$, where t replaces $\Box$ in C.

Pumping Lemma

Context $=$ tree over $\Sigma_r \uplus \{\Box\}$ where $\begin{cases} \mathrm{ar}(\Box) = 0 \text{ and} \\ \Box \text{ appears exactly once.} \end{cases}$

Intuitively, *context = tree with a hole*.

(context C) $+$ (tree t) $=$ a new tree $C[t]$, where t replaces $\Box$ in C.

Pumping lemma

Let L be a *recognizable* tree language. Then there exists $k > 0$ such that for every tree $t \in L$ of depth $> k$, there exist contexts C_1, C_2 (with $C_2 \neq \Box$) and a tree t' such that $\begin{cases} t = C_1[C_2[t']] \\ \forall n \geq 0.\ C_1[C_2^n[t']] \in L \end{cases}$

idea: for A s.t. $\mathcal{L}(A) = L$, take $k = |Q|$ and consider a branch of length k.

Tree homomorphisms

$$\text{Tree homomorphism} = \begin{cases} \text{transformation } h : \mathcal{T}_\Sigma \to \mathcal{T}_{\Sigma'} \\ \text{based on a mapping } m : \Sigma \to \mathcal{T}_{\Sigma' \cup \mathcal{X}_{\text{ar}(m)}} \end{cases}$$

Theorem

$$\left.\begin{array}{l} L\text{: recognizable tree language in } \mathcal{T}_\Sigma \\ h\text{: linear tree homom. } h : \mathcal{T}_\Sigma \to \mathcal{T}_{\Sigma'} \end{array}\right\} \Rightarrow h(L) \text{ is recognizable (over } \mathcal{T}_{\Sigma'})$$

Linear means that each variable appears at most once in $m(a)$.

Simple counter-example with $m(a) = a(x_1, x_1)$.

Tree homomorphisms

$$\text{Tree homomorphism} = \left\{ \begin{array}{l} \text{transformation } h : \mathcal{T}_\Sigma \to \mathcal{T}_{\Sigma'} \\ \text{based on a mapping } m : \Sigma \to \mathcal{T}_{\Sigma' \cup \mathcal{X}_{\text{ar}(m)}} \end{array} \right.$$

Theorem

$$\left. \begin{array}{l} L\text{: recognizable tree language in } \mathcal{T}_\Sigma \\ h\text{: linear tree homom. } h : \mathcal{T}_\Sigma \to \mathcal{T}_{\Sigma'} \end{array} \right\} \Rightarrow h(L) \text{ is recognizable (over } \mathcal{T}_{\Sigma'})$$

Linear means that each variable appears at most once in $m(a)$.

Simple counter-example with $m(a) = a(x_1, x_1)$.

Theorem

$$\left. \begin{array}{l} L\text{: recognizable tree language in } \mathcal{T}_{\Sigma'} \\ h\text{: any tree homom. } h : \mathcal{T}_\Sigma \to \mathcal{T}_{\Sigma'} \end{array} \right\} \Rightarrow h^{-1}(L) \text{ is recognizable (over } \mathcal{T}_\Sigma)$$

Minimization

of Tree Automata

Let $\equiv$ be an equivalence relation. It is:

- a congruence if $\forall a \in \mathcal{T}_{\Sigma_r}$,

  $$\text{if} \quad t_i \equiv t'_i \text{ for all } 1 \leq i \leq n \quad \text{then} \quad a(t_1, \ldots, t_n) \equiv a(t'_1, \ldots, t'_n)$$

- of finite index if there are only finitely many $\equiv$-classes

Minimization

of Tree Automata

Let $\equiv$ be an equivalence relation. It is:

- a congruence if $\forall a \in \mathcal{T}_{\Sigma_r}$,

$$\text{if} \quad t_i \equiv t'_i \text{ for all } 1 \leq i \leq n \quad \text{then} \quad a(t_1, \ldots, t_n) \equiv a(t'_1, \ldots, t'_n)$$

- of finite index if there are only finitely many $\equiv$-classes

★★★

Given a tree language L, we define the congruence $\equiv_L$:

$$t \equiv_L t' \qquad \text{if for all contexts } C \text{ over } \Sigma_r, C[t] \in L \Leftrightarrow C[t'] \in L$$

Minimization

of Tree Automata

Def: $t \equiv_L t'$ if $\forall C.\ C[t] \in L \Leftrightarrow C[t'] \in L$

Myhill-Nerode Theorem

L is a recognizable tree language iff $\equiv_L$ is of finite index.

- ($\Rightarrow$) Let A be a complete d$\uparrow$TA recognizing L. Let $\equiv_A$ defined by: $t \equiv_A t'$ iff $\Delta(t) = \Delta(t')$ (state of A). It is of finite index ($\leq |Q_A|$), and $L = \bigcup_{t \mid \Delta(t) \in F_A} \mathit{class}_{\equiv_A}(t)$.

Minimization

of Tree Automata

Def: $t \equiv_L t'$ if $\forall C.\ C[t] \in L \Leftrightarrow C[t'] \in L$

Myhill-Nerode Theorem

L is a recognizable tree language iff $\equiv_L$ is of finite index.

- ($\Rightarrow$) Let A be a complete d$\uparrow$TA recognizing L. Let $\equiv_A$ defined by: $t \equiv_A t'$ iff $\Delta(t) = \Delta(t')$ (state of A). It is of finite index ($\leq |Q_A|$), and $L = \bigcup_{t \mid \Delta(t) \in F_A} class_{\equiv_A}(t)$.
  As $\equiv_A$ is of finite index, it suffices to prove that $t \equiv_A t' \Rightarrow t \equiv_L t'$.
  Assume $t \equiv_A t'$. By an easy induction, $C[t] \equiv_A C[t']$ for all C. As L is a union of classes of $\equiv_A$, $C[t] \in L \Leftrightarrow C[t'] \in L$, and thus $t \equiv_L t'$.

Minimization

of Tree Automata

Def: $t \equiv_L t'$ if $\forall C.\ C[t] \in L \Leftrightarrow C[t'] \in L$

Myhill-Nerode Theorem

L is a recognizable tree language iff $\equiv_L$ is of finite index.

- ($\Rightarrow$) Let A be a complete d$\uparrow$TA recognizing L. Let $\equiv_A$ defined by: $t \equiv_A t'$ iff $\Delta(t) = \Delta(t')$ (state of A). It is of finite index ($\leq |Q_A|$), and $L = \bigcup_{t \mid \Delta(t) \in F_A} class_{\equiv_A}(t)$.
  As $\equiv_A$ is of finite index, it suffices to prove that $t \equiv_A t' \Rightarrow t \equiv_L t'$.
  Assume $t \equiv_A t'$. By an easy induction, $C[t] \equiv_A C[t']$ for all C. As L is a union of classes of $\equiv_A$, $C[t] \in L \Leftrightarrow C[t'] \in L$, and thus $t \equiv_L t'$.
- ($\Leftarrow$) $A_{min} = (Q_{min}, F_{min}, \Delta_{min}, \Sigma_r)$ with:
  - $Q_{min} = class_{\equiv_L}$
  - $F_{min} = \{class_{\equiv_L}(t) \mid t \in L\}$
  - Δ_{min} contains rules:
    $f(class_{\equiv_L}(t_1), \ldots, class_{\equiv_L}(t_n)) \to class_{\equiv_L}(f(t_1, \ldots, t_n))$

Minimization

of Tree Automata

Consequence of Myhill-Nerode

The minimum d↑TA recognizing L is unique, up to a renaming of states.

if A recognizes L, $\equiv_A$ is a refinement of $\equiv_L$, so $|Q_A| \geq |Q_{A_{min}}|$.

Minimization algorithm (sketch)

- input: complete and reduced d↑TA A
- start from $\equiv_A$ and build $\equiv_L$,
- by merging q and q' if $\forall a \in \Sigma_r$, $\forall i$, $\forall q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_n \in Q_A$, $\Delta(a(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n)) \equiv \Delta(a(q_1, \ldots, q_{i-1}, q', q_{i+1}, \ldots, q_n))$
- until fixed point

Complexity of some decision problems

Name	Input	Output	Complexity
Membership	$\uparrow$TA A, tree t	$t \in \mathcal{L}(A)$?	PTIME
Emptiness	$\uparrow$TA A	$\mathcal{L}(A) = \emptyset$?	PTIME
Intersection non-emptiness	set S of $\uparrow$TAs	$\bigcap_{A \in S} \mathcal{L}(A) = \emptyset$?	EXPTIME-compl.
	set S of d$\uparrow$TAs		EXPTIME-compl.
Universality	$\uparrow$TA A	$\mathcal{L}(A) = T_{\Sigma_r}$?	EXPTIME-compl.
	d$\uparrow$TA A		PTIME
Equivalence	$\uparrow$TAs A_1, A_2	$\mathcal{L}(A_1) = \mathcal{L}(A_2)$?	EXPTIME-compl.
	d$\uparrow$TAs A_1, A_2		PTIME

Tree Grammars

Let $\mathcal{X}$ be a set of variables.

Tree grammar

A tree grammar is a tuple $G = (S, N, F, R)$ where:

- N is a set of non-terminal symbols,
- $S \in N$ an axiom,
- F a set of terminal symbols ($F \cap N = \emptyset$),
- R a set of production rules $\alpha \to \beta$ with $\alpha, \beta \in \mathcal{T}_{N \cup F \cup \mathcal{X}}$ and α contains at least one non-terminal.

Each element of $N \cup F$ has a fixed arity, $\mathrm{ar}(S) = 0$, and $\mathrm{ar}(x) = 0$, for $x \in \mathcal{X}$.

$a(b(x, d(N_1(c)))) \to a(d(x), b(N_2, c))$

Regular Tree Grammars

Definition

Regular tree grammars

A tree grammar G is regular if $\mathrm{ar}(N_i) = 0$ for all $N_i \in N$, and production rules are of the form $N_i \to \beta$ with $N_i \in N$ and $\beta \in \mathcal{T}_{N \cup F}$.

$\mathcal{L}(G)$ is the set of trees obtained by applying a series of rules, starting from S.
$L \subseteq \mathcal{T}_{\Sigma_r}$ is said regular if $L = \mathcal{L}(G)$ for some regular tree grammar G.

Example

$$R = \left\{ \begin{array}{lllll} S \to a(S,X) & S \to b(S,X) & S \to a(B,X) & B \to b(X,X) & X \to b(X,X) \\ S \to a(X,S) & S \to b(X,S) & S \to a(X,B) & X \to a(X,X) & X \to c \end{array} \right.$$

$a(b(c,c),c) \in \mathcal{L}(G)$ because:
$S \to_G a(B,X) \to_G a(b(X,X),X) \to_G a(b(c,X),X) \to_G a(b(c,c),X) \to_G a(b(c,c),c)$

Regular tree grammars can be normalized, so that rules have the form $N_0 \to a(N_1, \ldots, N_n)$.

Regular Tree Grammars

Expressiveness

Proposition

regular tree languages $=$ recognizable tree languages

Normalized regular tree grammars are $\downarrow$TAs...

Regular Tree Grammars

Expressiveness

Proposition

regular tree languages = recognizable tree languages

Normalized regular tree grammars are $\downarrow$TAs...

Regular Expressions

There also exists a notion of regular expressions for trees, with the same expressiveness as regular tree grammar (Kleene's theorem).

skipped in this talk...

1. Ranked Trees
   - Trees on Ranked Alphabet
   - Tree Automata
   - Tree Grammars
   - Logic

2. Unranked Trees
   - Unranked Trees
   - Automata
   - Logic

Monadic Second-Order (MSO) Logic

Syntax

- Σ_r a ranked signature
- $\mathcal{X}$ contains first-order $(x, y \ldots)$ and second-order $(X, Y \ldots)$ variables.
- $\Omega = \{\mathrm{lab}_a \mid a \in \Sigma_r\} \cup \{ch_i \mid \exists a \in \Sigma_r.\ \mathrm{ar}(a) \geq i\}$
- predicates lab_a are unary, while ch_i are binary

$MSO[\Omega]$: Syntax

$$\phi ::= \mathrm{lab}_a(x) \mid ch_i(x, y) \mid \phi \wedge \phi \mid \neg\phi \mid \exists x.\ \phi \mid \exists X.\ \phi \mid x \in X$$

where $a \in \Sigma$, $\exists a \in \Sigma_r.\ \mathrm{ar}(a) \geq i$, and $x, x_1, \ldots, x_k, X \in \mathcal{X}$.

Monadic Second-Order (MSO) Logic

Syntax

- Σ_r a ranked signature
- $\mathcal{X}$ contains first-order $(x, y \ldots)$ and second-order $(X, Y \ldots)$ variables.
- $\Omega = \{\mathrm{lab}_a \mid a \in \Sigma_r\} \cup \{ch_i \mid \exists a \in \Sigma_r.\ \mathrm{ar}(a) \geq i\}$
- predicates lab_a are unary, while ch_i are binary

$MSO[\Omega]$: Syntax

$$\phi ::= \mathrm{lab}_a(x) \mid ch_i(x, y) \mid \phi \wedge \phi \mid \neg\phi \mid \exists x.\ \phi \mid \exists X.\ \phi \mid x \in X$$

where $a \in \Sigma$, $\exists a \in \Sigma_r.\ \mathrm{ar}(a) \geq i$, and $x, x_1, \ldots, x_k, X \in \mathcal{X}$.

$MSO[\Omega]$ is also known as WSkS, the *Weak Second-order logic with k Successors*. "Weak" means "interpreted over finite structures" (here, terms).

Monadic Second-Order (MSO) Logic

Example

For instance: all trees having an even number of a-labeled nodes.

hint: define X as the set of nodes having an even number of a-descendants

Monadic Second-Order (MSO) Logic

Example

For instance: all trees having an even number of a-labeled nodes.

hint: define X as the set of nodes having an even number of a-descendants

$$even_a = \exists X\ \forall x\ \mathsf{lab}_a(x) \Rightarrow \begin{cases} \exists y_1\ ch_1(x, y_1) \wedge \\ \exists y_2\ ch_2(x, y_2) \wedge \\ x \in X \Leftrightarrow (y_1 \in X \oplus y_2 \in X) \end{cases}$$

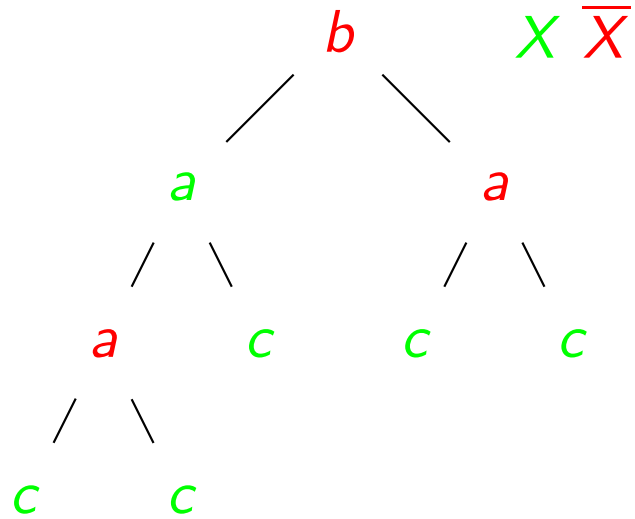
$$\bigwedge_{\alpha \neq a} \mathsf{lab}_\alpha(x) \Rightarrow \begin{cases} \bigwedge_{1 \leq i \leq \mathsf{ar}(\alpha)} \exists y_i\ ch_i(x, y_i) \wedge \\ x \notin X \Leftrightarrow (y_1 \notin X \oplus \ \ldots\ \oplus y_{\mathsf{ar}(\alpha)} \notin X) \end{cases}$$

$$\wedge \exists x\ root(x)\ \wedge\ x \in X$$

where:

$\phi \oplus \phi' = (\phi\ \wedge \neg\phi')\ \vee (\neg\phi\ \wedge\ \phi')$

$root(x) = \neg\exists y.\ ch_1(y, x)$



Monadic Second-Order (MSO) Logic

Semantics

convention: $\phi(\overline{x}, \overline{X})$ means that ϕ has free FO variables $\overline{x}$ and free SO variables $\overline{X}$.

A formula $\phi(\overline{x}, \overline{X}) \in MSO[\Omega]$ is interpreted over a tree t under an assignment $\mu : \overline{x} \cup \overline{X} \to \text{nodes}(t)$.

Satisfiability $t, \mu \models \phi$ is defined inductively by:

$$
\begin{array}{ll}
t, \mu \models \text{lab}_a(x) & \text{iff } \text{lab}_a^t(\mu(x)) \\
t, \mu \models ch_i(x, y) & \text{iff } ch_i(\mu(x), \mu(y)) \\
t, \mu \models \phi \ \wedge \ \phi' & \text{iff } t, \mu \models \phi \text{ and } t, \mu \models \phi' \\
t, \mu \models \neg\phi & \text{iff } t, \mu \not\models \phi \\
t, \mu \models \exists x \phi & \text{iff there exists } \pi \in \text{nodes}(t) \text{ s.t. } t, \mu[x \leftarrow \pi] \models \phi \\
t, \mu \models \exists X \phi & \text{iff there exists } S \subseteq \text{nodes}(t) \text{ s.t. } t, \mu[X \leftarrow S] \models \phi \\
t, \mu \models x \in X & \text{iff } \mu(x) \in \mu(X)
\end{array}
$$

MSO and Tree Languages

Expressiveness of MSO vs ↑TA?

For closed formulas ϕ, define $\mathcal{L}_\phi = \{t \mid t \models \phi\} \subseteq \mathcal{T}_{\Sigma_r}$.

What about $\phi(\overline{x}, \overline{X})$?

$$\phi(\overline{x}, \overline{X}) \xrightarrow{\text{semantics}} \{(t, \mu) \mid t, \mu \models \phi\} \xrightarrow{\text{tree language}} \mathcal{L}_\phi = \{t * \mu \mid t, \mu \models \phi\}$$

MSO and Tree Languages

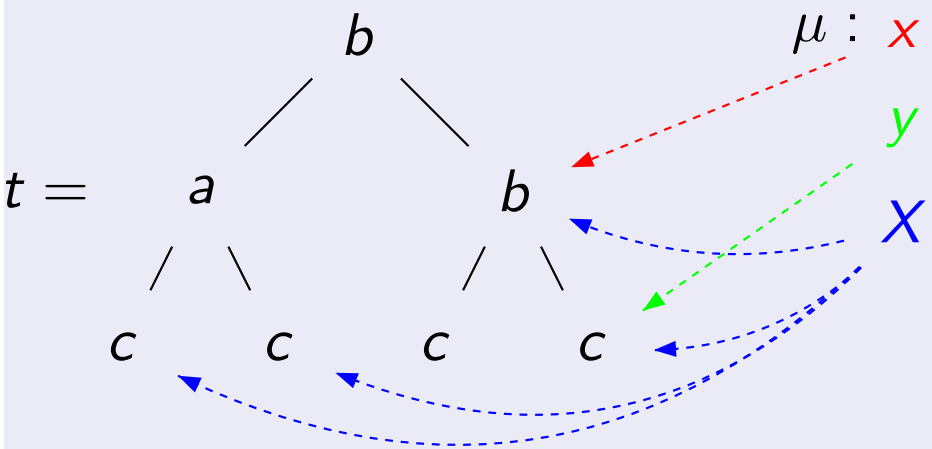
Expressiveness of MSO vs ↑TA?

For closed formulas ϕ, define $\mathcal{L}_\phi = \{t \mid t \models \phi\} \subseteq \mathcal{T}_{\Sigma_r}$.

What about $\phi(\overline{x}, \overline{X})$?

$$\phi(\overline{x}, \overline{X}) \xrightarrow{\text{semantics}} \{(t, \mu) \mid t, \mu \models \phi\} \xrightarrow{\text{tree language}} \mathcal{L}_\phi = \{t * \mu \mid t, \mu \models \phi\}$$

$t * \mu \in \mathcal{T}_{\Sigma_r \times \mathbb{B}^n}$



where $n = |\overline{x}| + |\overline{X}|$ and $\mathbb{B} = \{0, 1\}$.

MSO and Tree Languages

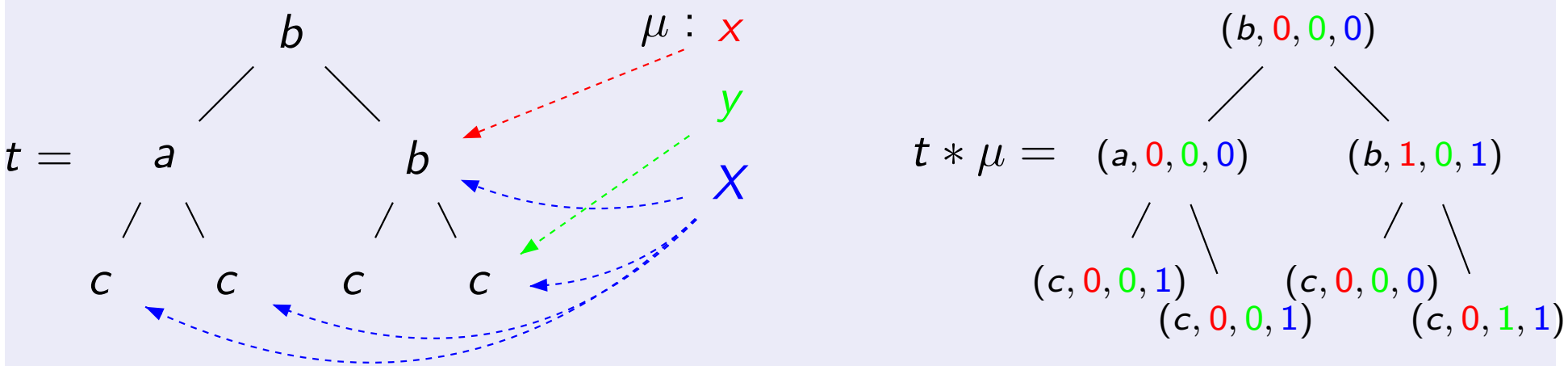
Expressiveness of MSO vs ↑TA?

For closed formulas ϕ, define $\mathcal{L}_\phi = \{t \mid t \models \phi\} \subseteq \mathcal{T}_{\Sigma_r}$.

What about $\phi(\overline{x}, \overline{X})$?

$$\phi(\overline{x}, \overline{X}) \xrightarrow{\text{semantics}} \{(t, \mu) \mid t, \mu \models \phi\} \xrightarrow{\text{tree language}} \mathcal{L}_\phi = \{t * \mu \mid t, \mu \models \phi\}$$

$t * \mu \in \mathcal{T}_{\Sigma_r \times \mathbb{B}^n}$



where $n = |\overline{x}| + |\overline{X}|$ and $\mathbb{B} = \{0, 1\}$.

Recognizable languages = MSO-definable languages

Theorem [TW68, Don70]

$L \subseteq \mathcal{T}_{\Sigma_r}$ is recognizable iff there exists $\phi \in MSO[\Omega]$ such that $\mathcal{L}_\phi = L$.

consequence: $MSO[\Omega]$ is decidable (convert to $\uparrow$TA, test emptiness)

($\Rightarrow$) *idea: encode a run of A in MSO.*
$A = (Q, F, \Delta, \Sigma_r)$ a complete d$\uparrow$TA recognizing L. Let $\{q_1, \ldots, q_n\} = Q$.

$$\phi_A = \exists X_{q_1} \ldots \exists X_{q_n}\ partition(X_{q_1}, \ldots, X_{q_n}) \wedge$$
$$\bigwedge_{a \to q \in \Delta} leaf(x) \wedge \mathrm{lab}_a(x) \Rightarrow x \in X_q$$
$$\bigwedge_{a(q_{i_1}, \ldots, q_{i_k}) \to q_i \in \Delta} \forall x\ \forall y_1 \ldots \forall y_k \left\{ \begin{array}{l} \mathrm{lab}_a(x) \wedge \\ ch_1(x, y_1) \wedge \ldots \wedge ch(x, y_k) \wedge \\ y_1 \in X_{q_{i_1}} \wedge \ldots \wedge y_k \in X_{q_{i_k}} \end{array} \right\} \Rightarrow x \in X_{q_i}$$
$$\wedge\ \exists x. root(x) \wedge \bigvee_{q \in F} x \in X_q$$

with: $\left\{ \begin{array}{l} leaf(x) = \nexists y.\ ch_1(x, y) \\ partition(X_{q_1}, \ldots, X_{q_n}) = \ldots \end{array} \right.$

($\Leftarrow$) *idea: equivalence betw. logical connectives and automata operators.*

- $\mathcal{L}_{\mathsf{lab}_a} = \left\{ \ldots, \begin{array}{ccc} & (a,1) & \\ \swarrow & & \searrow \\ (a,0) & & (b,0) \end{array}, \begin{array}{ccc} & (a,0) & \\ \swarrow & & \searrow \\ (a,1) & & (b,0) \end{array}, \ldots \right\}$ recognizable

($\Leftarrow$) *idea: equivalence betw. logical connectives and automata operators.*

- $\mathcal{L}_{\mathrm{lab}_a} = \left\{ \ldots, \begin{array}{ccc} & (a,1) & \\ \swarrow & & \searrow \\ (a,0) & & (b,0) \end{array}, \begin{array}{ccc} & (a,0) & \\ \swarrow & & \searrow \\ (a,1) & & (b,0) \end{array}, \ldots \right\}$ recognizable
- $\mathcal{L}_{ch_1} = \left\{ \ldots, \begin{array}{ccc} & (a,1,0) & \\ \swarrow & & \searrow \\ (a,0,1) & & (b,0,0) \end{array}, \ldots \right\}$ recognizable (ch_i also)

($\Leftarrow$) *idea: equivalence betw. logical connectives and automata operators.*

- $\mathcal{L}_{\mathsf{lab}_a} = \left\{ \ldots, \begin{array}{ccc} & (a,1) & \\ \swarrow & & \searrow \\ (a,0) & & (b,0) \end{array}, \begin{array}{ccc} & (a,0) & \\ \swarrow & & \searrow \\ (a,1) & & (b,0) \end{array}, \ldots \right\}$ recognizable
- $\mathcal{L}_{ch_1} = \left\{ \ldots, \begin{array}{ccc} & (a,1,0) & \\ \swarrow & & \searrow \\ (a,0,1) & & (b,0,0) \end{array}, \ldots \right\}$ recognizable (ch_i also)
- $\mathcal{L}_{\phi \wedge \phi'} = \mathcal{L}_\phi \cap \mathcal{L}_{\phi'}$ recognizable (product construction)

($\Leftarrow$) *idea: equivalence betw. logical connectives and automata operators.*

- $\mathcal{L}_{\mathrm{lab}_a} = \left\{ \ldots, \begin{array}{ccc} & (a,1) & \\ \swarrow & & \searrow \\ (a,0) & & (b,0) \end{array}, \begin{array}{ccc} & (a,0) & \\ \swarrow & & \searrow \\ (a,1) & & (b,0) \end{array}, \ldots \right\}$ recognizable
- $\mathcal{L}_{ch_1} = \left\{ \ldots, \begin{array}{ccc} & (a,1,0) & \\ \swarrow & & \searrow \\ (a,0,1) & & (b,0,0) \end{array}, \ldots \right\}$ recognizable (ch_i also)
- $\mathcal{L}_{\phi \wedge \phi'} = \mathcal{L}_\phi \cap \mathcal{L}_{\phi'}$ recognizable (product construction)
- $\mathcal{L}_{\neg\phi} = \mathcal{L}_{valid} \setminus \mathcal{L}_\phi$ where $\mathcal{L}_{valid} = \{t * \mu \mid t \in \mathcal{T}_{\Sigma_r},\ \mu \text{ an assignment of free vars of } \phi\}$
  $\mathcal{L}_{valid}$ is recognizable so $\mathcal{L}_{\neg\phi}$ is recognizable

($\Leftarrow$) *idea: equivalence betw. logical connectives and automata operators.*

- $\mathcal{L}_{\mathsf{lab}_a} = \left\{ \ldots, \begin{array}{ccc} & (a,1) & \\ / & & \backslash \\ (a,0) & & (b,0) \end{array}, \begin{array}{ccc} & (a,0) & \\ / & & \backslash \\ (a,1) & & (b,0) \end{array}, \ldots \right\}$ recognizable
- $\mathcal{L}_{ch_1} = \left\{ \ldots, \begin{array}{ccc} & (a,1,0) & \\ / & & \backslash \\ (a,0,1) & & (b,0,0) \end{array}, \ldots \right\}$ recognizable (ch_i also)
- $\mathcal{L}_{\phi \wedge \phi'} = \mathcal{L}_\phi \cap \mathcal{L}_{\phi'}$ recognizable (product construction)
- $\mathcal{L}_{\neg\phi} = \mathcal{L}_{valid} \setminus \mathcal{L}_\phi$ where $\mathcal{L}_{valid} = \{t * \mu \mid t \in \mathcal{T}_{\Sigma_r},\ \mu \text{ an assignment of free vars of } \phi\}$
  $\mathcal{L}_{valid}$ is recognizable so $\mathcal{L}_{\neg\phi}$ is recognizable
- $\mathcal{L}_{\exists x \phi}$ is obtained from $\mathcal{L}_\phi$ by removing the x-component:

$$\begin{array}{ccc} & (a,1,0) & \\ / & & \backslash \\ (a,0,1) & & (b,0,0) \end{array} \in \mathcal{L}_{ch_1(x,y)} \quad \rightarrow \quad \begin{array}{ccc} & (a,0) & \\ / & & \backslash \\ (a,1) & & (b,0) \end{array} \in \mathcal{L}_{\exists x\ ch_1(x,y)}$$

  recognizable (remove the component in rules)

($\Leftarrow$) *idea: equivalence betw. logical connectives and automata operators.*

- $\mathcal{L}_{\mathrm{lab}_a} = \left\{ \ldots, \begin{array}{ccc} & (a,1) & \\ / & & \backslash \\ (a,0) & & (b,0) \end{array}, \begin{array}{ccc} & (a,0) & \\ / & & \backslash \\ (a,1) & & (b,0) \end{array}, \ldots \right\}$ recognizable
- $\mathcal{L}_{ch_1} = \left\{ \ldots, \begin{array}{ccc} & (a,1,0) & \\ / & & \backslash \\ (a,0,1) & & (b,0,0) \end{array}, \ldots \right\}$ recognizable (ch_i also)
- $\mathcal{L}_{\phi \wedge \phi'} = \mathcal{L}_\phi \cap \mathcal{L}_{\phi'}$ recognizable (product construction)
- $\mathcal{L}_{\neg\phi} = \mathcal{L}_{valid} \setminus \mathcal{L}_\phi$ where $\mathcal{L}_{valid} = \{t * \mu \mid t \in \mathcal{T}_{\Sigma_r},\ \mu \text{ an assignment of free vars of } \phi\}$
  $\mathcal{L}_{valid}$ is recognizable so $\mathcal{L}_{\neg\phi}$ is recognizable
- $\mathcal{L}_{\exists x \phi}$ is obtained from $\mathcal{L}_\phi$ by removing the x-component:

  $\begin{array}{ccc} & (a,1,0) & \\ / & & \backslash \\ (a,0,1) & & (b,0,0) \end{array} \in \mathcal{L}_{ch_1(x,y)} \quad \rightarrow \quad \begin{array}{ccc} & (a,0) & \\ / & & \backslash \\ (a,1) & & (b,0) \end{array} \in \mathcal{L}_{\exists x\ ch_1(x,y)}$

  recognizable (remove the component in rules)
- $\mathcal{L}_{\exists X \phi}$: same idea

($\Leftarrow$) *idea: equivalence betw. logical connectives and automata operators.*

- $\mathcal{L}_{\mathsf{lab}_a} = \left\{ \ldots, \begin{array}{ccc} & (a,1) & \\ / & & \backslash \\ (a,0) & & (b,0) \end{array}, \begin{array}{ccc} & (a,0) & \\ / & & \backslash \\ (a,1) & & (b,0) \end{array}, \ldots \right\}$ recognizable
- $\mathcal{L}_{ch_1} = \left\{ \ldots, \begin{array}{ccc} & (a,1,0) & \\ / & & \backslash \\ (a,0,1) & & (b,0,0) \end{array}, \ldots \right\}$ recognizable (ch_i also)
- $\mathcal{L}_{\phi \wedge \phi'} = \mathcal{L}_\phi \cap \mathcal{L}_{\phi'}$ recognizable (product construction)
- $\mathcal{L}_{\neg\phi} = \mathcal{L}_{valid} \setminus \mathcal{L}_\phi$ where $\mathcal{L}_{valid} = \{t * \mu \mid t \in \mathcal{T}_{\Sigma_r},\ \mu \text{ an assignment of free vars of } \phi\}$
  $\mathcal{L}_{valid}$ is recognizable so $\mathcal{L}_{\neg\phi}$ is recognizable
- $\mathcal{L}_{\exists x \phi}$ is obtained from $\mathcal{L}_\phi$ by removing the x-component:

$$\begin{array}{ccc} & (a,1,0) & \\ / & & \backslash \\ (a,0,1) & & (b,0,0) \end{array} \in \mathcal{L}_{ch_1(x,y)} \quad \rightarrow \quad \begin{array}{ccc} & (a,0) & \\ / & & \backslash \\ (a,1) & & (b,0) \end{array} \in \mathcal{L}_{\exists x\ ch_1(x,y)}$$

  recognizable (remove the component in rules)
- $\mathcal{L}_{\exists X \phi}$: same idea
- $\mathcal{L}_{x \in X}$: easily recognizable (a 1 on x-component implies a 1 on X-component)

$$\begin{array}{ccc} & (a,1,1) & \\ / & & \backslash \\ (a,0,1) & & (b,0,0) \end{array} \in \mathcal{L}_{x \in X} \quad \text{but} \quad \begin{array}{ccc} & (a,0,1) & \\ / & & \backslash \\ (a,1,0) & & (b,0,1) \end{array} \notin \mathcal{L}_{x \in X}$$

Bonus: Recognizable tree relations

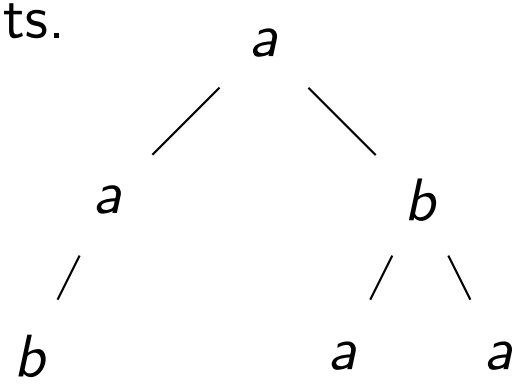
see [CDG+07] Chapter 3, and also [BLN07, Gau09]

1. Ranked Trees
   - Trees on Ranked Alphabet
   - Tree Automata
   - Tree Grammars
   - Logic

2. Unranked Trees
   - Unranked Trees
   - Automata
   - Logic

Unranked Trees

An unranked tree over Σ is a tree, labeled by elements of Σ, without arity constraints.



We write $\mathcal{T}_\Sigma$ for the set of unranked trees over Σ.

Which notion of automata can we use?

For ranked trees, rules were $a(q_1, \ldots, q_n) \to q$ but here n is not bounded.
Several approaches:

- horizontal languages: use a string language on the states of children
  - hedge automata
  - DTDs
- binary encodings: encode unranked trees into binary trees
  - stepwise tree automata...
- linearization: serialize trees and use a pushdown system
  - nested word automata
  - visibly pushdown automata

Horizontal Languages: Hedge Automata

idea: use a regular language on the states of children

"hedge" = finite series of trees

Hedge automata [BKWM01]

A hedge automaton over Σ is a tuple $A = (Q, F, \Delta, \Sigma)$ where:

- Q is a finite set of states
- $F \subseteq Q$ is the set of final states
- Δ is a set of rules $a(L) \to q$ where $a \in \Sigma$, $q \in Q$ and L is a regular string language over Q.

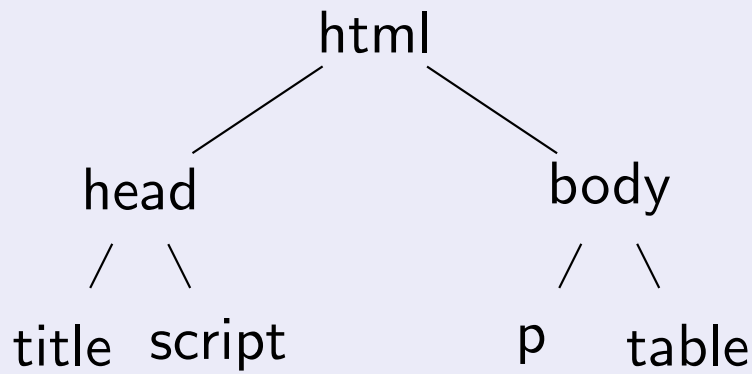
A run of A on t is a function $\rho : \text{nodes}(t) \to Q$ such that for all nodes π of t with children $\pi_1, \ldots, \pi_n$ and label a, there is a rule $a(L) \to \rho(\pi) \in \Delta$ with $\rho(\pi_1) \ldots \rho(\pi_n) \in L$.

Horizontal Languages: DTDs

- Document Type Definitions [BPSM+08]
- W3C standard for specifying valid XML documents
- hedge automata, where horizontal languages are specified by regexp.

Example: HTML DTD

$$
\begin{array}{lcl}
\mathit{html} & \rightarrow & \mathit{head}.\mathit{body} \\
\mathit{head} & \rightarrow & \mathit{title}.\mathit{meta}?.\mathit{style}?.\mathit{script}?... \\
\mathit{body} & \rightarrow & (p|\mathit{div}|\mathit{table}|h1|...)^* \\
... & &
\end{array}
$$



Binary Encodings

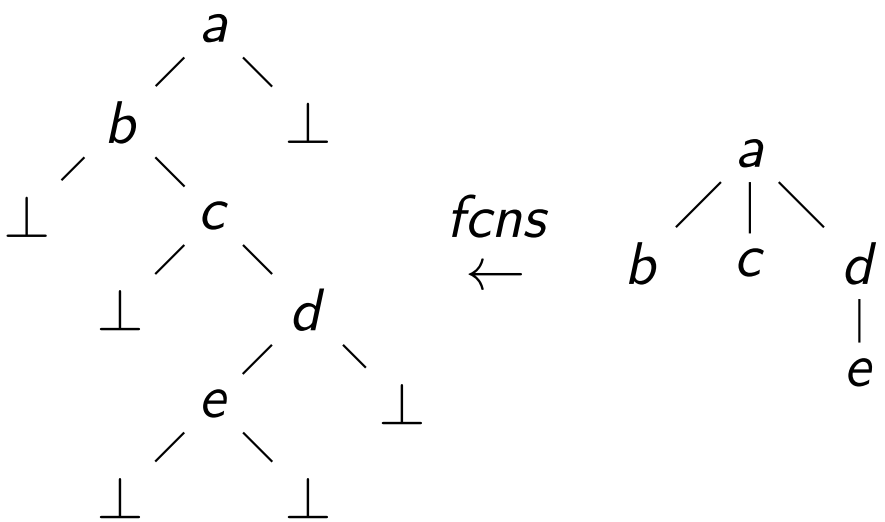
- define a bijection between unranked trees and binary trees
- 2 common encodings: first-child next-sibling and Curryfication

Binary Encodings

- define a bijection between unranked trees and binary trees
- 2 common encodings: first-child next-sibling and Curryfication

first-child next-sibling [Rab69, Koc03]

$fcns : \mathcal{T}_\Sigma \to \mathcal{T}_{\Sigma_f}$ where $\Sigma_f = (\Sigma \uplus \{\bot\}, \mathrm{ar})$ with $\mathrm{ar}(a)=2$ for $a \in \Sigma$ and $\mathrm{ar}(\bot)=0$



Binary Encodings

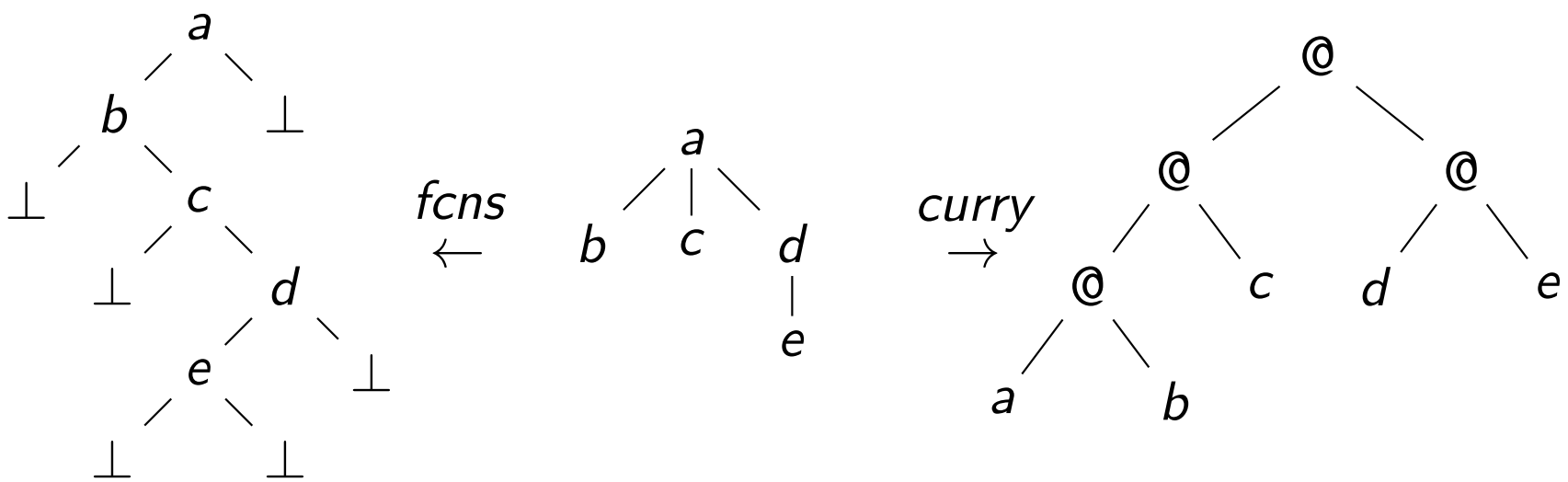
- define a bijection between unranked trees and binary trees
- 2 common encodings: first-child next-sibling and Curryfication

first-child next-sibling [Rab69, Koc03]

$fcns : \mathcal{T}_\Sigma \to \mathcal{T}_{\Sigma_f}$ where $\Sigma_f = (\Sigma \uplus \{\bot\}, \mathrm{ar})$ with $\mathrm{ar}(a)=2$ for $a \in \Sigma$ and $\mathrm{ar}(\bot)=0$

Curryfication

$curry : \mathcal{T}_\Sigma \to \mathcal{T}_{\Sigma_c}$ where $\Sigma_c = (\Sigma \uplus \{@\}, \mathrm{ar})$ with $\mathrm{ar}(a)=0$ for $a \in \Sigma$ and $\mathrm{ar}(@)=2$
$t_1 @ t_2 =$ "add t_2 as last child of the root of t_1"



Binary Encodings

Automata

We can use tree automata over ranked languages on encoded trees:

	fcns	*curry*
$\uparrow$TA	C_1	C_2
$\downarrow$TA	C_3	C_4

These 4 classes are equally expressive.

C_2 corresponds to stepwise tree automata [CNT04].

Binary Encodings

Automata

We can use tree automata over ranked languages on encoded trees:

	fcns	*curry*
$\uparrow$TA	C_1	C_2
$\downarrow$TA	C_3	C_4

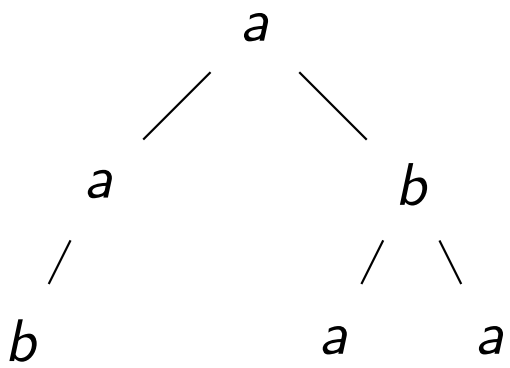
These 4 classes are equally expressive.

C_2 corresponds to stepwise tree automata [CNT04].

d$\downarrow$TAs $\circ \{fcns, curry\}$ define two other classes, that have different expressiveness. d$\downarrow$TAs $\circ fcns$ is the determinism of DTDs.

Linearization

Let $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$.
$lin(t)$, the linearization of t, is the word over $\Sigma \cup \overline{\Sigma}$ produced by the pre-order traversal of t:



$$lin(t) = a.a.b.\overline{b}.\overline{a}.b.a.\overline{a}.a.\overline{a}.\overline{b}.\overline{a}$$

This corresponds to the XML serialization: <a><a>...

Visibly Pushdown Automata [AM04]

- a pushdown automaton over $\Sigma \uplus \overline{\Sigma}$
- visible means that 1 action (push/pop) is performed by each letter:
  - rules using $a \in \Sigma$ only push
  - rules using $\overline{a} \in \overline{\Sigma}$ only pop

Visibly Pushdown Automaton (VPA)

A VPA is a tuple $(Q, I, F, \Gamma, \Delta, \Sigma \uplus \overline{\Sigma})$ with $I, F \subseteq Q$ and Δ is a set of rules with the form:

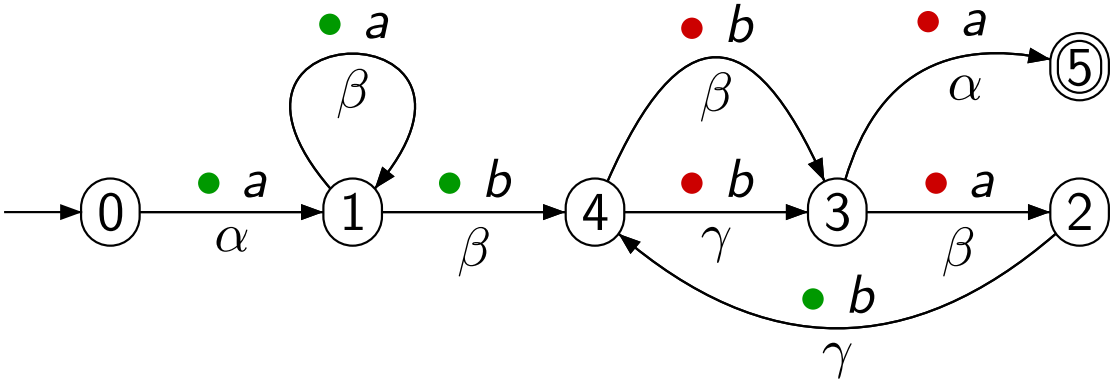
- $q_1, a \to \gamma, q_2$
- $q_1, \overline{a}, \gamma \to q_2$

with $q_1, q_2 \in Q$, $a \in \Sigma$, $\overline{a} \in \overline{\Sigma}$, $\gamma \in \Gamma$.

The semantics is defined as for usual pushdown automata (ends on final states, not on empty stack).

VPAs as Tree Automata

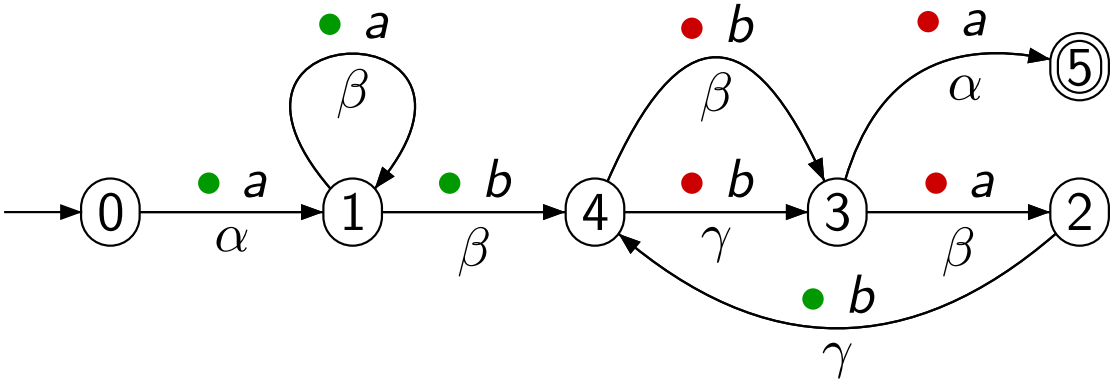
- a run of a VPA on a tree consists in:
  - when opening node π: update the current state, and assign $\gamma \in \Gamma$ to π
  - when closing node π: update the current state according to γ



A: VPA on $\mathcal{T}_\Sigma$ with $\Sigma = \{a, b\}$
and $Q = \{0, 1, 2, 3, 4, 5\}$
and $\Gamma = \{\alpha, \beta, \gamma\}$

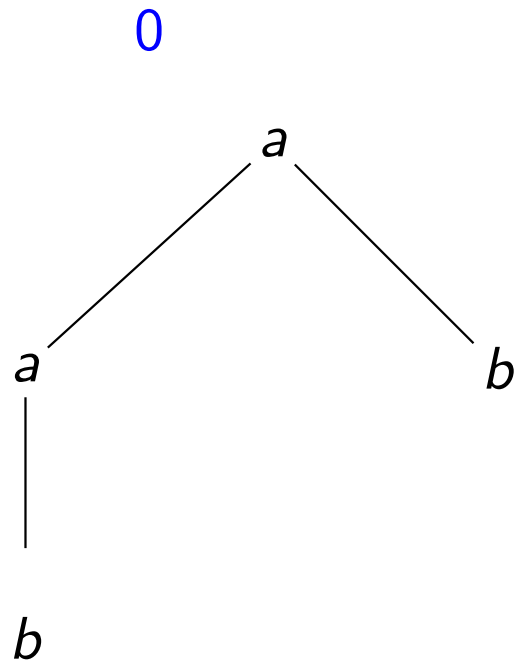
VPAs as Tree Automata

- a run of a VPA on a tree consists in:
  - when opening node π: update the current state, and assign $\gamma \in \Gamma$ to π
  - when closing node π: update the current state according to γ



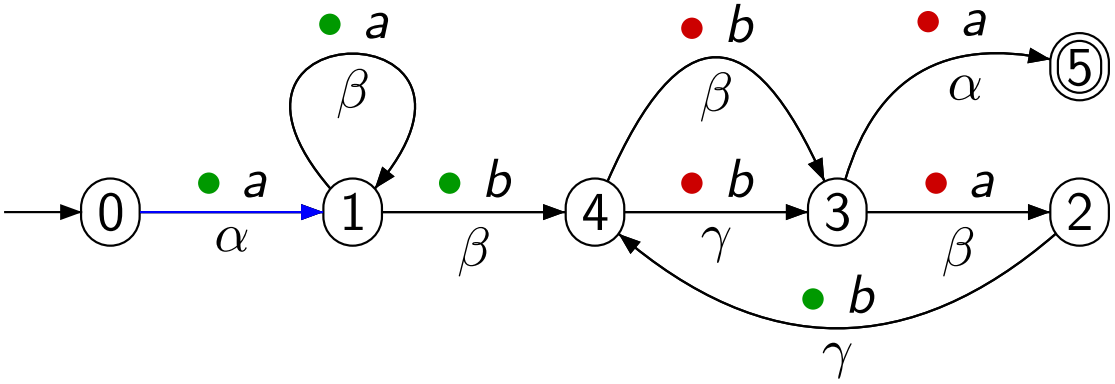
A: VPA on $\mathcal{T}_\Sigma$ with $\Sigma = \{a, b\}$
and $Q = \{0, 1, 2, 3, 4, 5\}$
and $\Gamma = \{\alpha, \beta, \gamma\}$

$(0, \emptyset)$



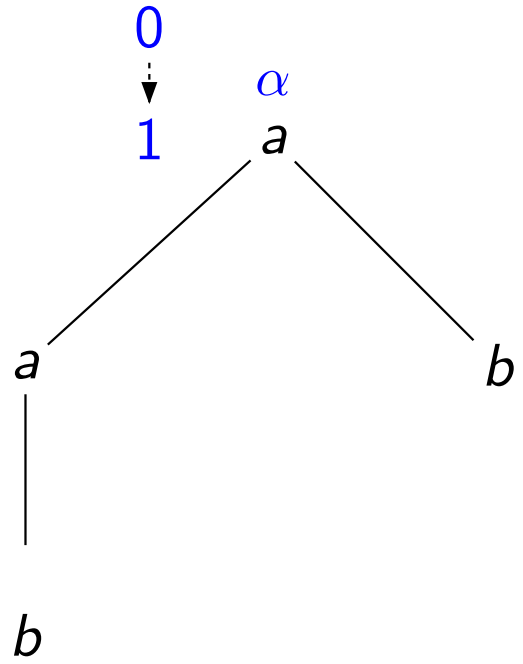
VPAs as Tree Automata

- a run of a VPA on a tree consists in:
  - when opening node π: update the current state, and assign $\gamma \in \Gamma$ to π
  - when closing node π: update the current state according to γ



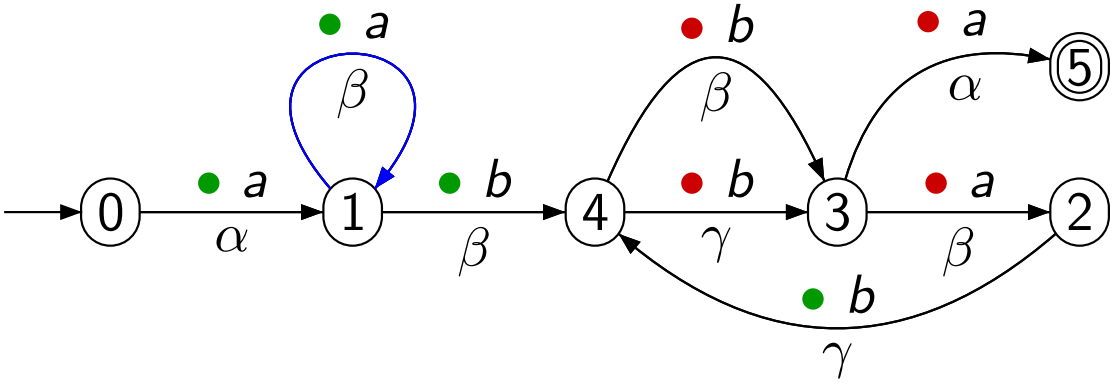
A: VPA on $\mathcal{T}_\Sigma$ with $\Sigma = \{a, b\}$
and $Q = \{0, 1, 2, 3, 4, 5\}$
and $\Gamma = \{\alpha, \beta, \gamma\}$

$(0, \emptyset) \xrightarrow{\bullet a} (1, \alpha)$



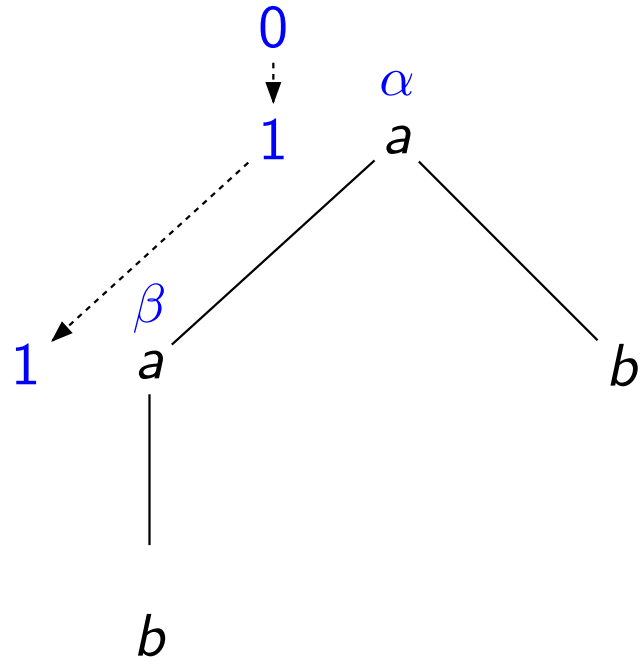
VPAs as Tree Automata

- a run of a VPA on a tree consists in:
  - when opening node π: update the current state, and assign $\gamma \in \Gamma$ to π
  - when closing node π: update the current state according to γ



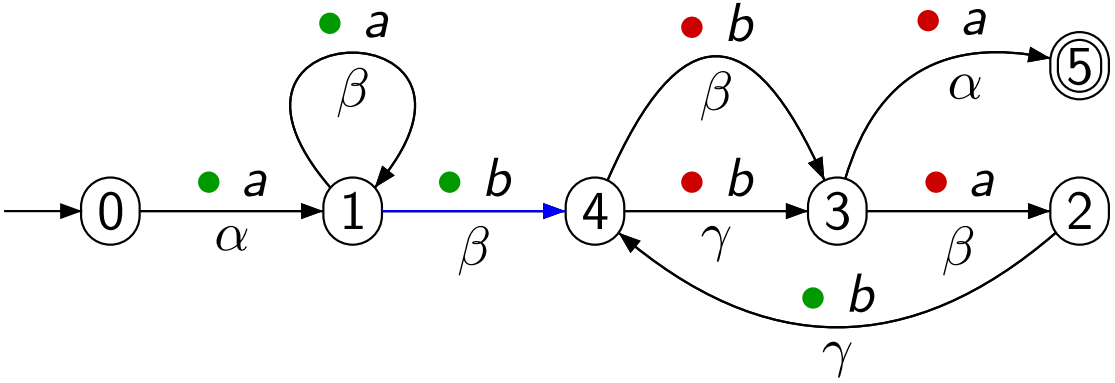
A: VPA on $\mathcal{T}_\Sigma$ with $\Sigma = \{a, b\}$
and $Q = \{0, 1, 2, 3, 4, 5\}$
and $\Gamma = \{\alpha, \beta, \gamma\}$

$(0, \emptyset) \xrightarrow{\bullet a} (1, \alpha) \xrightarrow{\bullet a} (1, \alpha.\beta)$



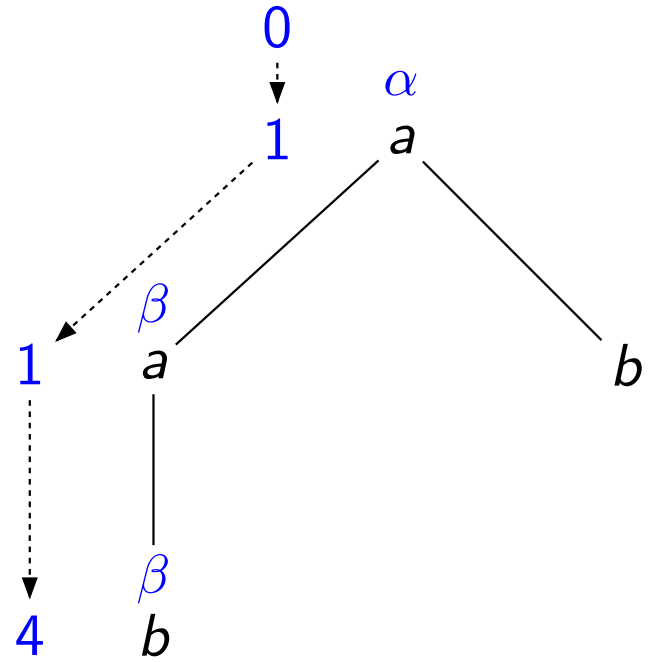
VPAs as Tree Automata

- a run of a VPA on a tree consists in:
  - when opening node π: update the current state, and assign $\gamma \in \Gamma$ to π
  - when closing node π: update the current state according to γ



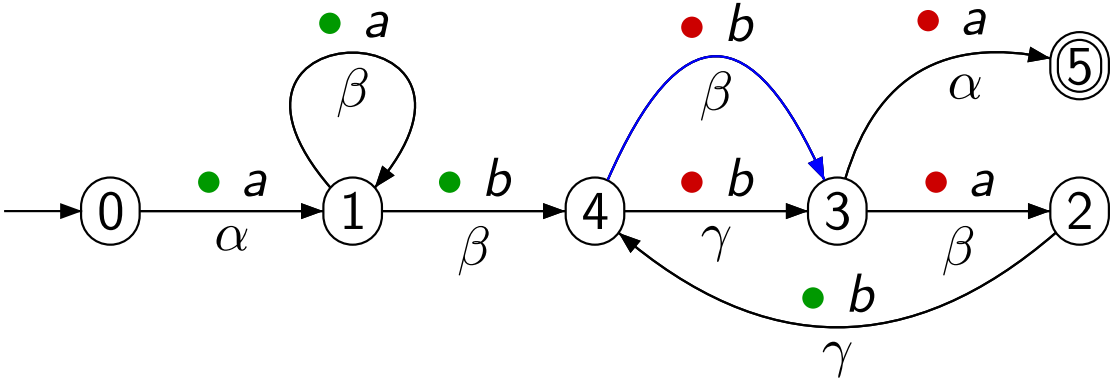
A: VPA on $\mathcal{T}_\Sigma$ with $\Sigma = \{a, b\}$
and $Q = \{0, 1, 2, 3, 4, 5\}$
and $\Gamma = \{\alpha, \beta, \gamma\}$

$$(0, \emptyset) \xrightarrow{\bullet a} (1, \alpha) \xrightarrow{\bullet a} (1, \alpha.\beta) \xrightarrow{\bullet b} (4, \alpha.\beta.\beta)$$



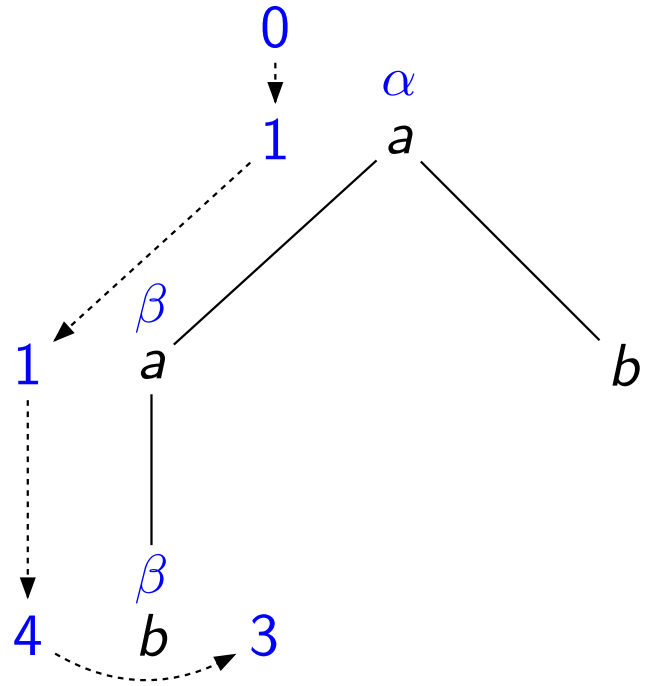
VPAs as Tree Automata

- a run of a VPA on a tree consists in:
  - when opening node π: update the current state, and assign $\gamma \in \Gamma$ to π
  - when closing node π: update the current state according to γ



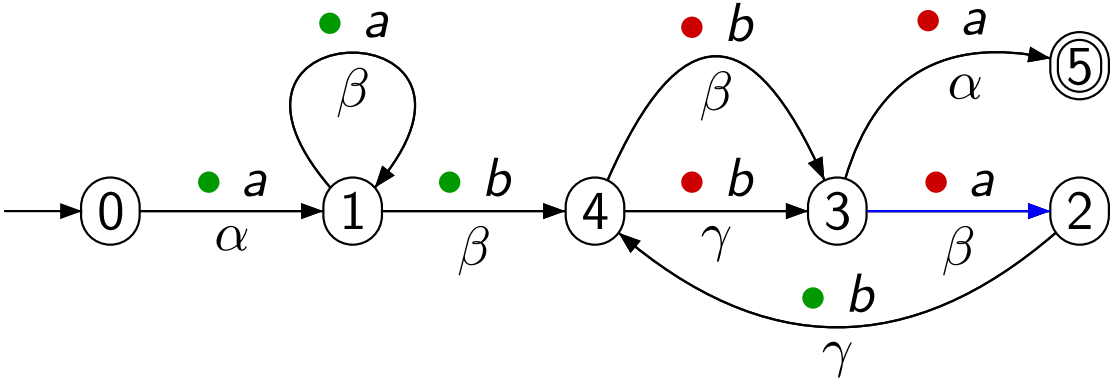
A: VPA on $\mathcal{T}_\Sigma$ with $\Sigma = \{a, b\}$
and $Q = \{0, 1, 2, 3, 4, 5\}$
and $\Gamma = \{\alpha, \beta, \gamma\}$

$(0, \emptyset) \xrightarrow{\bullet a} (1, \alpha) \xrightarrow{\bullet a} (1, \alpha.\beta) \xrightarrow{\bullet b} (4, \alpha.\beta.\beta)$
$\xrightarrow{\bullet b} (3, \alpha.\beta)$



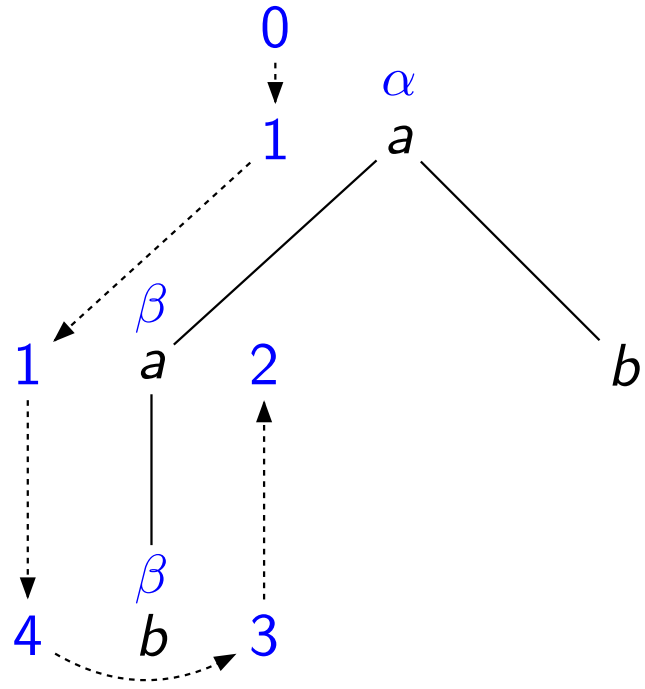
VPAs as Tree Automata

- a run of a VPA on a tree consists in:
  - when opening node π: update the current state, and assign $\gamma \in \Gamma$ to π
  - when closing node π: update the current state according to γ



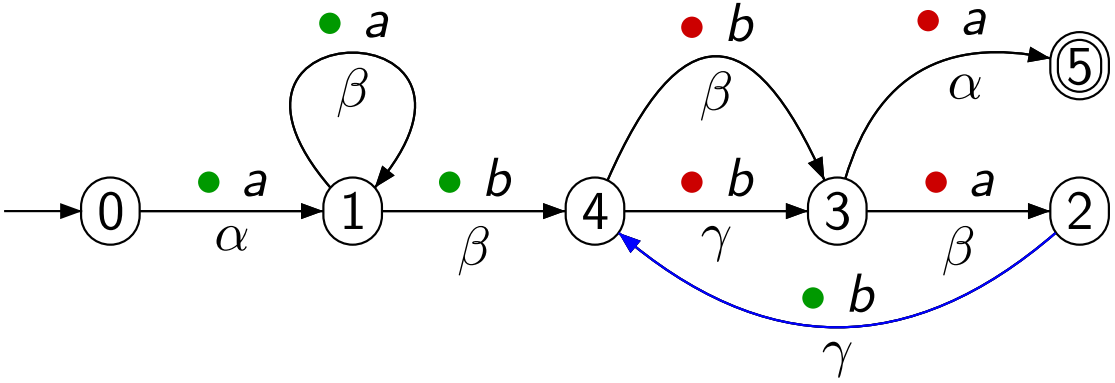
A: VPA on $\mathcal{T}_\Sigma$ with $\Sigma = \{a, b\}$
and $Q = \{0, 1, 2, 3, 4, 5\}$
and $\Gamma = \{\alpha, \beta, \gamma\}$

$(0, \emptyset) \xrightarrow{\bullet a} (1, \alpha) \xrightarrow{\bullet a} (1, \alpha.\beta) \xrightarrow{\bullet b} (4, \alpha.\beta.\beta)$
$\xrightarrow{\bullet b} (3, \alpha.\beta) \xrightarrow{\bullet a} (2, \alpha)$



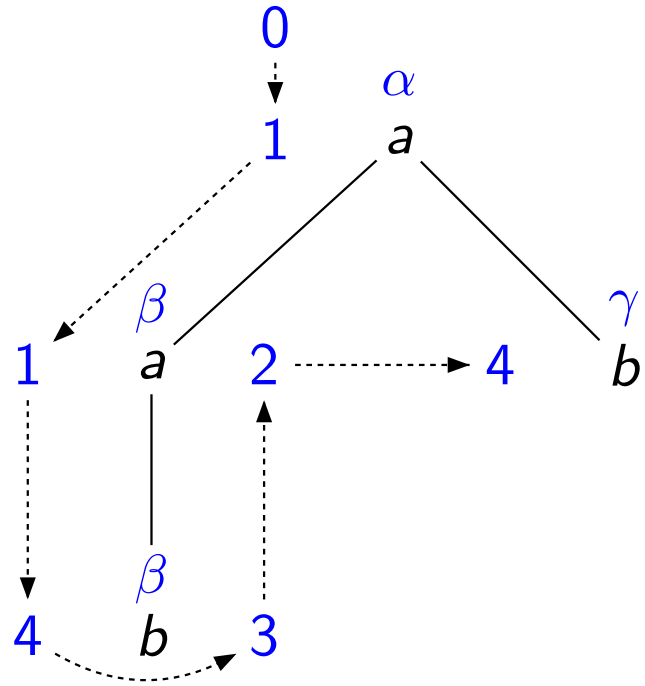
VPAs as Tree Automata

- a run of a VPA on a tree consists in:
  - when opening node π: update the current state, and assign $\gamma \in \Gamma$ to π
  - when closing node π: update the current state according to γ



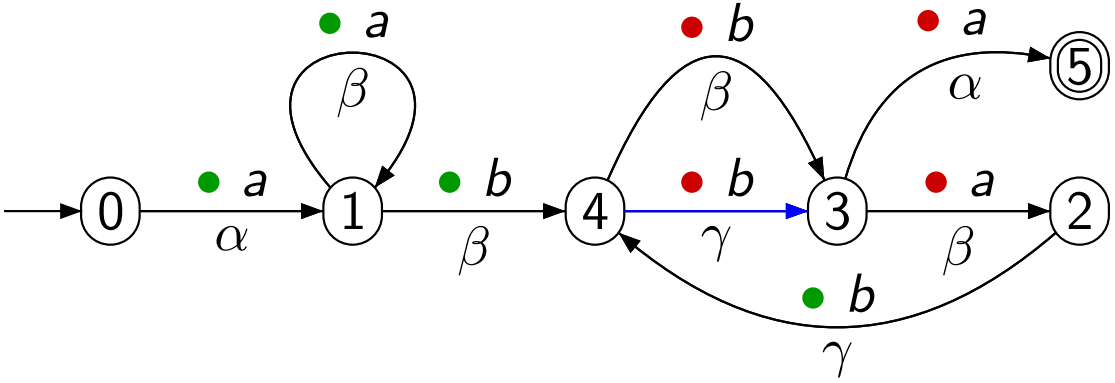
A: VPA on $\mathcal{T}_\Sigma$ with $\Sigma = \{a, b\}$
and $Q = \{0, 1, 2, 3, 4, 5\}$
and $\Gamma = \{\alpha, \beta, \gamma\}$

$$(0, \emptyset) \xrightarrow{\bullet a} (1, \alpha) \xrightarrow{\bullet a} (1, \alpha.\beta) \xrightarrow{\bullet b} (4, \alpha.\beta.\beta)$$
$$\xrightarrow{\bullet b} (3, \alpha.\beta) \xrightarrow{\bullet a} (2, \alpha) \xrightarrow{\bullet b} (4, \alpha.\gamma)$$



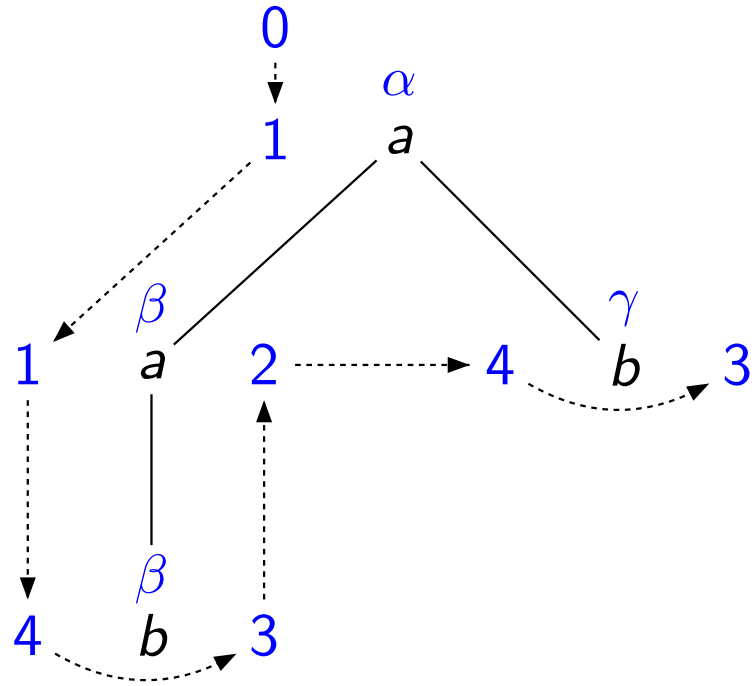
VPAs as Tree Automata

- a run of a VPA on a tree consists in:
  - when opening node π: update the current state, and assign $\gamma \in \Gamma$ to π
  - when closing node π: update the current state according to γ



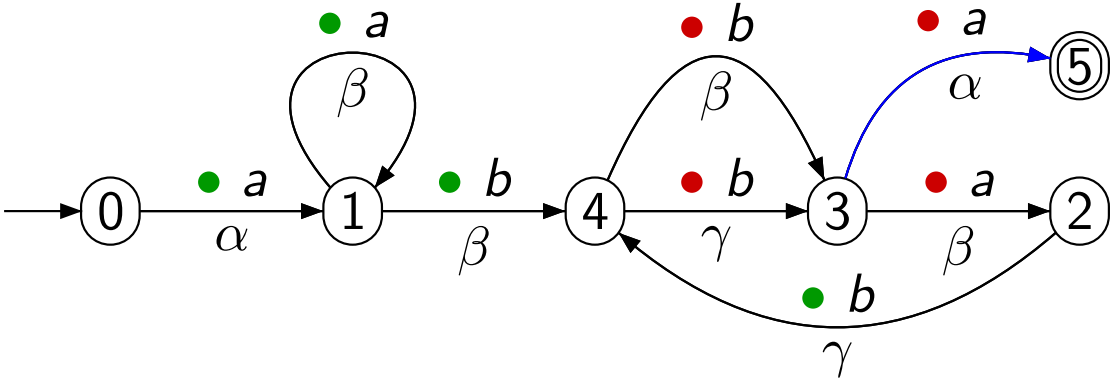
A: VPA on $\mathcal{T}_\Sigma$ with $\Sigma = \{a, b\}$
and $Q = \{0, 1, 2, 3, 4, 5\}$
and $\Gamma = \{\alpha, \beta, \gamma\}$

$(0, \emptyset) \xrightarrow{\bullet a} (1, \alpha) \xrightarrow{\bullet a} (1, \alpha.\beta) \xrightarrow{\bullet b} (4, \alpha.\beta.\beta)$
$\xrightarrow{\bullet b} (3, \alpha.\beta) \xrightarrow{\bullet a} (2, \alpha) \xrightarrow{\bullet b} (4, \alpha.\gamma)$
$\xrightarrow{\bullet b} (3, \alpha)$



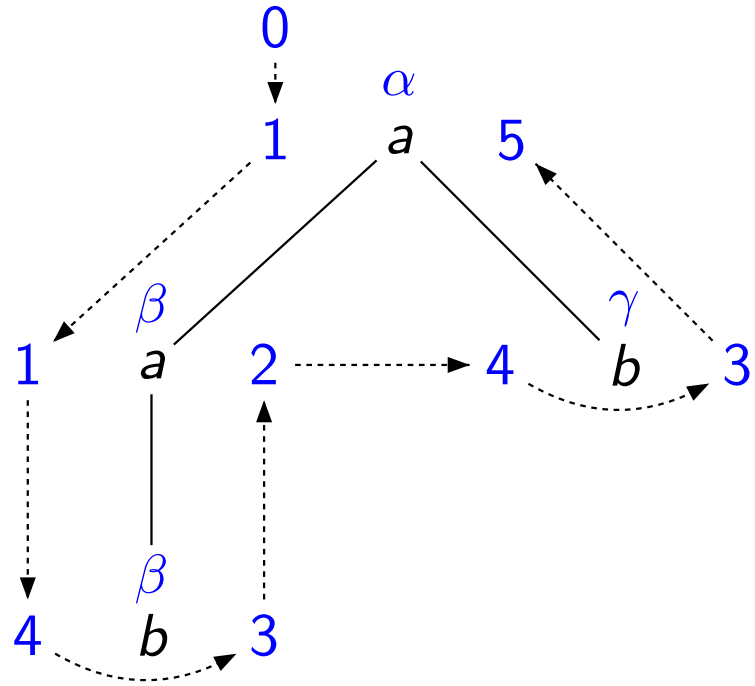
VPAs as Tree Automata

- a run of a VPA on a tree consists in:
  - when opening node π: update the current state, and assign $\gamma \in \Gamma$ to π
  - when closing node π: update the current state according to γ



A: VPA on $\mathcal{T}_\Sigma$ with $\Sigma = \{a, b\}$
and $Q = \{0, 1, 2, 3, 4, 5\}$
and $\Gamma = \{\alpha, \beta, \gamma\}$

$(0, \emptyset) \xrightarrow{\bullet a} (1, \alpha) \xrightarrow{\bullet a} (1, \alpha.\beta) \xrightarrow{\bullet b} (4, \alpha.\beta.\beta)$
$\xrightarrow{\bullet b} (3, \alpha.\beta) \xrightarrow{\bullet a} (2, \alpha) \xrightarrow{\bullet b} (4, \alpha.\gamma)$
$\xrightarrow{\bullet b} (3, \alpha) \xrightarrow{\bullet a} (5, \emptyset)$



VPAs: properties & related models

From now on, we will consider VPAs as tree automata.

Translations between VPAs and $\uparrow$TA $\circ$*curry* exist [Gau09], so:

- VPAs are as expressive as the 4 classes using $\{\uparrow \text{TA}, \downarrow \text{TA}\} + \{\mathit{fcns}, \mathit{curry}\}$
- hedge automata also have this expressiveness
- we will call an unranked tree language recognizable if it belongs to this class
- other equivalent models (in expressiveness):
  - nested word automata [Alu07]: reformulation of VPAs
  - pushdown forest automata [NS98]: VPAs on forests (i.e. hedges)
  - streaming tree automata [GNR08, Gau09]: VPAs on trees

VPAs: determinism

the class of deterministic VPAs (dVPAs) has numerous interesting properties:

- as expressive as VPAs
  - so, as string acceptors: $NFAs \subsetneq VPAs = dVPAs \subsetneq dPAs \subsetneq PAs$
- determinization procedure in $O(2^{|Q|^2})$
- corresponds to streaming XML deterministically
  - yardstick class for streamability

VPAs: determinization

VPA $A = (Q, I, F, \Gamma, \Delta, \Sigma \uplus \overline{\Sigma})$ recognizing encodings of trees
(see [AM04] otherwise)

For an hedge h, let
$acc_A(h) = \{(q, q') \in Q^2 \mid \text{there is a run of } A \text{ on } h \text{ from } q \text{ to } q'\}$

The determinization procedure computes $acc_A(h)$ for all hedges h of the tree t. More precisely, the current state at node π is $acc(h)$ where h is the hedge of left siblings of π:

- when opening a node,
  - the previous state is pushed on the stack
  - the current state is set to the identity of Q^2 (h=empty hedge)
- when closing a node, the current state is updated from:
  - the top of the stack, i.e. hedge accessibility before traversing π
  - the previous state, i.e. hedge accessibility through $t.\pi$

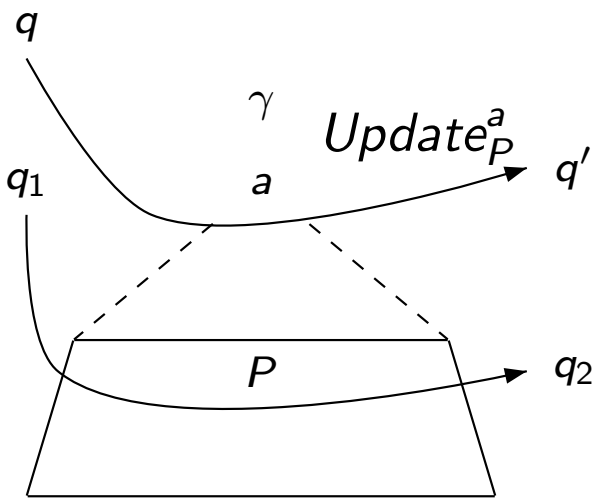
Determinization

$$Q^{A'} = 2^{Q^A \times Q^A}$$
$$I^{A'} = id_{I^A}$$
$$F^{A'} = \{P \mid \pi_2(P) \cap F^A \neq \emptyset\}$$

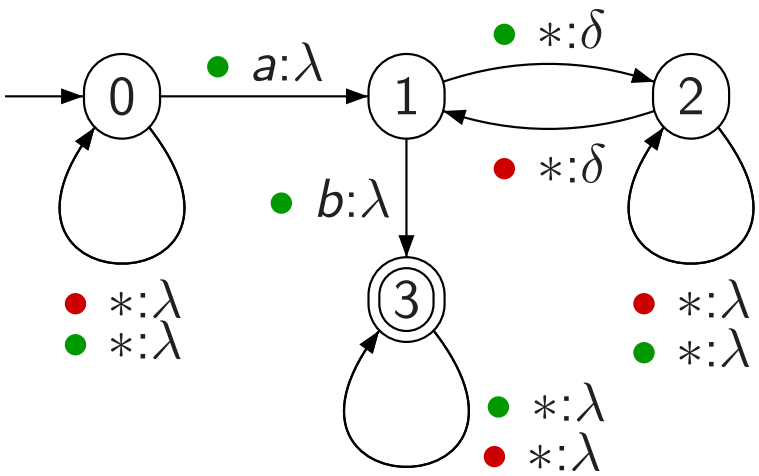
$$\frac{a \in \Sigma \qquad P \in Q^{A'}}{P \xrightarrow{\bullet\ a:P} id_{Q^A} \in \Delta^{A'}} \qquad \frac{a \in \Sigma \qquad P, P' \subseteq Q^A}{P \xrightarrow{\bullet\ a:P'} P' \circ Update^a_P \in \Delta^{A'}}$$

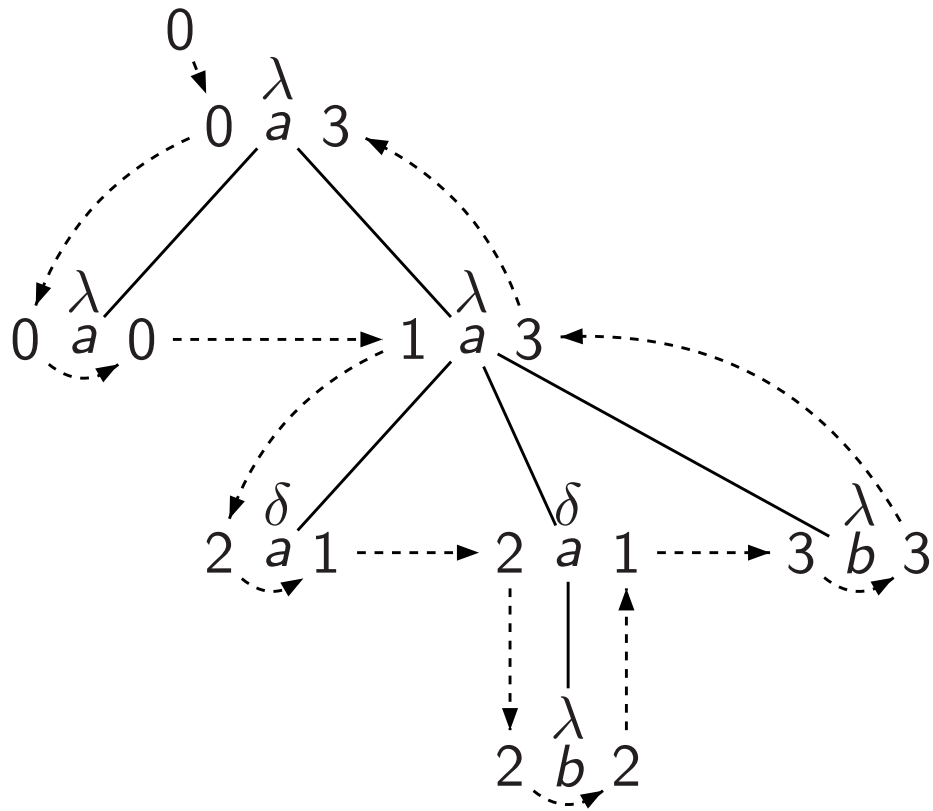
with

$$Update^a_P = \{(q, q') \mid \exists (q_1, q_2) \in P.\ \exists \gamma.\ q \xrightarrow{\bullet\ a:\gamma} q_1 \in \Delta^A \ \&\ q_2 \xrightarrow{\bullet\ a:\gamma} q' \in \Delta^A\}$$



Determinization: example





exercise: run of $det(A)$ on t

Linearization

Place of recognizable languages

$$NFAs \subsetneq VPAs = dVPAs \subsetneq dPAs \subsetneq PAs = CFLs$$

	closed by			det.	decidable		
	$\cap$	$\cup$	compl.		$\mathcal{L}(A)=\emptyset$	$\mathcal{L}(A)=\Sigma^*_{wf}$ $\mathcal{L}(A)=\mathcal{L}(B)$	$\mathcal{L}(A)\subseteq\mathcal{L}(B)$
NFAs	✓	✓	✓	✓	✓	✓	✓
VPAs	✓	✓	✓	✓	✓	✓	✓
dPAs	x	x	✓	✓	✓	✓	x
PAs	x	✓	x	x	✓	x	x

Minimization

of unranked tree automata

	unique minimal automaton?	procedure cost	ref.
det. hedge automata using DFAs for horiz. lang.	x	not PTIME unless PTIME=NP	[MN07]
$d \uparrow \mathrm{TA} \circ \mathit{curry}$ = stepwise tree automata	✓	PTIME	[MN07]
$d \uparrow \mathrm{TA} \circ \mathit{fcns}$	✓	PTIME	[MN07]
dVPAs	x	*open?*	[AKMV05] [CW07]

Congruence of a language $L \subseteq \hat{\Sigma}^*$ with $\hat{\Sigma} = \Sigma_{push} \uplus \Sigma_{pop}$

For well-matched words w and w',

$$w \equiv_L w' \quad \Leftrightarrow \quad \forall u, v \in \hat{\Sigma}^*,\ uwv \in L \text{ iff } uw'v \in L$$

A well-matched language $L \subseteq \hat{\Sigma}^*$ is VPA-recognizable iff $\equiv_A$ is of finite index.
This permits to define canonical VPAs, but not minimal.

1. Ranked Trees
   - Trees on Ranked Alphabet
   - Tree Automata
   - Tree Grammars
   - Logic

2. Unranked Trees
   - Unranked Trees
   - Automata
   - Logic

MSO

Equivalence with recognizable tree languages

for unranked trees, use first-child/next-sibling predicates:
$\Omega_u = \{\mathsf{lab}_a \mid a \in \Sigma\} \cup \{fc, ns\}$

Equivalence with automata

A tree language $L \subseteq \mathcal{T}_\Sigma$ is recognizable iff $\exists \phi \in MSO[\Omega_u]$ s.t. $L = \mathcal{L}_\phi$.

($\Rightarrow$) Similar to the ranked case: define a formula recognizing runs.

($\Leftarrow$) From ϕ, define ϕ' recognizing $fcns(\mathcal{L}_\phi)$. Then use equivalence for ranked trees.

XPath

XPath $\rightarrow$ det. automata: in $O(2^{2^{|e|}})$

★★★

For the fragment k-Downward XPath, it is in PTIME.

k-Downward XPath

Syntax

axis	d	$::=$	$self \mid ch \mid ch^*$
steps	S	$::=$	$d::a \mid d::* \quad$ (where $a \in \Sigma$)
paths	P	$::=$	$S \mid P[F] \mid P_1/P_2$
filters	F	$::=$	$P \mid \neg F \mid F_1 \wedge F_2$
rooted paths	R	$::=$	$/P$

k-Downward XPath

Syntax

axis	d	$::=$	$self \mid ch \mid ch^*$
steps	S	$::=$	$d::a \mid d::* \quad$ (where $a \in \Sigma$)
paths	P	$::=$	$S \mid P[F] \mid P_1/P_2$
filters	F	$::=$	$P \mid \neg F \mid F_1 \wedge F_2$
rooted paths	R	$::=$	$/P$

Semantic

$$[\![d::*]\!]_{path}(t) = d^t$$
$$[\![d::a]\!]_{path}(t) = \{(\pi, \pi') \in d^t \mid \mathsf{lab}_a^t(\pi')\}$$
$$[\![P_1/P_2]\!]_{path}(t) = [\![P_1]\!]_{path}(t) \circ [\![P_2]\!]_{path}(t)$$
$$[\![P[F]]\!]_{path}(t) = \{(\pi, \pi') \in [\![P]\!]_{path}(t) \mid \pi' \in [\![F]\!]_{filter}(t)\}$$

k-Downward XPath

Syntax

axis	d	$::=$	$self \mid ch \mid ch^*$
steps	S	$::=$	$d::a \mid d::* \quad$ (where $a \in \Sigma$)
paths	P	$::=$	$S \mid P[F] \mid P_1/P_2$
filters	F	$::=$	$P \mid \neg F \mid F_1 \wedge F_2$
rooted paths	R	$::=$	$/P$

Semantic

$$[\![d::*]\!]_{path}(t) = d^t$$
$$[\![d::a]\!]_{path}(t) = \{(\pi, \pi') \in d^t \mid \mathsf{lab}_a^t(\pi')\}$$
$$[\![P_1/P_2]\!]_{path}(t) = [\![P_1]\!]_{path}(t) \circ [\![P_2]\!]_{path}(t)$$
$$[\![P[F]]\!]_{path}(t) = \{(\pi, \pi') \in [\![P]\!]_{path}(t) \mid \pi' \in [\![F]\!]_{filter}(t)\}$$
$$[\![P]\!]_{filter}(t) = \{\pi \mid \exists \pi'.\ (\pi, \pi') \in [\![P]\!]_{path}(t)\}$$
$$[\![\neg F]\!]_{filter}(t) = \mathsf{nodes} \setminus [\![F]\!]_{filter}(t)$$
$$[\![F_1 \wedge F_2]\!]_{filter}(t) = [\![F_1]\!]_{filter}(t) \cap [\![F_2]\!]_{filter}(t)$$
$$[\![/P]\!]_{filter}(t) = \{\pi \mid (root, \pi) \in [\![P]\!]_{path}(t)\}$$

k-Downward XPath

Restrictions:

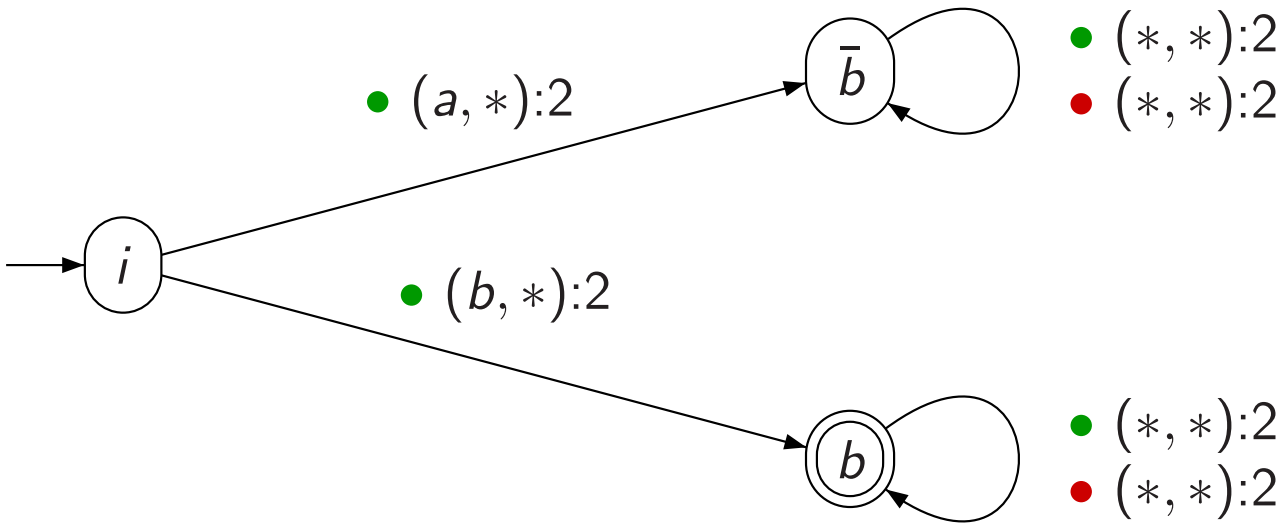
- $|\text{conjunctions} + \text{filters}| \leq k$
- if $ch^*::a$ appears, then there are no 2 a-nodes on the same branch

k-Downward XPath $\rightarrow$ dVPA

- inductive construction
- at each step, automata are deterministic and pseudo-complete:
  $\rightarrow$ for every tree t, there is exactly one run on t.

Example: $[ch::b]$

First step: A_b checks whether the root is labeled by b



Example: $[ch::b]$

Second step: $A_{ch::b}$ runs A_b on every child of the root

Procedure:

- add 3 states: `start`, 0 and 1
- add the rules inferred from what follows
  - this builds rules for $F = [ch[F']]$

Example: $[ch::b]$

Second step: $A_{ch::b}$ runs A_b on every child of the root

Procedure:

- add 3 states: `start`, 0 and 1
- add the rules inferred from what follows
  - this builds rules for $F = [ch[F']]$

$$\frac{a \in \Sigma \qquad V \in \{0,1\}}{\texttt{start} \xrightarrow{\bullet\ (a,V):0} 0}$$

opening the root: move to 0

$$\frac{q_1 \xrightarrow{\bullet\ (a,V):\gamma} q_2 \in \Delta^{A'} \qquad q_1 \in I^{A'} \qquad \flat \in \{0,1\}}{\flat \xrightarrow{\bullet\ (a,V):\ \flat} q_2}$$

opening a child: start testing F'

$$\frac{q_1 \xrightarrow{\alpha\ (a,V):\gamma} q_2 \in \Delta^{A'}}{q_1 \xrightarrow{\alpha\ (a,V):\gamma} q_2}$$

run test of F'

Example: $[ch::b]$

$$\frac{q_1 \xrightarrow{\bullet\ (a,V):\gamma} q_2 \in \Delta^{A'} \qquad q_2 \notin F^{A'} \qquad q_1' \xrightarrow{\bullet\ (a,V):\gamma} q_2' \in \Delta^{A'} \qquad q_1' \in I^{A'} \qquad \flat \in \{0,1\}}{q_1 \xrightarrow{\bullet\ (a,V):\ \flat} \flat}$$

failure of F': no new match

$$\frac{q_1 \xrightarrow{\bullet\ (a,V):\gamma} q_2 \in \Delta^{A'} \qquad q_2 \in F^{A'} \qquad q_1' \xrightarrow{\bullet\ (a,V):\gamma} q_2' \in \Delta^{A'} \qquad q_1' \in I^{A'} \qquad \flat \in \{0,1\}}{q_1 \xrightarrow{\bullet\ (a,V):\ \flat} 1}$$

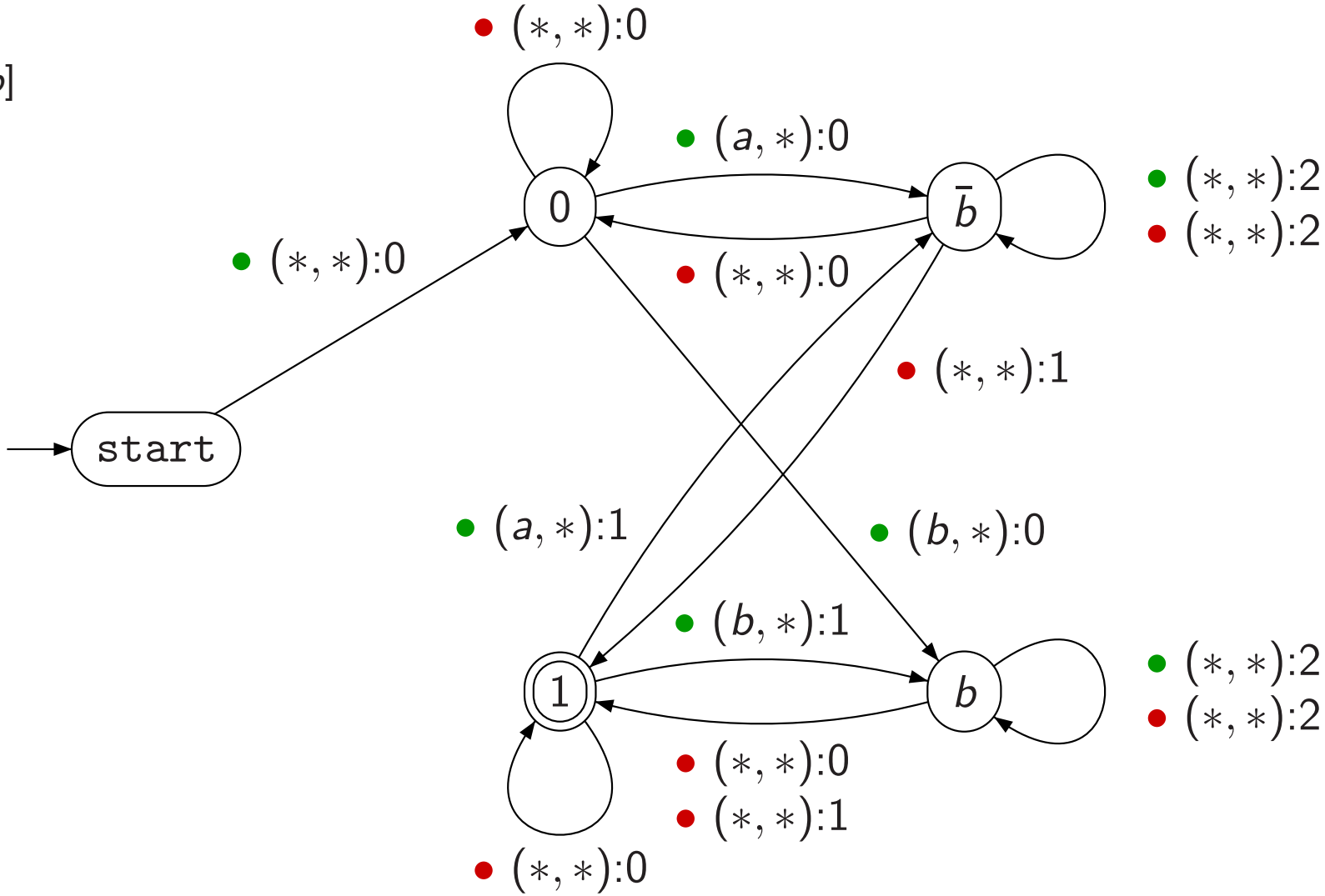
success of F': move to 1

$$\frac{a \in \Sigma \qquad V \in \{0,1\} \qquad \flat \in \{0,1\}}{\flat \xrightarrow{\bullet\ (a,V):0} \flat}$$

closing the root

Example: $[ch::b]$

$A_{[ch::b]}$



k-Downward XPath $\rightarrow$ dVPAs

other steps

- ch^* is similar to ch
- P_1/P_2 is transformed into filter (w/ variables): $ch^*::a/ch::b$ becomes $[ch^*::a[ch::b[x]]]$
- $A_{F_1 \wedge F_2} = A_{F_1} \wedge A_{F_2}$ (product construction)
- $A_{\neg F} = \neg A_F$

Other logics

- Conditional XPath, Regular XPath
- other modal logics: Tree TL, CTL*, PDL$_{tree}$
- μ-calculus

Language, Automata and Logic for Finite Trees

Olivier Gauwin

UMons

Feb/March 2010

References

[AKMV05] Rajeev Alur, Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan.
Congruences for visibly pushdown languages.
In *Automata, Languages and Programming, 32nd International Colloquium*, volume 3580 of *Lecture Notes in Computer Science*, pages 1102–1114. Springer Verlag, 2005.

[Alu07] Rajeev Alur.
Marrying words and trees.
In *26th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 233–242. ACM-Press, 2007.

[AM04] Rajeev Alur and P. Madhusudan.
Visibly pushdown languages.
In *36th ACM Symposium on Theory of Computing*, pages 202–211. ACM-Press, 2004.

[BKWM01] Anne Brüggemann-Klein, Derick Wood, and Makoto Murata.

Regular tree and regular hedge languages over unranked alphabets: Version 1, April 07 2001.

[BLN07] Michael Benedikt, Leonid Libkin, and Frank Neven.
Logical definability and query languages over ranked and unranked trees.
ACM Transactions on Computational Logics, 8(2), April 2007.

[BPSM+08] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau.
Extensible Markup Language (XML) 1.0 (Fifth Edition), November 2008.
http://www.w3.org/TR/2008/REC-xml-20081126/.

[CDG+07] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi.
Tree automata techniques and applications.
Available online since 1997:
http://tata.gforge.inria.fr, October 2007.

Revised October, 12th 2007.

[CNT04] Julien Carme, Joachim Niehren, and Marc Tommasi.
Querying unranked trees with stepwise tree automata.
In *19th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 105–118. Springer Verlag, 2004.

[CW07] Patrick Chervet and Igor Walukiewicz.
Minimizing variants of visibly pushdown automata.
In *Mathematical Foundations of Computer Science*, volume 4708 of *Lecture Notes in Computer Science*, pages 135–146. Springer Verlag, 2007.

[Don70] John E. Doner.
Tree acceptors and some of their applications.
4:406–451, 1970.

[Gau09] Olivier Gauwin.
Streaming Tree Automata and XPath.
PhD thesis, Université Lille 1, 2009.

[GNR08] Olivier Gauwin, Joachim Niehren, and Yves Roos.
Streaming tree automata.
Information Processing Letters, 109(1):13–17, December 2008.

[Koc03] Christoph Koch.
Efficient processing of expressive node-selecting queries on XML data in secondary storage: A tree automata-based approach.
In *Proc. VLDB 2003*, 2003.

[MN07] Wim Martens and Joachim Niehren.
On the minimization of XML schemas and tree automata for unranked trees.
Journal of Computer and System Science, 73(4):550–583, 2007.

[NS98] Andreas Neumann and Helmut Seidl.
Locating matches of tree patterns in forests.
In *18th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1530 of *Lecture*

Notes in Computer Science, pages 134–145. Springer Verlag, 1998.

[Rab69] Michael O. Rabin.
Decidability of Second-Order Theories and Automata on Infinite Trees.
Transactions of the American Mathematical Society, 141:1–35, 1969.

[TW68] J. W. Thatcher and J. B. Wright.
Generalized finite automata with an application to a decision problem of second-order logic.
Mathematical System Theory, 2:57–82, 1968.