Language, Automata and Logic for Finite Trees

Olivier Gauwin

UMons

Feb/March 2010
Example for regular word languages: $\Sigma^* . a.b.\Sigma^*$

Grammar:

\[
\begin{align*}
S & \to aS \\
S & \to aB \\
X & \to aX \\
X & \to \epsilon \\
S & \to bS \\
B & \to bX \\
X & \to bX
\end{align*}
\]
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

Automata

Grammar / Expressions

Logic

Complexity

$\exists x. \exists y. \text{lab}_a(x) \land \text{lab}_b(y) \land \text{succ}(x, y)$

$\exists x. \exists y. \text{lab}_a(x) \land \text{lab}_b(y) \land \text{child}(x, y)$
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
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

\[
\Sigma_r = \text{finite alphabet} + \text{arity function} \\
\Sigma = \{a, b, c\} \quad \text{ar} : \Sigma \rightarrow \mathbb{N} \\
ar(a) = 2 \\
ar(b) = 2 \\
ar(c) = 0
\]

Ranked trees over \(\Sigma_r\)

\(\mathcal{T}_{\Sigma_r}\), the set of ranked trees, is the smallest set of terms \(f(t_1, \ldots, t_k)\) such that: \(f \in \Sigma_r\), \(k = \text{ar}(f)\), 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 $(\text{nodes}^t, \{\text{lab}_a^t, \text{lab}_b^t, \text{lab}_c^t, \text{ch}_1^t, \text{ch}_2^t\})$.

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

\begin{itemize}
  \item $\text{lab}_a^t = \{1\}$
  \item $\text{lab}_b^t = \{\epsilon, 1.1, 2\}$
  \item $\text{lab}_c^t = \{1.1.1, 1.1.2, 1.2, 2.1, 2.2\}$
  \item $\text{ch}_1^t = \{(\epsilon, 1), (1, 1.1), (2, 2.1) \ldots \}$
  \item $\text{ch}_2^t = \{(\epsilon, 2), (1, 1.2), (2, 2.2) \ldots \}$
\end{itemize}
1 Ranked Trees
- Trees on Ranked Alphabet
- Tree Automata
- Tree Grammars
- Logic

2 Unranked Trees
- Unranked Trees
- Automata
- Logic
**Definition**

A tree automaton (TA) over $\Sigma_r = (\Sigma, \text{ar})$ 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,
- $\Delta$ are rules of type: $a(q_1, \ldots, q_k) \rightarrow q$
  with $a \in \Sigma$, $k = \text{ar}(a)$ and $q, q_1, \ldots, q_k \in Q$

**Runs**

A run $\rho$ of $A$ on $t$ is a function $\rho : \text{nodes}^t \rightarrow Q$ such that:

if $\pi \in \text{nodes}^t$ with children $\pi_1, \ldots, \pi_k$ and label $a$ then $a(\rho(\pi_1), \ldots, \rho(\pi_k)) \rightarrow \rho(\pi) \in \Delta$
Bottom-up vs top-down

Bottom-up view

\[ a(q_1, \ldots, q_k) \rightarrow q \text{ is a bottom-up point of view:} \]

```
a(q_s, q_x) \rightarrow q_s
a(q_x, q_s) \rightarrow q_s
b(q_s, q_x) \rightarrow q_s
b(q_x, q_s) \rightarrow q_s
a(q_B, q_x) \rightarrow q_s
a(q_x, q_B) \rightarrow q_s
b(q_x, q_x) \rightarrow q_B
a(q_x, q_x) \rightarrow q_X
b(q_x, q_x) \rightarrow q_X
c \rightarrow q_X
```

\[
\text{Rules:}
\]

```
\text{Bottom-up view}
```

Olivier Gauwin (UMons)
Bottom-up vs top-down

Bottom-up view

\[ a(q_1, \ldots, q_k) \rightarrow q \] is a **bottom-up** point of view:

\[
\begin{align*}
\text{Rules:} & \\
\{ & \\
&a(q_s, q_x) \rightarrow q_s \\
&a(q_x, q_s) \rightarrow q_s \\
&b(q_s, q_x) \rightarrow q_s \\
&b(q_x, q_s) \rightarrow q_s \\
&a(q_B, q_x) \rightarrow q_s \\
&a(q_x, q_B) \rightarrow q_s \\
&b(q_x, q_x) \rightarrow q_B \\
&a(q_x, q_x) \rightarrow q_X \\
&b(q_x, q_x) \rightarrow q_X \\
&c \rightarrow q_X
\end{align*}
\]
Bottom-up vs top-down

Bottom-up view

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

- **Rules:**
  - \( a(q_s, q_x) \rightarrow q_s \)
  - \( a(q_x, q_s) \rightarrow q_s \)
  - \( b(q_s, q_x) \rightarrow q_s \)
  - \( b(q_x, q_s) \rightarrow q_s \)
  - \( a(q_B, q_x) \rightarrow q_s \)
  - \( a(q_x, q_B) \rightarrow q_s \)
  - \( b(q_x, q_x) \rightarrow q_B \)
  - \( a(q_x, q_x) \rightarrow q_x \)
  - \( b(q_x, q_x) \rightarrow q_x \)
  - \( c \rightarrow q_x \)
Bottom-up vs top-down

Bottom-up view

\[ a(q_1, \ldots, q_k) \rightarrow q \text{ is a bottom-up point of view:} \]

\[
\begin{align*}
\text{Rules: } & \quad a(q_s, q_x) \rightarrow q_s \\
& \quad a(q_x, q_s) \rightarrow q_s \\
& \quad b(q_s, q_x) \rightarrow q_s \\
& \quad b(q_x, q_s) \rightarrow q_s \\
& \quad a(q_B, q_x) \rightarrow q_s \\
& \quad a(q_x, q_B) \rightarrow q_s \\
& \quad b(q_x, q_x) \rightarrow q_B \\
& \quad a(q_x, q_x) \rightarrow q_B \\
& \quad b(q_x, q_x) \rightarrow q_B \\
& \quad a(q_x, q_B) \rightarrow q_B \\
& \quad b(q_x, q_B) \rightarrow q_B \\
& \quad a(q_B, q_x) \rightarrow q_B \\
& \quad b(q_B, q_x) \rightarrow q_B \\
& \quad c \rightarrow q_B
\end{align*}
\]
Bottom-up vs top-down

Bottom-up view

\[ a(q_1, \ldots, q_k) \rightarrow q \text{ is a bottom-up point of view:} \]

\[
\begin{align*}
\text{Rules:} & \quad \begin{cases}
    a(q_s, q_x) \rightarrow q_s \\
    a(q_x, q_s) \rightarrow q_s \\
    b(q_s, q_x) \rightarrow q_s \\
    b(q_x, q_s) \rightarrow q_s \\
    a(q_B, q_x) \rightarrow q_s \\
    a(q_x, q_B) \rightarrow q_s \\
    b(q_x, q_x) \rightarrow q_B \\
    a(q_x, q_x) \rightarrow q_x \\
    b(q_x, q_x) \rightarrow q_x \\
    c \rightarrow q_x
\end{cases}
\end{align*}
\]
Bottom-up vs top-down

Bottom-up view

\[ a(q_1, \ldots, q_k) \rightarrow q \] is a \textbf{bottom-up} point of view:

Rules:

\[
\begin{align*}
    a(q_s, q_x) & \rightarrow q_s \\
    a(q_x, q_s) & \rightarrow q_s \\
    b(q_s, q_x) & \rightarrow q_s \\
    b(q_x, q_s) & \rightarrow q_s \\
    a(q_B, q_x) & \rightarrow q_s \\
    a(q_x, q_B) & \rightarrow q_s \\
    b(q_x, q_x) & \rightarrow q_B \\
    a(q_x, q_x) & \rightarrow q_X \\
    b(q_x, q_x) & \rightarrow q_X \\
    c & \rightarrow q_X
\end{align*}
\]
Bottom-up vs top-down

Bottom-up view

\[ a(q_1, \ldots, q_k) \rightarrow q \text{ is a bottom-up point of view:} \]

Rules:
\[
\begin{align*}
    a(q_s, q_x) &\rightarrow q_s \\
    a(q_x, q_s) &\rightarrow q_s \\
    b(q_s, q_x) &\rightarrow q_s \\
    b(q_x, q_s) &\rightarrow q_s \\
    a(q_B, q_x) &\rightarrow q_s \\
    a(q_x, q_B) &\rightarrow q_s \\
    b(q_x, q_x) &\rightarrow q_B \\
    a(q_x, q_x) &\rightarrow q_x \\
    b(q_x, q_x) &\rightarrow q_x \\
    c &\rightarrow q_x \\
\end{align*}
\]

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: \( a(q) \rightarrow (q_1, \ldots, q_k) \), and name \( F \) the initial states.

This corresponds to a top-down definition:

Initial: \( \{ q_S \} \)

Rules:

\[
\begin{align*}
  a(q_S) & \rightarrow (q_S, q_X) \\
  a(q_S) & \rightarrow (q_X, q_S) \\
  b(q_S) & \rightarrow (q_S, q_X) \\
  b(q_S) & \rightarrow (q_X, q_S) \\
  a(q_S) & \rightarrow (q_B, q_X) \\
  a(q_S) & \rightarrow (q_X, q_B) \\
  b(q_B) & \rightarrow (q_X, q_X) \\
  a(q_X) & \rightarrow (q_X, q_X) \\
  b(q_X) & \rightarrow (q_X, q_X) \\
  c(q_X) & \\
\end{align*}
\]
Bottom-up vs top-down

Top-down view

We could have written rules this way: \( a(q) \rightarrow (q_1, \ldots, q_k) \), and name \( F \) the initial states.
This corresponds to a top-down definition:

Initial: \( \{ q_S \} \)

Rules:

\[
\begin{align*}
  a(q_S) &\rightarrow (q_S, q_X) \\
  a(q_S) &\rightarrow (q_X, q_S) \\
  b(q_S) &\rightarrow (q_S, q_X) \\
  b(q_S) &\rightarrow (q_X, q_S) \\
  a(q_S) &\rightarrow (q_B, q_X) \\
  a(q_S) &\rightarrow (q_X, q_B) \\
  b(q_B) &\rightarrow (q_X, q_X) \\
  a(q_X) &\rightarrow (q_X, q_X) \\
  b(q_X) &\rightarrow (q_X, q_X) \\
  c(q_X) &
\end{align*}
\]
We could have written rules this way: \( a(q) \rightarrow (q_1, \ldots, q_k) \), and name \( F \) the initial states.

This corresponds to a top-down definition:

**Initial:** \( \{ q_S \} \)

**Rules:**

- \( a(q_S) \rightarrow (q_S, q_X) \)
- \( a(q_S) \rightarrow (q_X, q_S) \)
- \( b(q_S) \rightarrow (q_S, q_X) \)
- \( b(q_S) \rightarrow (q_X, q_S) \)
- \( a(q_S) \rightarrow (q_B, q_X) \)
- \( a(q_S) \rightarrow (q_X, q_B) \)
- \( b(q_B) \rightarrow (q_X, q_X) \)
- \( a(q_X) \rightarrow (q_X, q_X) \)
- \( b(q_X) \rightarrow (q_X, q_X) \)
- \( c(q_X) \)
Bottom-up vs top-down

Top-down view

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

This corresponds to a top-down definition:

Initial: \[ \{q_S\} \]

Rules:

\begin{align*}
  a(q_S) &\rightarrow (q_S, q_X) \\
  a(q_S) &\rightarrow (q_X, q_S) \\
  b(q_S) &\rightarrow (q_S, q_X) \\
  b(q_S) &\rightarrow (q_X, q_S) \\
  a(q_S) &\rightarrow (q_B, q_X) \\
  a(q_S) &\rightarrow (q_X, q_B) \\
  b(q_B) &\rightarrow (q_X, q_X) \\
  a(q_X) &\rightarrow (q_X, q_X) \\
  b(q_X) &\rightarrow (q_X, q_X) \\
  c(q_X) & \\
\end{align*}
Bottom-up vs top-down

Top-down view

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

This corresponds to a top-down definition:

- **Initial:** \( \{ q_S \} \)
- **Rules:**
  - \( a(q_S) \rightarrow (q_S, q_X) \)
  - \( a(q_S) \rightarrow (q_X, q_S) \)
  - \( b(q_S) \rightarrow (q_S, q_X) \)
  - \( b(q_S) \rightarrow (q_X, q_S) \)
  - \( a(q_S) \rightarrow (q_B, q_X) \)
  - \( a(q_S) \rightarrow (q_X, q_B) \)
  - \( b(q_B) \rightarrow (q_X, q_X) \)
  - \( a(q_X) \rightarrow (q_X, q_X) \)
  - \( b(q_X) \rightarrow (q_X, q_X) \)
  - \( c(q_X) \)
Bottom-up vs top-down

Top-down view

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

This corresponds to a top-down definition:

\[
\begin{align*}
\text{Initial: } & \{ q_s \} \\
\text{Rules: } & \\
& \begin{align*}
ad(q_s) & \rightarrow (q_s, q_x) \\
ad(q_s) & \rightarrow (q_x, q_s) \\
b(q_s) & \rightarrow (q_s, q_x) \\
b(q_s) & \rightarrow (q_x, q_s) \\
a(q_s) & \rightarrow (q_B, q_x) \\
a(q_s) & \rightarrow (q_x, q_B) \\
b(q_B) & \rightarrow (q_x, q_x) \\
a(q_x) & \rightarrow (q_x, q_x) \\
b(q_x) & \rightarrow (q_x, q_x) \\
c(q_x) & \\
\end{align*}
\end{align*}
\]
Bottom-up vs top-down

Top-down view

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

This corresponds to a top-down definition:

- **Initial:** \( \{ q_s \} \)
- **Rules:**
  - \( a(q_s) \rightarrow (q_s, q_x) \)
  - \( a(q_s) \rightarrow (q_x, q_s) \)
  - \( b(q_s) \rightarrow (q_s, q_x) \)
  - \( b(q_s) \rightarrow (q_x, q_s) \)
  - \( a(q_s) \rightarrow (q_B, q_x) \)
  - \( a(q_s) \rightarrow (q_x, q_B) \)
  - \( b(q_B) \rightarrow (q_x, q_x) \)
  - \( a(q_x) \rightarrow (q_x, q_x) \)
  - \( b(q_x) \rightarrow (q_x, q_x) \)
  - \( c(q_X) \)
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:

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

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

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

a subset construction, very similar to the determinization of NFAs:
- $Q_d = 2^Q$
- $\Delta_d = \{ a(s_1, \ldots, s_n) \rightarrow s \mid s = \{ q \in Q \mid \exists q_1 \in s_1, \ldots, \exists q_n \in s_n, f(q_1, \ldots, q_n) \rightarrow q \in \Delta \} \}$
- $F_d = \{ S \mid S \subseteq Q \text{ and } S \cap F \neq \emptyset \}$

This simulates all runs of $A$. 

Determinization
of $\uparrow$TA
Determinization

Example run

\[ F = \{ q_s \} \]

Rules of ↑TA:

\[
\begin{align*}
    a(q_s, q_x) & \rightarrow q_s \\
    a(q_x, q_s) & \rightarrow q_s \\
    b(q_s, q_x) & \rightarrow q_s \\
    b(q_x, q_s) & \rightarrow q_s \\
    a(q_B, q_x) & \rightarrow q_s \\
    a(q_x, q_B) & \rightarrow q_s \\
    b(q_x, q_x) & \rightarrow q_B \\
    a(q_x, q_x) & \rightarrow q_x \\
    b(q_x, q_x) & \rightarrow q_x \\
    c & \rightarrow q_x
\end{align*}
\]
Determinization

Example run

\[
F = \{ qs \}
\]

Rules of ↑TA:

\[
\begin{align*}
\text{a}(qs, qx) & \rightarrow qs \\
\text{a}(qx, qs) & \rightarrow qs \\
b(qs, qx) & \rightarrow qs \\
b(qx, qs) & \rightarrow qs \\
a(qB, qx) & \rightarrow qs \\
a(qx, qB) & \rightarrow qs \\
b(qx, qx) & \rightarrow qB \\
a(qx, qx) & \rightarrow qx \\
b(qx, qx) & \rightarrow qx \\
c & \rightarrow qx
\end{align*}
\]
Determinization

Example run

\[ F = \{ q_s \} \]

Rules of ↑TA:

\[
\begin{align*}
    a(q_s, q_x) & \rightarrow q_s \\
    a(q_x, q_s) & \rightarrow q_s \\
    b(q_s, q_x) & \rightarrow q_s \\
    b(q_x, q_s) & \rightarrow q_s \\
    a(q_B, q_x) & \rightarrow q_s \\
    a(q_x, q_B) & \rightarrow q_s \\
    b(q_x, q_x) & \rightarrow q_B \\
    a(q_x, q_x) & \rightarrow q_x \\
    b(q_x, q_x) & \rightarrow q_x \\
    c & \rightarrow q_X
\end{align*}
\]
Determinization

Example run

\[ F = \{ q_s \} \]

Rules of \( \uparrow \)TA:

\[
\begin{aligned}
a(q_s, q_x) &\rightarrow q_s \\
a(q_x, q_s) &\rightarrow q_s \\
b(q_s, q_x) &\rightarrow q_s \\
b(q_x, q_s) &\rightarrow q_s \\
a(q_B, q_x) &\rightarrow q_s \\
a(q_x, q_B) &\rightarrow q_s \\
b(q_x, q_x) &\rightarrow q_B \\
a(q_x, q_x) &\rightarrow q_X \\
b(q_X, q_X) &\rightarrow q_X \\
c &\rightarrow q_X 
\end{aligned}
\]
Determinization

Example run

\[ F = \{ q_s \} \]

Rules of \( \uparrow \text{TA} \):

\[
\begin{align*}
    a(q_s, q_x) &\rightarrow q_s \\
    a(q_x, q_s) &\rightarrow q_s \\
    b(q_s, q_x) &\rightarrow q_s \\
    b(q_x, q_s) &\rightarrow q_s \\
    a(q_B, q_x) &\rightarrow q_s \\
    a(q_x, q_B) &\rightarrow q_s \\
    b(q_x, q_x) &\rightarrow q_B \\
    a(q_x, q_x) &\rightarrow q_B \\
    b(q_x, q_x) &\rightarrow q_B \\
    c &\rightarrow q_B
\end{align*}
\]
Determinization

Example run

\[ F = \{ q_s \} \]

Rules of ↑TA:

\[
\begin{align*}
    a(q_s, q_x) & \rightarrow q_s \\
    a(q_x, q_s) & \rightarrow q_s \\
    b(q_s, q_x) & \rightarrow q_s \\
    b(q_x, q_s) & \rightarrow q_s \\
    a(q_B, q_x) & \rightarrow q_s \\
    a(q_x, q_B) & \rightarrow q_s \\
    b(q_x, q_x) & \rightarrow q_B \\
    a(q_x, q_x) & \rightarrow q_B \\
    b(q_x, q_x) & \rightarrow q_B \\
    c & \rightarrow q_X
\end{align*}
\]
Determinization

Example run

\[ F = \{qs\} \]

\[ \{qs, qx\} \\setminus F \neq \emptyset \text{ so this tree is accepted by } A_d \]

Rules of ↑TA:

\[
\begin{align*}
    a(qs, qx) & \rightarrow qs \\
    a(qx, qs) & \rightarrow qs \\
    b(qs, qx) & \rightarrow qs \\
    b(qx, qs) & \rightarrow qs \\
    a(qB, qx) & \rightarrow qs \\
    a(qx, qB) & \rightarrow qs \\
    b(qx, qx) & \rightarrow qB \\
    a(qx, qx) & \rightarrow qx \\
    b(qx, qx) & \rightarrow qx \\
    c & \rightarrow qx
\end{align*}
\]
TA classes

d\uparrowTA = \uparrowTA
TA classes

\[ d^{\uparrow\text{TA}} = \uparrow\text{TA} = \downarrow\text{TA} \]
TA classes

\[d \downarrow_{\text{TA}} \subsetneq d \uparrow_{\text{TA}} = \uparrow_{\text{TA}} = \downarrow_{\text{TA}}\]
TA classes

\[
d\downarrow\text{TA} \subseteq d\uparrow\text{TA} = \uparrow\text{TA} = \downarrow\text{TA}
\]

**Definition**

A ranked tree language \( L \subseteq T_{\Sigma_r} \) is **recognizable** if there is a \( \uparrow\text{TA} \) recognizing \( L \).
Closure Properties
of recognizable tree languages

If $L_1$ and $L_2$ are recognizable tree languages, then:

<table>
<thead>
<tr>
<th>$L_1$ is recognizable</th>
</tr>
</thead>
<tbody>
<tr>
<td>if $A = (Q, F, \Delta, \Sigma_r)$ is a complete ↑TA, then $A' = (Q, Q \setminus F, \Delta, \Sigma_r)$ recognizes $L(A)$. Completing is easy, by adding a sink state.</td>
</tr>
</tbody>
</table>

| $L_1 \cup L_2$ 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:

$L_1$ is recognizable

$L_1 \cup L_2$ is recognizable

Let

$$
\begin{align*}
A_1 &= (Q_1, F_1, \Delta_1, \Sigma_r) \\
A_2 &= (Q_2, F_2, \Delta_2, \Sigma_r)
\end{align*}
$$

be a complete $\uparrow$TA recognizing $L_1$

be a complete $\uparrow$TA recognizing $L_2$

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

$$
\begin{align*}
a(q_1, \ldots, q_n) &\rightarrow q \in \Delta_1 \\
a(q_1', \ldots, q'_n) &\rightarrow q' \in \Delta_2 \\
a((q_1, q_1'), \ldots, (q_n, q'_n)) &\rightarrow (q, q') \in \Delta'
\end{align*}
$$

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

$L_1 \cap L_2$ is recognizable
Closure Properties
of recognizable tree languages

If $L_1$ and $L_2$ are recognizable tree languages, then:

$L_1$ is recognizable

$L_1 \cup L_2$ is recognizable

$L_1 \cap L_2$ is recognizable

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

**Context** = tree over $\Sigma_r \uplus \{\square\}$ where

\[
\begin{align*}
\text{ar}(\square) &= 0 \text{ and} \\
\square &\text{ appears exactly once.}
\end{align*}
\]

Intuitively, *context* = *tree with a hole*.

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

Context = tree over $\Sigma_r \uplus \{\Box\}$ where

\[
\begin{cases}
\text{ar}(\Box) = 0 \quad \text{and} \\
\Box \text{ appears exactly once.}
\end{cases}
\]

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

\[
\begin{cases}
t = C_1[C_2[t']] \\
\forall n \geq 0. \quad C_1[C_2^n[t']] \in L
\end{cases}
\]

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

Tree homomorphism = \{ \text{transformation } h : \mathcal{T}_\Sigma \rightarrow \mathcal{T}_\Sigma', \text{ based on a mapping } m : \Sigma \rightarrow \mathcal{T}_{\Sigma'} \cup \mathcal{X}_{\text{ar}(m)} \}

Theorem

$L$: recognizable tree language in $\mathcal{T}_\Sigma$ \hspace{1cm} $h$: linear tree homom. $h : \mathcal{T}_\Sigma \rightarrow \mathcal{T}_\Sigma'$ \hspace{1cm} $\Rightarrow$ $h(L)$ 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

Tree homomorphism = \{ \text{transformation } h : \mathcal{T}_\Sigma \to \mathcal{T}_{\Sigma'}, \\
\text{based on a mapping } m : \Sigma \to \mathcal{T}_{\Sigma'} \cup X_{\text{ar}(m)} \}

**Theorem**

\( L : \text{recognizable tree language in } \mathcal{T}_\Sigma \) \Rightarrow \( h(L) \) 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**

\( L : \text{recognizable tree language in } \mathcal{T}_{\Sigma'} \) \Rightarrow \( h^{-1}(L) \) is recognizable (over \( \mathcal{T}_\Sigma \))

\[ \text{Olivier Gauwin (UMons)} \]

\[ \text{Feb/March 2010} \]
Minimization
of Tree Automata

Let \( \equiv \) be an equivalence relation. It is:

- a **congruence** if \( \forall a \in \mathcal{T}_\Sigma, \)
  
  \[
  \text{if } t_i \equiv t'_i \text{ for all } 1 \leq i \leq n \text{ then } a(t_1, \ldots, t_n) \equiv a(t'_1, \ldots, t'_n)\]

- 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 $\forall a \in T_{\Sigma_r}$,

  \[ \text{if } t_i \equiv t_i' \text{ for all } 1 \leq i \leq n \text{ then } a(t_1, \ldots, t_n) \equiv a(t_1', \ldots, t_n') \]

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

★★★★

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

\[ t \equiv_L t' \text{ if for all contexts } C \text{ over } \Sigma_r, C[t] \in L \iff C[t'] \in L \]
Minimization of Tree Automata

**Def:** \( t \equiv_L t' \) if \( \forall C. C[t] \in L \iff C[t'] \in L \)

**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 \( L = \bigcup_{t \mid \Delta(t) \in F_A} \text{class} \equiv_A (t) \).
Minimization
of Tree Automata

Def: \( t \equiv_L t' \) if \( \forall C. C[t] \in L \iff C[t'] \in L \)

Myhill-Nerode Theorem

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

\((\Rightarrow)\) Let \( A \) be a complete \( d\uparrow\text{TA} \) 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 \( L = \bigcup_{t \mid \Delta(t) \in F_A} \text{class}_{\equiv_A}(t) \).
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 \( \equiv_A \), \( C[t] \in L \iff C[t'] \in L \), and thus \( t \equiv_L t' \).
Minimization of Tree Automata

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

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:
  
  \[ t \equiv_A t' \text{ iff } \Delta(t) = \Delta(t') \text{ (state of } A) \]

  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 $\equiv_A$, $C[t] \in L \iff C[t'] \in L$, and thus $t \equiv_L t'$.

- $(\Leftarrow)$ $A_{\min} = (Q_{\min}, F_{\min}, \Delta_{\min}, \Sigma_r)$ with:
  
  - $Q_{\min} = class_{\equiv_L}$
  - $F_{\min} = \{class_{\equiv_L}(t) \mid t \in L\}$
  - $\Delta_{\min}$ contains rules:
    \[ f(class_{\equiv_L}(t_1), \ldots, class_{\equiv_L}(t_n)) \rightarrow class_{\equiv_L}(f(t_1, \ldots, t_n)) \]
Minimization of Tree Automata

Consequence of Myhill-Nerode

The minimum $d^{\uparrow}TA$ recognizing $L$ is unique, up to a renaming of states.

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

Minimization algorithm (sketch)

- input: complete and reduced $d^{\uparrow}TA$ $A$
- start from $\equiv_A$ and build $\equiv_L$,
- by merging $q$ and $q'$ if $\forall a \in \Sigma, \forall i, \forall q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_n \in Q_A$, $\Delta(a(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n)) \equiv \Delta(a(q_1, \ldots, q_{i-1}, q', q_{i+1}, \ldots, q_n))$
- until fixed point
## Complexity of some decision problems

<table>
<thead>
<tr>
<th>Name</th>
<th>Input</th>
<th>Output</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Membership</td>
<td>$\uparrow$TA $A$, tree $t$</td>
<td>$t \in L(A)$?</td>
<td>PTIME</td>
</tr>
<tr>
<td>Emptiness</td>
<td>$\uparrow$TA $A$</td>
<td>$L(A) = \emptyset$?</td>
<td>PTIME</td>
</tr>
<tr>
<td>Intersection</td>
<td>set $S$ of $\uparrow$TAs</td>
<td>$\bigcap_{A \in S} L(A) = \emptyset$?</td>
<td>EXPTIME-compl.</td>
</tr>
<tr>
<td>non-emptiness</td>
<td>set $S$ of $d\uparrow$TAs</td>
<td>$\bigcap_{A \in S} L(A) = \emptyset$?</td>
<td>EXPTIME-compl.</td>
</tr>
<tr>
<td>Universality</td>
<td>$\uparrow$TA $A$</td>
<td>$L(A) = T_{\Sigma_r}$?</td>
<td>EXPTIME-compl.</td>
</tr>
<tr>
<td></td>
<td>$d\uparrow$TA $A$</td>
<td>$L(A) = T_{\Sigma_r}$?</td>
<td>PTIME</td>
</tr>
<tr>
<td>Equivalence</td>
<td>$\uparrow$TAs $A_1$, $A_2$</td>
<td>$L(A_1) = L(A_2)$?</td>
<td>EXPTIME-compl.</td>
</tr>
<tr>
<td></td>
<td>$d\uparrow$TAs $A_1$, $A_2$</td>
<td>$L(A_1) = L(A_2)$?</td>
<td>PTIME</td>
</tr>
</tbody>
</table>
1 Ranked Trees
- Trees on Ranked Alphabet
- Tree Automata
- Tree Grammars
- Logic

2 Unranked Trees
- Unranked Trees
- Automata
- Logic
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 $\alpha \rightarrow \beta$ with $\alpha, \beta \in T_{N \cup F \cup \mathcal{X}}$ and $\alpha$ contains at least one non-terminal.

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

$a(b(x, d(N_1(c)))) \rightarrow a(d(x), b(N_2, c))$
Regular Tree Grammars

Definition

Regular tree grammars

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

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

Example

$$R = \begin{cases} S \rightarrow a(S, X) & S \rightarrow b(S, X) & S \rightarrow a(B, X) & B \rightarrow b(X, X) & X \rightarrow b(X, X) \\ S \rightarrow a(X, S) & S \rightarrow b(X, S) & S \rightarrow a(X, B) & X \rightarrow a(X, X) & X \rightarrow c \end{cases}$$

$a(b(c, c), c) \in L(G)$ because:

$$S \rightarrow_G a(B, X) \rightarrow_G a(b(X, X), X) \rightarrow_G a(b(c, X), X) \rightarrow_G a(b(c, c), X) \rightarrow_G a(b(c, c), c)$$

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

Expressiveness

<table>
<thead>
<tr>
<th>Proposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular tree languages $\equiv$ recognizable tree languages</td>
</tr>
</tbody>
</table>

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

Regular tree languages = recognizable tree languages

Normalized regular tree grammars are ↓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

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

$\textbf{MSO}[\Omega]$ : Syntax

$$\phi ::= \text{lab}_a(x) \mid \text{ch}_i(x, y) \mid \phi \land \phi \mid \neg \phi \mid \exists x. \phi \mid \exists X. \phi \mid x \in X$$

where $a \in \Sigma$, $\exists a \in \Sigma_r. \text{ar}(a) \geq i$, and $x, x_1, \ldots, x_k, X \in \mathcal{X}$. 
Monadic Second-Order (MSO) Logic

Syntax

- $\Sigma_r$ a ranked signature
- $\mathcal{X}$ contains first-order ($x, y...$) and second-order ($X, Y...$) variables.
- $\Omega = \{\text{lab}_a | a \in \Sigma_r\} \cup \{\text{ch}_i | \exists a \in \Sigma_r. \text{ar}(a) \geq i\}$
- predicates $\text{lab}_a$ are unary, while $\text{ch}_i$ are binary

$\text{MSO[}\Omega\text{]}$: Syntax

$$\phi ::= \text{lab}_a(x) \mid \text{ch}_i(x, y) \mid \phi \land \phi \mid \neg \phi \mid \exists x. \phi \mid \exists X. \phi \mid x \in X$$

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

$\text{MSO[}\Omega\text{]}$ is also known as \text{WSkS}, the \textit{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

\[
even_a = \exists X \ \forall x \ \lab_a(x) \Rightarrow \begin{cases} 
\exists y_1 \ ch_1(x, y_1) \land \\
\exists y_2 \ ch_2(x, y_2) \land \\
x \in X \iff (y_1 \in X \oplus y_2 \in X) \\
\end{cases}
\]

\[
\land_{\alpha \neq a} \ \lab_{\alpha}(x) \Rightarrow \begin{cases} 
\land_{1 \leq i \leq \ar(\alpha)} \exists y_i \ ch_i(x, y_i) \land \\
x \not\in X \iff (y_1 \not\in X \oplus \ldots \oplus y_{\ar(\alpha)} \not\in X) \\
\end{cases}
\]

\[
\land \exists x \ \root(x) \land x \in X
\]

where:

\[
\phi \oplus \phi' = (\phi \land \neg \phi') \lor (\neg \phi \land \phi')
\]

\[
\root(x) = \neg \exists y. \ ch_1(y, x)
\]
Monadic Second-Order (MSO) Logic

Semantics

convention: \( \phi(\overline{x}, \overline{X}) \) means that \( \phi \) 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 \( \mu : \overline{x} \cup \overline{X} \rightarrow \text{nodes}(t) \).

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

- \( t, \mu \models \text{lab}_a(x) \) iff \( \text{lab}^t_a(\mu(x)) \)
- \( t, \mu \models \text{ch}_i(x, y) \) iff \( \text{ch}_i(\mu(x), \mu(y)) \)
- \( t, \mu \models \phi \land \phi' \) iff \( t, \mu \models \phi \) and \( t, \mu \models \phi' \)
- \( t, \mu \models \neg \phi \) iff \( t, \mu \not\models \phi \)
- \( t, \mu \models \exists x \phi \) iff there exists \( \pi \in \text{nodes}(t) \) s.t. \( t, \mu[x \leftarrow \pi] \models \phi \)
- \( t, \mu \models \exists X \phi \) iff there exists \( S \subseteq \text{nodes}(t) \) s.t. \( t, \mu[X \leftarrow S] \models \phi \)
- \( t, \mu \models x \in X \) iff \( \mu(x) \in \mu(X) \)
MSO and Tree Languages

Expressiveness of MSO vs $\uparrow$TA?

For closed formulas $\phi$, define $L_{\phi} = \{ t \mid t \models \phi \} \subseteq T_{\Sigma_r}$.

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

$\phi(\overline{x}, \overline{X})$ \text{ semantics} \rightarrow \{(t, \mu) \mid t, \mu \models \phi\}$ \text{ tree language} \rightarrow L_{\phi} = \{ t \ast \mu \mid t, \mu \models \phi\}$
For closed formulas $\phi$, define $L_\phi = \{ t \mid t \models \phi \} \subseteq T_{\Sigma_r}$.

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

\[
\phi(\overline{x}, \overline{X}) \xrightarrow{\text{semantics}} \{(t, \mu) \mid t, \mu \models \phi\} \xrightarrow{\text{tree language}} L_\phi = \{ t \ast \mu \mid t, \mu \models \phi\}
\]

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

$t \ast \mu \in T_{\Sigma_r} \times \mathbb{B}^n$
MSO and Tree Languages

Expressiveness of MSO vs $\uparrow$TA?

For closed formulas $\phi$, define $L_\phi = \{ t \mid t \models \phi \} \subseteq T_{\Sigma_r}$.

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

$\phi(\overline{x}, \overline{X}) \xrightarrow{\text{semantics}} \{(t, \mu) \mid t, \mu \models \phi\}$ $\xrightarrow{\text{tree language}} L_\phi = \{ t \cdot \mu \mid t, \mu \models \phi\}$

$t \cdot \mu \in T_{\Sigma_r \times \mathbb{B}^n}$

\[ t = \begin{array}{c}
\text{a} \\
\text{c} \\
\text{c} \\
\text{c} \\
\text{c} \\
\end{array} \quad \xrightarrow{\mu} \quad \begin{array}{c}
\text{b} \\
\text{b} \\
\text{y} \\
\text{x} \\
\text{X} \\
\end{array} \]

\[ t \cdot \mu = \begin{array}{c}
\text{a} \quad \text{b} \\
\text{c} \quad \text{c} \quad \text{c} \quad \text{c} \\
\text{b} \quad \text{y} \quad \text{x} \quad \text{X} \\
\text{c} \quad \text{c} \quad \text{c} \quad \text{c} \\
\text{c} \quad \text{c} \quad \text{c} \quad \text{c} \\
\end{array} \]

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

Olivier Gauwin (UMons)  Finite Tree Automata  Feb/March 2010  31 / 66
Recognizable languages $=$ MSO-definable languages

**Theorem [TW68, Don70]**

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

**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 $\{q_1, \ldots, q_n\} = Q$.

$\phi_A = \exists X_{q_1} \ldots \exists X_{q_n} \text{partition}(X_{q_1}, \ldots, X_{q_n}) \land$

$\land_{a \rightarrow q \in \Delta} \text{leaf}(x) \land \text{lab}_a(x) \Rightarrow x \in X_q$

$\land_{a(q_{i_1}, \ldots, q_{i_k}) \rightarrow q_i \in \Delta} \forall x \forall y_1 \ldots \forall y_k \left\{ \begin{array}{c}
\text{lab}_a(x) \land \\
\text{ch}_1(x, y_1) \land \ldots \land \text{ch}(x, y_k) \land \\
y_1 \in X_{q_{i_1}} \land \ldots \land y_k \in X_{q_{i_k}}
\end{array} \right\} \Rightarrow x \in X_{q_i}$

$\land \exists x. \text{root}(x) \land \bigvee_{q \in F} x \in X_q$

with:

$\left\{ \begin{array}{c}
\text{leaf}(x) = \not\exists y. \text{ch}_1(x, y) \\
\text{partition}(X_{q_1}, \ldots, X_{q_n}) = \ldots
\end{array} \right\}$
(⇐) idea: equivalence betw. logical connectives and automata operators.

\[ \mathcal{L}_{\text{lab}_a} = \left\{ \ldots, \frac{(a,1)}{(a,0)}, \frac{(a,0)}{(b,0)}, \frac{(a,1)}{(b,0)}, \ldots \right\} \text{ recognizable} \]
(⇐) idea: equivalence betw. logical connectives and automata operators.

\[ L_{\text{lab}_a} = \left\{ \ldots, \frac{\ldots, (a,1)}{(a,0)} \frac{(a,0)}{(b,0)}, \frac{(a,1)}{(b,0)}, \frac{\ldots}{(a,0,1)}, \frac{(a,0,1)}{(b,0,0)}, \ldots \right\} \text{recognizable} \]

\[ L_{ch_1} = \left\{ \ldots, \frac{\ldots, (a,1,0)}{(a,0,1)} \frac{(a,0,1)}{(b,0,0)}, \frac{\ldots}{(a,0,1)}, \frac{(a,0,1)}{(b,0,0)}, \ldots \right\} \text{recognizable (ch}_i\text{ also)} \]
(⇐) idea: equivalence betw. *logical connectives* and *automata operators*.

- \( L_{\text{lab}_{a}} = \{ \ldots, \ \begin{array}{c} (a,1) \\ (a,0) \end{array}, \ \begin{array}{c} (b,0) \\ (a,1) \end{array}, \ldots \} \) recognizable

- \( L_{\text{ch}_{1}} = \{ \ldots, \ \begin{array}{c} (a,1,0) \\ (a,0,1) \end{array}, \ldots \} \) recognizable (*ch* also)

- \( L_{\phi \land \phi'} = L_{\phi} \cap L_{\phi'} \) recognizable (product construction)
idea: equivalence betw. logical connectives and automata operators.

\[ L_{\text{lab}_a} = \{ (a,1), (a,0), \ldots, (a,0) \ , \ (b,0) \ , \ (a,1) \ , \ (b,0) \ , \ \ldots \} \text{ recognizable} \]

\[ L_{\text{ch}_1} = \{ (a,1,0), \ldots, (a,0,1) \ , \ (b,0,0) \ , \ \ldots \} \text{ recognizable (ch}_i \text{ also)} \]

\[ L_{\phi \land \phi'} = L_{\phi} \cap L_{\phi'} \text{ recognizable (product construction)} \]

\[ L_{\neg \phi} = L_{\text{valid}} \setminus L_{\phi} \text{ where } L_{\text{valid}} = \{ t \ast \mu \mid t \in T_{\Sigma_r}, \mu \text{ an assignment of free vars of } \phi \} \]

\[ L_{\text{valid}} \text{ is recognizable so } L_{\neg \phi} \text{ is recognizable} \]
idea: equivalence betw. logical connectives and automata operators.

- \( L_{\text{lab}} = \{ \ldots, (a,1), (a,0), \ldots \} \) recognizable
- \( L_{\text{ch}_1} = \{ \ldots, (a,1,0), (a,0,1), \ldots \} \) recognizable (\( ch_i \) also)

\( L_{\phi \land \phi'} = L_\phi \cap L_{\phi'} \) recognizable (product construction)

\( L_{\neg \phi} = L_{\text{valid}} \setminus L_\phi \) where \( L_{\text{valid}} = \{ t \ast \mu \mid t \in T_{\Sigma_r}, \mu \text{ an assignment of free vars of } \phi \} \)

\( L_{\text{valid}} \) is recognizable so \( L_{\neg \phi} \) is recognizable

\( L_{\exists x \phi} \) is obtained from \( L_\phi \) by removing the \( x \)-component:

\[
\begin{align*}
(a,1,0) & \\
/ & \setminus & \in L_{\text{ch}_1(x,y)} & \rightarrow & / & \setminus & \in L_{\exists x \ ch_1(x,y)} \\
(a,0,1) & & (b,0,0) & & (a,1) & & (b,0)
\end{align*}
\]

recognizable (remove the component in rules)
(⇐) idea: equivalence betw. logical connectives and automata operators.

\[ L_{\text{lab}_a} = \{ (a,1), (a,0), \ldots, (a,0), (b,0), (a,1), (b,0) \} \text{ recognizable} \]

\[ L_{ch_1} = \{ (a,1,0), \ldots, (a,0,1), (b,0,0) \} \text{ recognizable (ch}_i \text{ also)} \]

\[ L_{\phi \land \phi'} = L_{\phi} \cap L_{\phi'} \text{ recognizable (product construction)} \]

\[ L_{\neg \phi} = L_{\text{valid}} \setminus L_{\phi} \text{ where } L_{\text{valid}} = \{ t \star \mu \mid t \in T_{\Sigma_r}, \mu \text{ an assignment of free vars of } \phi \} \]

\[ L_{\text{valid}} \text{ is recognizable so } L_{\neg \phi} \text{ is recognizable} \]

\[ L_{\exists x \phi} \text{ is obtained from } L_{\phi} \text{ by removing the } x \text{-component:} \]

\[ (a,1,0) \quad \quad \quad \quad \quad (a,0) \]
\[ / \quad \setminus \quad \in L_{ch_1(x,y)} \rightarrow \quad / \quad \setminus \quad \in L_{\exists x \ ch_1(x,y)} \]
\[ (a,0,1) \quad (b,0,0) \quad \quad \quad \quad \quad (a,1) \quad (b,0) \]

recognizable (remove the component in rules)

\[ L_{\exists x \phi} : \text{ same idea} \]
(⇐) idea: equivalence betw. logical connectives and automata operators.

- \( \mathcal{L}_{\text{lab}_a} = \left\{ \cdots, \frac{(a,1)}{(a,0)}, \frac{(a,0)}{(b,0)}, \frac{(a,1)}{(b,0)} \right\} \) recognizable
- \( \mathcal{L}_{\text{ch}_1} = \left\{ \cdots, \frac{(a,1,0)}{(a,0,1)}, \frac{(a,0,1)}{(b,0,0)} \right\} \) recognizable (\( \text{ch}_i \) also)

\( \mathcal{L}_{\phi \land \phi'} = \mathcal{L}_\phi \cap \mathcal{L}_{\phi'} \) recognizable (product construction)

\( \mathcal{L}_{\neg \phi} = \mathcal{L}_{\text{valid}} \setminus \mathcal{L}_\phi \) where \( \mathcal{L}_{\text{valid}} = \{ t * \mu \mid t \in T_{\Sigma_r}, \mu \, \text{an assignment of free vars of } \phi \} \)

\( \mathcal{L}_{\text{valid}} \) 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:

\[
\begin{array}{c}
(a,1,0) \\
\downarrow \\
(a,0,1)
\end{array} \quad \begin{array}{c}
(a,0) \\
\downarrow \\
(b,0,0)
\end{array} \in \mathcal{L}_{\text{ch}_1(x,y)} \quad \rightarrow \quad \begin{array}{c}
(a,1) \\
\downarrow \\
(b,0)
\end{array} \in \mathcal{L}_{\exists x \, \text{ch}_1(x,y)}
\]

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)

\[
\begin{array}{c}
(a,1,1) \\
\downarrow \\
(a,0,1)
\end{array} \quad \begin{array}{c}
(a,0,1) \\
\downarrow \\
(b,0,0)
\end{array} \in \mathcal{L}_{x \in X} \quad \text{but} \quad \begin{array}{c}
(a,1,0) \\
\downarrow \\
(b,0,1)
\end{array} \notin \mathcal{L}_{x \in X}
\]
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
An unranked tree over $\Sigma$ is a tree, labeled by elements of $\Sigma$, without arity constraints.

We write $T_{\Sigma}$ for the set of unranked trees over $\Sigma$. 

![Diagram of an unranked tree](image)
1 Ranked Trees
   - Trees on Ranked Alphabet
   - Tree Automata
   - Tree Grammars
   - Logic

2 Unranked Trees
   - Unranked Trees
   - Automata
   - Logic
Which notion of automata can we use?

For ranked trees, rules were $a(q_1, \ldots, q_n) \rightarrow q$ 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 $\Sigma$ 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
- $\Delta$ is a set of rules $a(L) \rightarrow q$ 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 $\rho : \text{nodes}(t) \rightarrow Q$ such that for all nodes $\pi$ of $t$ with children $\pi_1, \ldots, \pi_n$ and label $a$, there is a rule $a(L) \rightarrow \rho(\pi) \in \Delta$ 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

\[
\begin{align*}
html & \rightarrow \ head \cdot body \\
head & \rightarrow \ title\cdot meta?.\ style?\cdot script?\cdot ...
body & \rightarrow (p\mid div\mid table\mid h1\mid\cdot\cdot\cdot)*
\end{align*}
\]
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]**

\[ fcn : T_\Sigma \rightarrow T_{\Sigma_f} \text{ where } \Sigma_f = (\Sigma \cup \{\bot\}, \text{ar}) \text{ with } \text{ar}(a)=2 \text{ for } a \in \Sigma \text{ and } \text{ar}(\bot)=0 \]

\[
\begin{array}{c}
\text{Tree 1} \\
\begin{array}{c}
\text{Tree 2}
\end{array}
\end{array}
\]
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]

\[ fcns : T_\Sigma \to T_{\Sigma_f} \text{ where } \Sigma_f = (\Sigma \cup \{\bot\}, \text{ar}) \text{ with } \text{ar}(a)=2 \text{ for } a \in \Sigma \text{ and } \text{ar}(\bot)=0 \]

**Curryfication**

\[ curry : T_\Sigma \to T_{\Sigma_c} \text{ where } \Sigma_c = (\Sigma \cup \{@\}, \text{ar}) \text{ with } \text{ar}(a)=0 \text{ for } a \in \Sigma \text{ and } \text{ar}(\@)=2 \]

\[ t_1@t_2 = \text{“add } t_2 \text{ as last child of the root of } t_1\text{”} \]
We can use tree automata over ranked languages on encoded trees:

<table>
<thead>
<tr>
<th></th>
<th>$f CNS$</th>
<th>$Curry$</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑TA</td>
<td>$C_1$</td>
<td>$C_2$</td>
</tr>
<tr>
<td>↓TA</td>
<td>$C_3$</td>
<td>$C_4$</td>
</tr>
</tbody>
</table>

These 4 classes are equally expressive. $C_2$ corresponds to stepwise tree automata [CNT04].
We can use tree automata over ranked languages on encoded trees:

<table>
<thead>
<tr>
<th></th>
<th>$fcns$</th>
<th>$curry$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\uparrow$TA</td>
<td>$C_1$</td>
<td>$C_2$</td>
</tr>
<tr>
<td>$\downarrow$TA</td>
<td>$C_3$</td>
<td>$C_4$</td>
</tr>
</tbody>
</table>

These 4 classes are equally expressive.

$C_2$ corresponds to stepwise tree automata [CNT04].

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

Let $\Sigma = \{ \bar{a} \mid a \in \Sigma \}$. 

$n(t)$, the linearization of $t$, is the word over $\Sigma \cup \bar{\Sigma}$ produced by the pre-order traversal of $t$:

This corresponds to the XML serialization: `<a><a><b></b>...<b></b></a>`
Visibly Pushdown Automata [AM04]

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

---

**Visibly Pushdown Automaton (VPA)**

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

- $q_1, a \rightarrow \gamma, q_2$
- $q_1, \bar{a}, \gamma \rightarrow q_2$

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

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 $\pi$: update the current state, and assign $\gamma \in \Gamma$ to $\pi$
  - when closing node $\pi$: update the current state according to $\gamma$

A: VPA on $T_\Sigma$ with $\Sigma = \{a, b\}$
and $Q = \{0, 1, 2, 3, 4, 5\}$
and $\Gamma = \{\alpha, \beta, \gamma\}$
a run of a VPA on a tree consists in:

- when opening node $\pi$: update the current state, and assign $\gamma \in \Gamma$ to $\pi$
- when closing node $\pi$: update the current state according to $\gamma$

\[ (0, \emptyset) \]

$A$: VPA on $T_\Sigma$ 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 $\pi$: update the current state, and assign $\gamma \in \Gamma$ to $\pi$
  - when closing node $\pi$: update the current state according to $\gamma$

![Diagram](image)

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

$(0, \emptyset) \overset{a}{\rightarrow} (1, \alpha)$
VPAs as Tree Automata

a run of a VPA on a tree consists in:

- when opening node $\pi$: update the current state, and assign $\gamma \in \Gamma$ to $\pi$
- when closing node $\pi$: update the current state according to $\gamma$

\[
\begin{align*}
(0, \emptyset) & \xrightarrow{a} (1, \alpha) & \xrightarrow{a} (1, \alpha.\beta)
\end{align*}
\]

A: VPA on $T_{\Sigma}$ 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 $\pi$: update the current state, and assign $\gamma \in \Gamma$ to $\pi$
  - when closing node $\pi$: update the current state according to $\gamma$

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

$$(0, \emptyset) \xrightarrow{a} (1, \alpha) \xrightarrow{a} (1, \alpha.\beta) \xrightarrow{b} (4, \alpha.\beta.\beta)$$
VPAs as Tree Automata

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

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

$$(0, \emptyset) \xrightarrow{a} (1, \alpha) \xrightarrow{a} (1, \alpha.\beta) \xrightarrow{b} (4, \alpha.\beta.\beta)$$
$$(\cdot b) \xrightarrow{} (3, \alpha.\beta)$$
VPAs as Tree Automata

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

$$
\begin{align*}
(0, \emptyset) & \xrightarrow{a} (1, \alpha) \xrightarrow{a} (1, \alpha.\beta) \xrightarrow{b} (4, \alpha.\beta.\beta) \\
& \xrightarrow{b} (3, \alpha.\beta) \xrightarrow{a} (2, \alpha)
\end{align*}
$$

$A$: VPA on $T_\Sigma$ 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 $\pi$: update the current state, and assign $\gamma \in \Gamma$ to $\pi$
  - when closing node $\pi$: update the current state according to $\gamma$

\[\begin{align*}
(0, \emptyset) &\xrightarrow{a} (1, \alpha) &\xrightarrow{a} (1, \alpha.\beta) &\xrightarrow{b} (4, \alpha.\beta.\beta) \\
&\xrightarrow{b} (3, \alpha.\beta) &\xrightarrow{b} (2, \alpha) &\xrightarrow{b} (4, \alpha.\gamma)
\end{align*}\]

$A$: VPA on $T_\Sigma$ 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 $\pi$: update the current state, and assign $\gamma \in \Gamma$ to $\pi$
  - when closing node $\pi$: update the current state according to $\gamma$

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

\[
\begin{align*}
(0, \emptyset) & \xrightarrow{a} (1, \alpha) \xrightarrow{a} (1, \alpha.\beta) \xrightarrow{b} (4, \alpha.\beta.\beta) \\
& \xrightarrow{b} (3, \alpha.\beta) \xrightarrow{a} (2, \alpha) \xrightarrow{b} (4, \alpha.\gamma) \\
& \xrightarrow{b} (3, \alpha)
\end{align*}
\]
VPAs as Tree Automata

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

\[
A: \text{VPA on } T_{\Sigma} \text{ with } \Sigma = \{a, b\} \\
\text{and } Q = \{0, 1, 2, 3, 4, 5\} \\
\text{and } \Gamma = \{\alpha, \beta, \gamma\}
\]

\[
\begin{array}{c}
(0, \emptyset) \xrightarrow{a} (1, \alpha) \xrightarrow{a} (1, \alpha.\beta) \xrightarrow{b} (4, \alpha.\beta.\beta) \\
\xrightarrow{b} (3, \alpha.\beta) \xrightarrow{a} (2, \alpha) \xrightarrow{b} (4, \alpha.\gamma) \\
\xrightarrow{b} (3, \alpha) \xrightarrow{a} (5, \emptyset)
\end{array}
\]
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 TA, \downarrow TA\} + \{fcns, curry\}\n- 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: $\text{NFAs} \subsetneq \text{VPAs} = \text{dVPAs} \subsetneq \text{dPAs} \subsetneq \text{PAs}$
- determinization procedure in $O(2^{|Q|^2})$
- corresponds to streaming XML deterministically
  - yardstick class for streamability
VPAs: determinization

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

For an hedge $h$, let

$$\text{acc}_A(h) = \{(q, q') \in Q^2 \mid \text{there is a run of } A \text{ on } h \text{ from } q \text{ to } q'\}$$

The determinization procedure computes $\text{acc}_A(h)$ for all hedges $h$ of the tree $t$. More precisely, the current state at node $\pi$ is $\text{acc}(h)$ where $h$ is the hedge of left siblings of $\pi$:

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

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

\[
\begin{align*}
  a \in \Sigma & \quad P \in Q^{A'} \\
  P \cdot a: P & \rightarrow id_{Q^A} \in \Delta^{A'}
\end{align*}
\]

\[
\begin{align*}
  a \in \Sigma & \quad P, P' \subseteq Q^A \\
  P \cdot a: P' & \rightarrow P' \circ \text{Update}_P^a \in \Delta^{A'}
\end{align*}
\]

with

\[ \text{Update}_P^a = \{ (q, q') \mid \exists (q_1, q_2) \in P. \exists \gamma. q \xrightarrow{a: \gamma} q_1 \in \Delta^A \& q_2 \xrightarrow{a: \gamma} q' \in \Delta^A \} \]
Determinization: example

exercise: run of \( \text{det}(A) \) on \( t \)
Linearization
Place of recognizable languages

\[ \text{NFAs} \subset \text{VPAs} = \text{dVPAs} \subset \text{dPAs} \subset \text{PAs} = \text{CFLs} \]

<table>
<thead>
<tr>
<th></th>
<th>closed by</th>
<th>det.</th>
<th>decidable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>∩</td>
<td>∪</td>
<td>( \mathcal{L}(A) = \emptyset )</td>
</tr>
<tr>
<td>NFAs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>VPAs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dPAs</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>PAs</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
## Minimization of unranked tree automata

<table>
<thead>
<tr>
<th>det. hedge automata using DFAs for horiz. lang.</th>
<th>unique minimal automaton?</th>
<th>procedure cost</th>
<th>ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d \uparrow TA \circ curry$ = stepwise tree automata</td>
<td>✓</td>
<td>PTIME</td>
<td>[MN07]</td>
</tr>
<tr>
<td>$d \uparrow TA \circ fcns$</td>
<td>✓</td>
<td>PTIME</td>
<td>[MN07]</td>
</tr>
<tr>
<td>dVPAs</td>
<td>✗</td>
<td>open?</td>
<td>[AKMV05] [CW07]</td>
</tr>
</tbody>
</table>

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

For well-matched words $w$ and $w'$,

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

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
for unranked trees, use first-child/next-sibling predicates:

$$\Omega_u = \{ \text{lab}_a \mid a \in \Sigma \} \cup \{ \text{fc}, \text{ns} \}$$

### Equivalence with automata

A tree language $L \subseteq T_\Sigma$ 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 $\phi$, define $\phi'$ recognizing $\text{fcns}(\mathcal{L}_\phi)$. Then use equivalence for ranked trees.
XPath → det. automata: in $O(2^{2|e|})$

★★★★

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

#### Syntax

<table>
<thead>
<tr>
<th>Component</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis</td>
<td>$d ::= self \mid ch \mid ch^*$</td>
</tr>
<tr>
<td>steps</td>
<td>$S ::= d::a \mid d::*$ (where $a \in \Sigma$)</td>
</tr>
<tr>
<td>paths</td>
<td>$P ::= S \mid P[F] \mid P_1/P_2$</td>
</tr>
<tr>
<td>filters</td>
<td>$F ::= P \mid \neg F \mid F_1 \land F_2$</td>
</tr>
<tr>
<td>rooted paths</td>
<td>$R ::= /P$</td>
</tr>
</tbody>
</table>
**k-Downward XPath**

**Syntax**

- **axis**
  
  \[
d \ ::= \ self \mid ch \mid ch^*
  \]

- **steps**
  
  \[
  S \ ::= \ d::a \mid d::*
  \text{ (where } a \in \Sigma \text{)}
  \]

- **paths**
  
  \[
  P \ ::= \ S \mid P[F] \mid P_1/P_2
  \]

- **filters**
  
  \[
  F \ ::= \ P \mid \neg F \mid F_1 \land F_2
  \]

- **rooted paths**
  
  \[
  R \ ::= /P
  \]

**Semantic**

\[
\begin{align*}
\llbracket d::* \rrbracket_{path}(t) &= d^t \\
\llbracket d::a \rrbracket_{path}(t) &= \{ (\pi, \pi') \in d^t \mid \text{lab}^{t}_{a}(\pi') \} \\
\llbracket P_1/P_2 \rrbracket_{path}(t) &= \llbracket P_1 \rrbracket_{path}(t) \circ \llbracket P_2 \rrbracket_{path}(t) \\
\llbracket P[F] \rrbracket_{path}(t) &= \{ (\pi, \pi') \in \llbracket P \rrbracket_{path}(t) \mid \pi' \in \llbracket F \rrbracket_{filter}(t) \}
\end{align*}
\]
**Syntax**

axis \( d ::= \) \( \text{self} \mid ch \mid ch^* \)

steps \( S ::= d::a \mid d:::* \) (where \( a \in \Sigma \))

paths \( P ::= S \mid P[F] \mid P_1/P_2 \)

filters \( F ::= P \mid \neg F \mid F_1 \land F_2 \)

rooted paths \( R ::= /P \)

**Semantic**

\[
\begin{align*}
[d:::*]_{path}(t) &= d^t \\
[d::a]_{path}(t) &= \{(\pi, \pi') \in d^t \mid \text{lab}_a^t(\pi')\} \\
[P_1/P_2]_{path}(t) &= [P_1]_{path}(t) \circ [P_2]_{path}(t) \\
[P[F]]_{path}(t) &= \{(\pi, \pi') \in [P]_{path}(t) \mid \pi' \in [F]_{filter}(t)\} \\
[P]_{filter}(t) &= \{\pi \mid \exists \pi'. (\pi, \pi') \in [P]_{path}(t)\} \\
[\neg F]_{filter}(t) &= \text{nodes} \setminus [F]_{filter}(t) \\
[F_1 \land F_2]_{filter}(t) &= [F_1]_{filter}(t) \cap [F_2]_{filter}(t) \\
[/P]_{filter}(t) &= \{\pi \mid (\text{root}, \pi) \in [P]_{path}(t)\}
\end{align*}
\]
$k$-Downward XPath

Restrictions:

- $|\text{conjunctions + filters}| \leq k$
- 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 \(F = [ch[F']]\)

\[
\begin{align*}
a \in \Sigma, V \in \{0, 1\} \\
\text{start} & \xrightarrow{(a,V):0} 0 \\
\end{align*}
\]

opening the root: move to 0

\[
\begin{align*}
q_1 & \xrightarrow{(a,V):\gamma} q_2 \in \Delta^{A'} \\
q_1 & \in I^{A'} \\
b \in \{0, 1\} \\
\end{align*}
\]

opening a child: start testing \(F'\)

\[
\begin{align*}
q_1 & \xrightarrow{\alpha(a,V):\gamma} q_2 \in \Delta^{A'} \\
\end{align*}
\]

run test of \(F'\)
Example: \([ch::b]\)

\[
\begin{align*}
q_1 \xrightarrow{(a, V) : \gamma} q_2 & \in \Delta^{A'} & q_2 \notin F^{A'} & b \in \{0, 1\} \\
q_1' \xrightarrow{(a, V) : \gamma} q_2' & \in \Delta^{A'} & q_1' \in I^{A'} & b \in \{0, 1\}
\end{align*}
\]

failure of \(F'\): no new match

\[
q_1 \xrightarrow{(a, V) : b} b
\]

success of \(F'\): move to 1

\[
\begin{align*}
q_1 \xrightarrow{(a, V) : \gamma} q_2 & \in \Delta^{A'} & q_2 \in F^{A'} & b \in \{0, 1\} \\
q_1' \xrightarrow{(a, V) : \gamma} q_2' & \in \Delta^{A'} & q_1' \in I^{A'} & b \in \{0, 1\}
\end{align*}
\]

\[
q_1 \xrightarrow{(a, V) : b} 1
\]

\[
\begin{align*}
a & \in \Sigma \\
V & \in \{0, 1\} \\
b & \in \{0, 1\}
\end{align*}
\]

closing the root

\[
b \xrightarrow{(a, V) : 0} b
\]
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 $[ch^*:a[ch::b[x]]]$
- $A_{F_1 \land F_2} = A_{F_1} \land A_{F_2}$ (product construction)
- $A_{\neg F} = \neg A_F$
Other logics

- Conditional XPath, Regular XPath
- other modal logics: Tree TL, CTL\(^*\), PDL\(_{tree}\)
- \(\mu\)-calculus
Language, Automata and Logic for Finite Trees

Olivier Gauwin

UMons

Feb/March 2010
Rajeev Alur, Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan.

Congruences for visibly pushdown languages.


Rajeev Alur.

Marrying words and trees.


Rajeev Alur and P. Madhusudan.

Visibly pushdown languages.


Anne Brüggemann-Klein, Derick Wood, and Makoto Murata.
Regular tree and regular hedge languages over unranked alphabets: Version 1, April 07 2001.


Revised October, 12th 2007.


