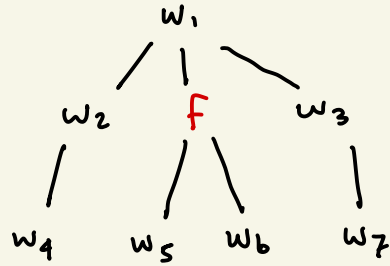


Tree automata

words

$w_1, w_2 \dots w_n$

trees



Bottom-up tree automaton $A = (Q, \Sigma, F, \delta)$

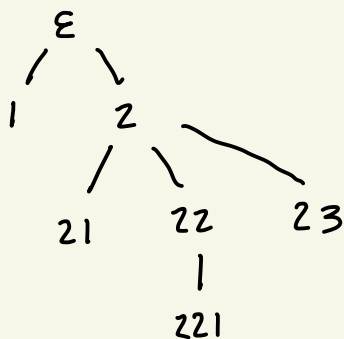
- states Q
- ranked alphabet $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \dots \cup \Sigma_k$
e.g. symbol f with $\text{arity}(f)=2$ and $f \in \Sigma_2$
- Accepting states $F \subseteq Q$
- Transition relations $\delta_i : \Sigma_i \times Q^i \rightarrow P(Q)$

How to represent the inputs to a tree automaton?

(Rooted, ordered, Dir-) tree: A structure T whose domain is a prefix-closed, nonempty subset of Dir^* , for some set $\text{Dir} \subseteq \mathbb{N}^{>0}$ of "directions"

Example

$\text{Dir} = \{1, 2, 3\}$



$\text{Succ}_T(x) := \{ u \in D(T) : u = xi \}$.

Σ -labeled tree: A tree T

together with a labeling $\lambda: D(T) \rightarrow \Sigma$.

If alphabet Σ is ranked, then call a

Σ -labeled tree T a Σ -term if T

respects the ranks of Σ in its labeling:

$\lambda(x) = a \in \Sigma$; iff $\text{Succ}(x) = \{xj : j \in [i]\}$.

We consider tree automata which process
 finite Σ -terms as input.

Fix a ranked alphabet Σ . For convenience,
 we specify sets of (finite) Σ -terms using grammars.

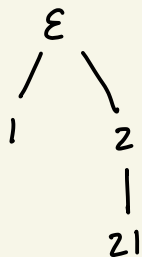
Use T_Σ for the terms corresponding to the language
 of following grammar:

$$t ::= a \in \Sigma_0 \mid s^k(t_1, \dots, t_k), \quad s^k \in \Sigma_k$$

Example

$$f(a, h(b)) \quad \text{is} \quad T = (D(T), \lambda)$$

$D(T)$:



$$\lambda(\varepsilon) = f$$

$$\lambda(1) = a$$

$$\lambda(2) = h$$

$$\lambda(21) = b$$

Tree automata $A = (Q, \Sigma, F, \delta)$ process an input term $t \in T_\Sigma$ by looking at the leaves of t and recursively labeling the internal nodes with states Q . They must follow the rules encoded by δ . If it's possible to label t 's root with some $q \in F \subseteq Q$, then t is accepted by the automaton, otherwise it's rejected. Call the labelings "runs".

Run: A run of A on term $t = (D(t), \lambda)$ is a Q -labeled tree $r_t = (D(r_t), \lambda_r)$ satisfying:

- 1) $D(r_t) = D(t)$
- 2) $\forall u \in D(r_t)$, let $s = \lambda(u)$, and $K = \text{arity}(s)$
 if $(K=0)$ $\lambda_r(u) \in \delta_0(s)$
 else $(K>0)$ $\lambda_r(u) \in \delta_K(s, \lambda_r(u_1), \dots, \lambda_r(u_K))$

An accepting run is one for which $\lambda_r(\varepsilon) \in F$.

Language $L(A) := \{t \in T_\Sigma : A \text{ has an accepting run on } t\}$.

Example Let $\Sigma_0 = \{0, 1, \dots, 50\}$, $\Sigma_2 = \{+, *, -\}$.

$A = (Q, \Sigma, F = \{42\}, \delta)$, where

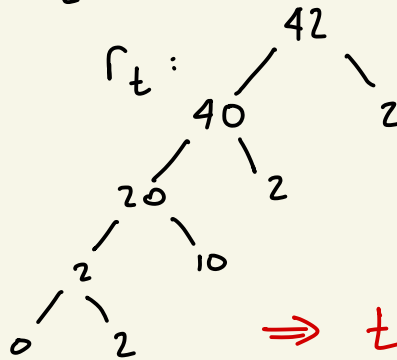
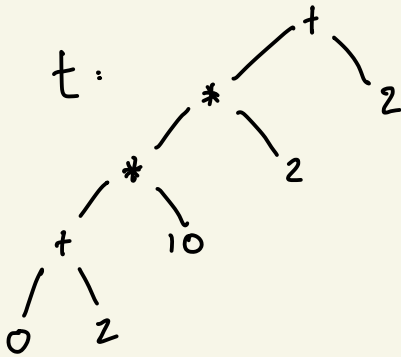
$$Q = \{0, 1, 2, \dots, 1000\} \sqcup \{\perp\}$$

$$\delta_0(i) = \{i\}$$

$$\delta_2(+, i, j) = \begin{cases} \{i+j\} & \text{if } i+j \leq 1000 \wedge i, j \neq \perp \\ \{\perp\} & \text{otherwise} \end{cases}$$

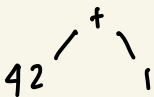
$$\delta_2(*, i, j) = \begin{cases} \{i*j\} & \text{if } i*j \leq 1000 \wedge i, j \neq \perp \\ \{\perp\} & \text{otherwise} \end{cases}$$

$$\delta_2(-, i, j) = \begin{cases} \{i-j\} & \text{if } i \geq j \wedge i, j \neq \perp \\ \{\perp\} & \text{otherwise} \end{cases}$$

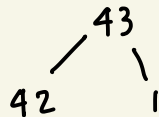


$\Rightarrow t \in L(A)$

t' :



$\tau_{t'}$:



$\Rightarrow t' \notin L(A)$

What (tree-)language $L(A)$ does A recognize (i.e. accept)?

The collection of languages L for which $L = L(A)$ for some tree automaton A is called regular tree languages.

Such languages are closed under:

- intersection $L(A) \cap L(B)$
- union $L(A) \cup L(B)$
- complement $\overline{L(A)}$

Proven by exhibiting constructions on tree automata which resemble corresponding ones for word automata, e.g. product for intersection and subset construction for determinization in order to form complement.

Recall our sketch of an FOL(K)-separation algorithm involved checking emptiness of the language of a tree automaton which is the product of several automata, one for each example structure.

Example: Product construction

Given tree automata $A = (Q_A, \Sigma, F_A, \delta_A)$ and $B = (Q_B, \Sigma, F_B, \delta_B)$, define:

$$A \times B := (Q_A \times Q_B, \Sigma, F_A \times F_B, \delta_{A \times B}), \text{ with}$$
$$\delta_{A \times B}(s, \bar{p}) = \delta_A(s, \pi_1(\bar{p})) \times \delta_B(s, \pi_2(\bar{p}))$$

Claim: $L(A \times B) = L(A) \cap L(B)$.

Emptiness Problem for Tree Automata

Given $A = (Q, \Sigma, F, \delta)$, determine whether $L(A) = \emptyset$, and if not, output $t \in L(A)$.

$$\text{Size}(A) := |Q| + \sum_{\substack{s \in \Sigma, \\ \bar{q} \in Q^{\text{arity}(s)}}} |\delta(s, \bar{q})| \cdot (|\bar{q}| + 2)$$

Data structure for tree automaton emptiness algorithm

- For each $s \in \Sigma$, $\bar{q} \in Q^*$
Keep a counter $c_{s, \bar{q}} \in \{0, |\bar{q}|\}$ initialized at 0.
- For each $q \in Q$:
 - a bit, $\text{visited}(q)$
 - pointer to tree, $\text{witness}(q)$
 - pointer $\ell(q)$ to list of entries $(s, \bar{q})$ for which $q \in \bar{q}$.

Emptiness Algorithm

Worklist = empty

visited(q) = false for every $q \in Q$

witness(q) = null for every $q \in Q$

For $a \in \Sigma_0$ with $\delta_0(a) = W \subseteq Q$:

For $q \in W$:

visited(q) := true

witness(q) := a

Worklist.append($\underbrace{[(s, \bar{q}), K] : (s, \bar{q}) \in l(q)}_{\text{work}(l(q), q)}$)

where $K = \#$ occurrences
of q in $\bar{q}$

While Worklist not empty:

$(s, \bar{q}), K \leftarrow \text{Worklist.remove}$

$c_{s, \bar{q}} := c_{s, \bar{q}} + K$

if $c_{s, \bar{q}} = |\bar{q}|$:

for $q \in \delta(s, \bar{q})$:

if not visited(q):

visited(q) := true

witness(q) := $(s, \text{witness}(\bar{q}))$

Worklist.append($\text{work}(l(q), q)$)

For $q \in F$:

if visited(q) then return witness(q)

return null

Claim: Emptiness Algorithm runs in time $O(\text{Size}(A))$.

Recall our sketch of an FOL(K)-separation algorithm:

1. Define tree automaton A_G s.t.

$$L(A_G) = \{ \text{ast}(\varphi) : \varphi \in L(G) \}.$$

2. Define tree automata A_M , for $M \in P$, s.t.

$$L(A_M) = \{ \text{ast}(\varphi) : M \models \varphi \}.$$

3. Define tree automata B_M , for $M \in N$, s.t.

$$L(B_M) = \{ \text{ast}(\varphi) : M \not\models \varphi \}.$$

4. Determine whether $L(A)$ is empty, where

$$A := A_G \times \bigtimes_{M \in P} A_M \times \bigtimes_{M \in N} B_M$$

Can construct $\varphi \in L(A)$ and determine non-emptiness in time $\text{poly}(|A|)$.

Let's fill in these details now ...

A ranked alphabet for $FOL(K)$ over a relational signature τ .

For simplicity, assume $\tau = \{R^2\}$ and $V = \{x, y\}$.

So our atomic formulas consist of

$$\{ R(x, x), R(x, y), R(y, x), R(y, y), \\ x = x, \quad x = y, \quad y = x, \quad y = y \} =: \text{Atomic}$$

$$\text{Let } \Sigma_0 = \text{Atomic}$$

$$\Sigma_1 = \{ \top, \exists x, \exists y, \forall x, \forall y \}$$

$$\Sigma_2 = \{ \wedge, \vee \}.$$

$$\Sigma_{\text{simple}} := \Sigma_0 \sqcup \Sigma_1 \sqcup \Sigma_2$$

Previous slide uses $\text{ast}(\varphi)$ to denote term corresponding to φ .

It remains for us to define the automata A_G , A_M , and B_M .

Grammar automaton by example...

$$G: \quad S ::= a \mid b \mid S \wedge N \\ N ::= N \vee N \mid \neg S$$

$$A_G = (Q = \{S, N\}, \Sigma = \{a^0, b^0, \neg^1, \vee^2, \wedge^2\}, F = \{S\}, \delta_G), \text{ where}$$

$$\begin{aligned} \delta_G(\vee, (N, N)) &= \{N\} & \delta_G(\neg, S) &= \{N\} \\ \delta_G(\wedge, (S, N)) &= \{S\} & \delta_G(a) &= \{S\} = \delta_G(b) \end{aligned}$$

Note: If the grammar has rules with nontrivial nesting, e.g. $S ::= \neg(\neg(N))$, then we flatten them by introducing new nonterminals and rules:

$$S ::= \neg N_{\text{fresh}}, \quad N_{\text{fresh}} ::= \neg N$$

Claim: $\text{Size}(A_G) = O(\text{Size}(G))$.

In order to define A_M for positive example $M \in P$, we will use a more succinct automaton which can be compiled to a (bottom-up) tree automaton at the cost of an exponential increase in size, analogous to the subset construction for determinizing word automata.