Tree Automata and Rewriting

Ralf Treinen

Université Paris Diderot
UFR Informatique
Laboratoire Preuves, Programmes et Systèmes

treinen@pps.jussieu.fr

July 23, 2010

Definition Tree Automaton

A tree automaton (TA) is a tuple $\mathcal{A} = (\Sigma, Q, Q_a, \Delta)$ where

- Σ: finite signature;
- $Q \parallel \Sigma_1$: set of states, a finite set of unary symbols;
- $Q_a \subseteq Q$: set of accepting states;
- Δ: finite set of transition rules, rewrite rules of the form

$$f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n))$$

where $f \in \Sigma_n$, $q, q_1, \ldots, q_n \in Q$, $x_1, \ldots, x_n$ different variables.

Languages defined by a TA

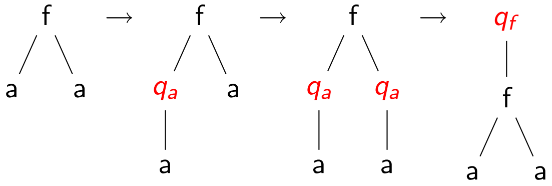
Given a TA $\mathcal{A} = (\Sigma, Q, Q_a, \Delta)$:

- The language of trees accepted in state q ($q \in Q$) is $L_{\mathcal{A},q} = \{t \in T(\Sigma) \mid t \rightarrow^* q(t)\}$.
- $\mathcal{A}$ accepts a tree $t \in T(\Sigma)$ if $t \rightarrow^* q(t)$ for some $q \in Q_a$.
- The language recognized by $\mathcal{A}$ $L(\mathcal{A})$ is the set of all Σ-trees accepted by $\mathcal{A}$.
- A language $L \subseteq T(\Sigma)$ is recognizable if $L = L(\mathcal{A})$ for some tree-automaton $\mathcal{A}$.

Example

$\Sigma_0\{a\}$, $\Sigma_1 = \{g\}$, $\Sigma_2 = \{f\}$.

$$a \;\rightarrow\; q_a(a) \qquad f(q_a(x_1), q_a(x_2)) \;\rightarrow\; q_f(f(x_1, x_2))$$



Remarks

- States "move" from the leaves to the root.
  This is also called a bottom-up automaton.
- Rules for constants ($a \longrightarrow q(a)$) initiate the process, they replace the notion of "initial states".
- The automaton may be non-deterministic:
  Two different rules with the same left-hand side.
- The automaton may be incomplete:
  No rule for some left-hand side.

Abbreviated Notation

Simplified notation (often used in examples):

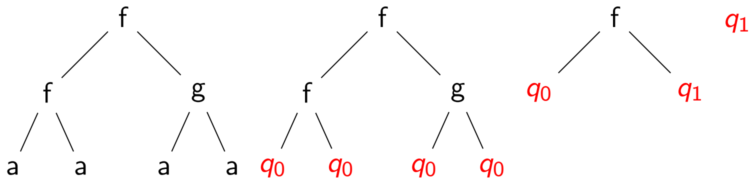
- States are constants (instead of unary symbols)
- Rules: $f(q_1, \ldots, q_n) \rightarrow q$
- Tree t is accepted when $t \rightarrow^* q \in Q_a$
- This would give raise to the same notion of recognizability
- . . . however, this no longer the case for certain extensions of TA (see Lecture 3).

We use this notation only as abbreviation (one does not mention that the system keeps track of the part of the tree already treated).

Example in abbreviated notation

$$\begin{array}{lclclclclcl} a & \rightarrow & q_0 & \quad f(q_0,q_0) & \rightarrow & q_0 & \quad f(q_0,q_1) & \rightarrow & q_1 \\ & & & \quad g(q_0,q_0) & \rightarrow & q_1 & \quad f(q_1,q_0) & \rightarrow & q_1 \end{array}$$

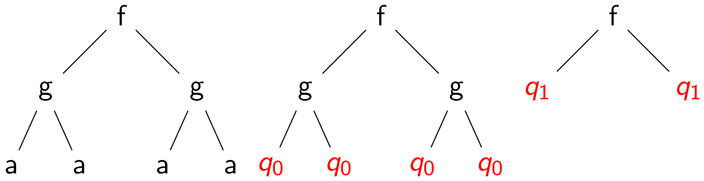
What is the language recognized in q_1 by this automaton? The set of trees containing exactly one g.



Example in abbreviated notation (cntd.)

$$
\begin{array}{rclrclrcl}
a & \rightarrow & q_0 & f(q_0, q_0) & \rightarrow & q_0 & f(q_0, q_1) & \rightarrow & q_1 \\
 & & & g(q_0, q_0) & \rightarrow & q_1 & f(q_1, q_0) & \rightarrow & q_1
\end{array}
$$

Example of a blocking execution. This is tree is not accepted in q_1.



Relation to Word Automata

Any finite word automaton can be seen as a tree automaton:

- Alphabet Γ yields a (monadic) signature Σ with

$$\Sigma_1 = \Gamma, \Sigma_0 = \{\epsilon\}$$

- Recognizable word languages ⇔ Recognizable tree languages over monadic signatures with exactly one constant:

$$\begin{tikzcd}
\text{word } abc & \Leftrightarrow & a \arrow[d, dash] \\
 & & b \arrow[d, dash] \\
 & & c \arrow[d, dash] \\
 & & \epsilon
\end{tikzcd}$$

As with word automata:

- Every automaton can be transformed into an equivalent deterministic automaton (potential explosion of state space!);

Convenient but unessential extensions:

- ϵ-transitions : $q_1(x) \longrightarrow q_2(x)$, can be eliminated by computing ϵ-closure of states;
- "Big step" transitions: $t \longrightarrow q$ (in abbreviated notation), where t is a linear $\Sigma \cup Q$-term. Can be eliminated by introducing intermediate states.

Elimination of big-step transitions

$$f(g(q_1, q_2), h(q_3)) \to q_4$$

Eliminate by introducing auxiliary states:

$$\begin{array}{rcl} g(q_1, q_2) & \to & p_1 \\ h(q_3) & \to & p_2 \\ f(p_1, p_2) & \to & q_4 \end{array}$$

with p_1, p_2 fresh state symbols.

Example big-step transition

For any left-linear rewrite rule $l \longrightarrow r$ the set of reducible terms is recognizable.
The automaton consists of three parts:

- Recognize any term in state q_*.
- Rule $l' \longrightarrow q_m$, where l' is l with all variables replaced by q_*.
- Rules to propagate q_m to the root, q_m is the only accepting state.

More on recognizing reducible terms

- Why is it essential that the rewrite rule is left-linear ?
- Tree automata only "see" the states in which direct subterms are recognized, not the subterms themselves.
- The set of instances of $f(x, x)$ is not recognizable.
- This is formalized in the pumping lemma for tree automata which generalizes the classical pumping lemma for word automata by using tree contexts.
- In general: Beware of non-linearity when trying to generalize from words to tree!

TA as top-down state machines

A top-down tree automaton (TA) is defined like a bottom-up tree automaton, except that

- additionally, $Q_I \subseteq Q$: finite set of initial states;
- Transition rules are now of the form

$$q(f(x_1, \ldots, x_n)) \rightarrow f(q_1(x_1), \ldots, q_n(x_n))$$

where $f \in \Sigma_n$, $q, q_1, \ldots, q_n \in Q$, $x_1, \ldots, x_n$ different variables.

Tree t is accepted if $q(t) \rightarrow^* t$ for some $q \in Q_I$.

Bottom-up versus Top-down Tree Automata

- Both automata models are equivalent in power.
- This is analogous to the fact that recognizable word languages are closed under mirroring, but there is a subtlety:
- Deterministic top-down automata are less expressive than general top-down, or bottom-up automata!
- Example: $\{f(a,a), f(b,b)\}$ is not recognized by a deterministic top-down TA, since a top-down automaton has to guess whether the leaves will be *a*s or *b*s.

TA as Horn clauses

- An automaton clause is a definite Horn clause of the form

$$Q(f(x_1, \ldots, x_n)) \leftarrow Q_1(x_1), \ldots, Q_n(x_n).$$

  where $Q, Q_1, \ldots, Q_n$ are unary predicate symbols, $x_1, \ldots, x_n$ different variables, $f \in \Sigma_n$.
- Tree automaton: set of automata clauses plus a set of accepting predicates. Semantics: minimal fixed point.
- Yields again an equivalent model.
- Horn clauses convenient for several extensions.

Closure under Boolean Operations

The class of recognizable tree languages is effectively closed under

- Union: union of state-disjoint copies automata.
- Complementation: make TA deterministic, invert accepting status of states.
- Intersection: Consequence of the above.

Application: For any left-linear term rewrite system, the set of normal forms is recognizable.

Tree homomorphisms

- Tree homomorphism: associates to each $f \in \Sigma_n$ a term over $t_f \in T(\Sigma, \{x_1, \ldots, x_n\})$.
- Defines a mapping $h\colon T(\Sigma) \to T(\Sigma)$ by

$$h(f(t_1, \ldots, t_n)) = t_f\{\underbrace{x_1 \mapsto h(t_1), \ldots, x_n \mapsto h(t_n)}_{\text{syntactic replacement}}\}$$

- Linear tree homomorphism: all terms t_f are linear.

Examples

$\Sigma_1 = \{\top, \bot\}$, $\Sigma_1 = \{\neg\}$, $\Sigma_2 = \{\wedge, \vee\}$.

- h_1: $t_\wedge = \neg(\vee(\neg(x_1), \neg(x_2)))$ eliminates $\wedge$ in a Boolean expression.
- h_2: $t_\neg = \wedge(x_1, x_1)$: non-linear!
  Set of trees containing only $\neg$ is recognizable ...
  ... but its image under h_2 is not!

Recognizable languages are not closed under arbitrary tree homomorphisms.

Closure under linear tree homomorphisms

Let A recognize L, h a linear tree homomorphism.
Construction of automaton A' recognizing $h(L)$:

- A' has the same states and accepting states as A.
- If A has transition $f(q_1, \ldots, q_n) \rightarrow q$
  Then A' has transition $t_f\{x_1 \mapsto q_1, \ldots, x_n \mapsto q_n\} \rightarrow q$.
- This is an automaton with "big steps", eliminating them possibly introduces additional state symbols.
- Recognizable sets are closed under linear homomorphisms.

Closure and inverse tree homomorphisms

Let A recognize L, h an arbitrary tree homomorphism.
Construction of automaton A' recognizing $h^{-1}(L)$:

- States of A': states of A plus $\{q_*\}$. States are accepting in A; iff accepting in A.
- All terms are accepted in q_*.
- If $t_f\{x_1 \mapsto q_1, \ldots, x_n \mapsto q_n\} \rightarrow_A^* q$
  then A' has transition $f(q_1, \ldots, q_n) \rightarrow q$.
  If $x_i \notin V(t_f)$ this implies $q_i = q_*$!
- Recognizable sets are closed under inverse homomorphisms.

Decision Results

The following properties are decidable:

- Membership: given $\mathcal{A}$ and t, is $t \in L(\mathcal{A})$? (PTIME)
- Emptiness: given A, is $L(\mathcal{A}) = \emptyset$? (PTIME)
- Finiteness: given $\mathcal{A}$, is $L(\mathcal{A})$ finite ? (PTIME)
- Universality: given $\mathcal{A}$, is $L(\mathcal{A}) = T(\Sigma)$?
  (EXPTIME complete since Σ is part of input!)
- ...

Infinite Trees

- Investigation of automata on infinite trees motivated by application to logic (see Lecture 2) starting from the late 60s.
- Groundbreaking work by Michael Rabin (Turing award 1979).
- Infinite Σ-tree τ consists of
  - a tree domain D_τ : a prefix-closed set of tree addresses.
  - a tree labelling $L_\tau : D_\tau \longrightarrow \Sigma$ which is consistent with Σ.

  (Definition adapted to term rewriting)

Automata on Infinite Trees

- On infinite trees, automata must be top-down.
- This means that automata cannot always be made deterministic. This is a problem since ...
- ... the usual construction of a complement automata uses determinisation plus inversion of the acceptance condition!
- When does an automata accept an infinite tree anyway?

Acceptance condition

Naïve approach to generalize top-down tree automata to infinite trees:

> *A tree automaton accepts an infinite tree when it executes without blocking.*

With this naïve definition, the class of recognizable sets would not be closed under complement!

Counterexample with the naïve definition

Signature $\Sigma_1 = \{a, b\}$ (infinite strings!)

Language $L = \{t \in T(\Sigma) \mid b \notin t\} = \{a \cdots a \cdots\}$

Automaton recognizing L:

$$\begin{array}{rcl} Q_I & = & \{q\} \\ q(a(x)) & \rightarrow & a(q(x)) \end{array}$$

Counterexample with the naïve definition (cntd.)

$\bar{L} = \{\{t \in T(\Sigma) \mid b \in t\}.$
Assume $\mathcal{A}$ with n states recognizes $\bar{L}$

$$\begin{array}{lll}
A \text{ accepts} & & \overbrace{a \cdots a}^{n} \;\; ba \cdots \\
\text{Run of } \mathcal{A} & q_0\, q_1 \cdots q_n & \cdots \\
\exists i \neq j : q_i = q_j & & \\
A \text{ accepts} & & a \cdots a \;\; a \cdots a\, a \cdots a \; \cdots
\end{array}$$

Definition of automata on infinite trees

- Like a top-down automaton in the finite case, plus $\mathcal{F}$ a set of sets of states.
- A run of an automaton on a tree τ : mapping $r\colon D_\tau \longrightarrow Q$ which is consistent with the transition rules.
- For a path π, $In(r \mid \pi)$ is the set of states that occur infinitely often on path π during run r.
- Run r is accepting when for each path π : $In(r \mid \pi) \in \mathcal{F}$
- Tree τ is accepted when it admits an accepting run.
- There are some other equivalent definitions (the difference lies in the form of the acceptance condition).

Example

- $\bar{L} = \{t \in T(\Sigma) \mid b \in t\}$.
- States : $\{q_0, q_1\}$
- q_0: "No b yet"; q_1: "Have seen some b"
- $Q_I = \{q_0\}$
- Rules:

$$\begin{array}{rcl} q_0(a(x)) & \rightarrow & a(q_0(x)) \\ q_0(b(x)) & \rightarrow & b(q_1(x)) \end{array} \qquad \begin{array}{rcl} q_1(a(x)) & \rightarrow & a(q_1(x)) \\ q_1(b(x)) & \rightarrow & b(q_1(x)) \end{array}$$

- $\mathcal{F} =? \{\{q_1\}\}$

Closure properties and Decision problems

The class of recognizable languages of infinite trees is closed under

- Union: very easy with non-determinism
- Intersection: very easy: execute both automata in parallel
- Complement: extremely difficult proof using game theory (first proof by Michael Rabin 1969)

Emptiness is decidable (Rabin 1969), and NP-complete (Emerson & Jutla 1988)

From word automata to finite tree automata

- Tree automata are an almost straightforward generalization of word automata.
- Inessential exception: Word automata are completely symmetric wrt. inversion of the direction (left/right), deterministic tree automata are not (bottom up/top down).
- Important exception: In the case of trees one has nonlinear patterns ($f(x,x)$) which do not exist in the word case. Classical tree automata cannot deal with non-linearity, extending the model to deal with non-linearity is difficult (see Lecture 3).

From finite to infinite tree automata

- Difficulty when going from the finite to the infinite case: find the right form of acceptance condition.
- Automata on infinite trees are intrinsically non-deterministic
- Finally one gets the same closure properties (with a lot of sweat for complementation), and decision results (with somewhat worse complexity), as for finite trees.

Literature (1)

All about **Finite** Tree Automata: **TATA**

Tree
Automata
Techniques and
Applications

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

`http://tata.gforge.inria.fr/`

Literature (2)

Automata on Infinite Trees (and Words):
Wolfgang Thomas:

- Automata on infinite objects. In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics, pages 133-192. Elsevier Science Publishers, Amsterdam, 1990.
  Recommended as introduction and overview.
- Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, Handbook of Formal Languages, volume III, pages 389-455. Springer, New York, 1997.
  Advanced level, contains a proof of the Rabin complementation theorem using game theory.