Tree Automata and Rewriting

Ralf Treinen

Université Paris Diderot
UFR Informatique
Laboratoire Preuves, Programmes et Systèmes

treinen@pps.jussieu.fr

July 23, 2010

What happened at the last episode

- Automata can be used (in some cases) to model FO-structures.
- Crucial properties of automata: emptiness decidable, closure under Boolean operations, but also under projection and cylindrification.
- Automata on finite or infinite words or trees can be used.
- Yields decidability of the logic $S2S$, probably the "strongest" known decidability result of a FO theory.

Extensions of the "classical" automaton model

1. Alternating automata
2. Two-way automata
3. Equational tree automata theory
4. Automata with constraints
5. Automata on different tree models, in particular unranked trees (like XML documents)

Definition

- Standard tree automata are given by a set of Horn clauses

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

where $x_1, \dots, x_n$ are distinct variables.

- Alternating tree automata:

$$Q(f(x_1, \dots, x_n)) \leftarrow Q_1(y_1), \dots, Q_m(y_m)$$

where $x_1, \dots, x_n$ are distinct variables (but $y_1, \dots, y_m$ are not necessarily distinct)

- Example:

$$q_1(f(x_1, x_2)) \leftarrow q_2(x_1), q_3(x_2), q_4(x_2)$$

  The subtree x_2 must be recognized both in state q_3 and q_4.
- Any alternating TA can be transformed into an equivalent standard TA (possibly exponentially bigger).
  Idea: use states $(q_1 \wedge \ldots \wedge q_n)(x)$, expressing that x is recognized in all of $q_1 \ldots, q_n$.
  Details: Exercise!

- Further generalization:

$$Q(f(x_1, \dots, x_n)) \leftarrow t$$

  with t positive Boolean combination of $Q_1(y_1), \dots, Q_m(y_m)$.
- Boolean operations are very easy in this form.
- Transformation into standard TA as before

Definition

- Word automata: two-way automata have in addition an indication of the direction (they are like a Turing machine on a read-only tape).
- Equivalent to standard word automata (see Hopcroft-Ullman)
- Generalization to trees : Pop clauses

$$Q(x) \leftarrow Q'(t)$$

  where t is a linear term.
- Example:

$$\begin{array}{rcll} Q(\mathrm{pair}(x_1, x_2)) & \leftarrow & Q(x_1), Q(x_2) & \text{push clause} \\ Q(x_1) & \leftarrow & Q(\mathrm{pair}(x_1, x_2)) & \text{pop clause} \\ Q(x_2) & \leftarrow & Q(\mathrm{pair}(x_1, x_2)) & \text{pop clause} \end{array}$$

Expressivity

- Two-Way automata are as expressive as standard TA (even when combined with alternation)
- Various formats of alternating two-way automata
- Class $\mathcal{H}$ (Nielson&Nielson&Seidl, Goubault): Horn clauses

$$H \leftarrow P_1(t_1), \ldots, P_n(t_n)$$

where H is either of the form $P(x)$ or $P(f(x_1, \ldots, x_n))$, $x_1, \ldots, x_n$ distinct.
All predicate symbols are unary!

Definition

Consider terms modulo an equational theory. Two different definitions:

1. When TA are seen as term rewrite systems, equational axioms apply to terms containing state symbols. This is the definition by Hitoshi Ohsaki.
2. When TA are seen as Horn clauses, equational axioms only apply to "data terms", but not to state symbols since they are predicate symbols. This is the definition by Goubault-Larrecq&Verma.

This makes a difference for axioms like $x \oplus x = 0$.
They coincide in case of linear equational theories.

Properties

- modulo AC: well behaved (Boolean closure, everything decidable)
- modulo A: not closed under $\cap$ or complement. Emptiness decidable but not universality (hence, also not $\equiv$ or $\subseteq$).
- In general very sensible to change of format of rules that is innocent in the non-equational case (epsilon rules, alternation, two-way, ...).
- See the papers by Ohsaki, and Goubault&Verma.

Definition

- Standard Tree Automaton: rewrite rules

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

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

- Constrained Tree Automaton: constrained rewrite rules:

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

  where $f \in \Sigma_n$, $q, q_1, \ldots, q_n \in Q$, $x_1, \ldots, x_n$ different variables, c a constraint with $free(c) \subseteq \{x_1, \ldots, x_n\}$.

Reminder: Constraints

- Constraint: 1st-order formula with a fixed interpretation. Systems of constraints are usually required to be closed under $\wedge$ and $\exists$.
- Constrained rewrite rule $l \to r \mid c$ where $\mathit{free}(c) \subseteq \mathit{free}(l)$: rewrites a ground term $C[l\sigma]$ to $C[r\sigma]$ if $\sigma \models c$.
- Constraint systems most interesting for us: equations and disequations between terms.

Example: equalities between brothers

- Most basic form: a constraint is a conjunction of equations between variables.
- Example:

$$\begin{array}{rcl} a & \rightarrow & q(a) \\ f(q(x_1), q(x_2)) & \rightarrow & q(f(x_1, x_2)) \mid x_1 = x_2 \end{array}$$

- Recognizes the set of balanced binary trees, which is a non-regular set!
- This is the class of tree automata with equality constraints (or "equality tests") between brothers.

Properties of TA with equality between brothers

- Closed under union: trivial, since automata non-deterministic
- Closed under intersection: product of two automata, execute in parallel
- not closed under complement: cannot recognize the set of binary trees that are not balanced.
- cannot be made deterministic (that would require disequality constraints)
- Emptiness decidable? see next slides.

Review: Emptiness of standard TA

Compute the set R of reachable states as a fixed point:

R `:=` $\emptyset$
`while` $\exists\ f(q_1, \ldots, q_n) \rightarrow\ \ q \in \Delta : q_1, \ldots, q_n \in R,\ \ q \notin R$
`do`
  R `:=` $R\ \cup\ \{q\}$
`od`
`return` $R \cap Q\_a = \emptyset$

Emptiness of TA with = between brothers

- We now need to know, for any set of states $q_1, \ldots, q_n$, whether $L(q_1) \cap \ldots \cap L(q_n) \neq \emptyset$.
- This is needed for rules like

$$f(q_1(x_1), q_2(x_2)) \to q(f(x_1, x_2)) \mid x_1 = x_2$$

  since we have to know whether $\exists x \in L(q_1) \cap L(q_2)$!
- R is now a set of set of states!
- Simplification : only constants and binary functions

Emptiness of TA with = between brothers

- Initially: $R := \{\{q_1, \ldots, q_n\} \mid a \to q_1, \ldots a \to q_n \in \Delta\}$
- Suppose $f(p_1, q_1) \to r_1, \ldots, f(p_n, q_n) \to r_n \in \Delta$.
  When do we add $\{r_1, \ldots, r_n\}$ to R ?
- If there are no constraints $x_1 = x_2$ in these rules:
  Condition is $\{p_1, \ldots, p_n\} \in R, \{q_1, \ldots, q_n\} \in R$
- If there is constraint $x_1 = x_2$ in these rules:
  Condition is $\{p_1, \ldots, p_n, q_1, \ldots, q_n\} \in R$
- Finally: $\exists P \in R : P \cap Q_a \neq \emptyset$?

Extension: TA with $=$ and $\neq$ between brothers

- Now closed under $\cup$, $\cap$, and complement
- Automata can be made deterministic.
- Emptiness still decidable, but more difficult: we have to count the number of terms recognized in a state. Why ?
-

$$f(q(x_1), q(x_2), q(x_3)) \rightarrow p \mid x_1 \neq x_1 \wedge x_2 \neq x_3 \wedge x_1 \neq x_3$$

q is reachable when $\#L(q) \geq 3$.

Towards a further generalization

TA with comparison between brothers can be written more compact:

$$f(q_1(x_1), q_2(x_2), q_3(x_3)) \rightarrow q(f(x_1, x_2, x_3)) \mid x_1 = x_2 \wedge x_2 \neq x_3$$

could be written shorter

$$f(q_1(x), q(x), q_3(x_3)) \rightarrow q(f(x, x)) \mid x \neq x_3$$

Generalization: TA with deep comparisons

$$t[q_1(x_1), \ldots, q_n(x_n)] \to q(t[x_1, \ldots, x_n]) \mid c$$

where

- t is a term
- variables $x_1, \ldots, x_n$ not necessarily distinct
  this serves to express equality constraints
- c is a conjunction of disequalities

Example with deep comparisons

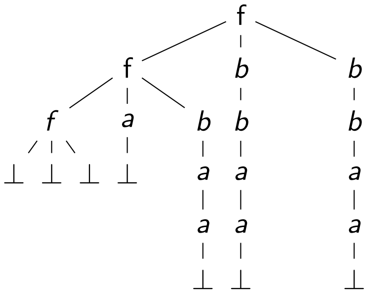
- $\Sigma_0\{\epsilon\}, \Sigma_2 = \{f, g\}$
- $Q = Q_a = \{q\}$
-

$$
\begin{array}{rcl}
\epsilon & \rightarrow & q(\epsilon) \\
f(f(q(x), q(y)), f(q(y), q(z))) & \rightarrow & q(f(f(x, y), f(y, z))) \\
& \dots &
\end{array}
$$

- This can be used to recognize grids
- Undecidability of emptiness is the consequence!

Another proof of undecidability

Post Correspondence Problem: $\{(a, aab), (abb, b)\}$



PCP solution sequence $((\epsilon, \epsilon), (a, aab), (aabb, aabb))$

Another proof of undecidability (cntd.)

Given instance of PCP $\{(v_1, w_1), \ldots, (v_n, w_n)\}$
Automaton that recognizes solution sequences (using big-step transitions):

$$\begin{array}{rcl} f(\bot,\bot,\bot) & \rightarrow & q \\ f(q(f(x, y, z)), v_i(y), w_i(z)) & \rightarrow & q(f(f(x, y, z), v_i(y), w_i(z))) \\ q(f(x, y, y)) & \rightarrow & q_a(f(x, y, y)) \mid y \neq \bot \end{array}$$

Accepting state q_a.
Equality constraints between different "levels". In the preceding proof, constraints are only between cousins.

Reduction Automata

- Reduction automata (Caron, Comon, Coquidé, Dauchet, Jacquemard '94) may perform an unlimited number of disequality tests, but only a limited number of equality tests on each branch of the tree.
- Formally: Q is equipped with an order $\leq$.
  For rules $f(q_1, \ldots, q_n) \to q \mid c$ one requires $\forall i : q_i \leq q$.
  If c contains an equality constraint then $\forall i : q_i < q$.
- Consequence: on each branch at most $\#Q - 1$ equality constraints used.

Constraints in Reduction Automata

- Attention: no big-step transitions in the definition of RA (one needs a fine control which states may occur where).
- In order to use deep equality constraints in absence of big-step transitions one uses path notation, e.g. $12 = 231$.
- Path constraint $12 = 231$ is satisfied by $f(t_1, t_2)$ iff

$$t_1 |_2 = t_2 |_{31}$$

Example Reduction Automaton

Recognize all terms that contain an instance of $f(f(x, y), x)$

$$
\begin{array}{rcl@{\qquad}rcl}
a & \rightarrow & q_0 & f(q_0, q_0) & \rightarrow & q_0 \\
f(q_0, q_0) & \rightarrow & q_1 & f(q_1, q_0) & \rightarrow & q_2 \mid 11 = 2 \\
f(q_0, q_2) & \rightarrow & q_2 & f(q_2, q_0) & \rightarrow & q_2
\end{array}
$$

Order of states: $q_0 < q_1 < q_2$
Accepting state: q_2

Making the automaton deterministic

- q_1: Instances of $f(x, y)$ that do not contain an instance of $f(f(x, y), x)$:

$$\begin{array}{rcl} a & \rightarrow & q_0 \\ f(q_0, q_0) & \rightarrow & q_1 \\ f(q_1, q_0) & \rightarrow & q_1 \mid 11 \neq 2 \end{array} \qquad \begin{array}{rcl} & & \\ f(q_0, q_1) & \rightarrow & q_1 \\ f(q_1, q_1) & \rightarrow & q_1 \mid 11 \neq 2 \end{array}$$

- q_2: Innermost instances of $f(f(x, y), x)$:

$$f(q_1, q_0) \rightarrow q_2 \mid 11 = 2 \qquad f(q_1, q_1) \rightarrow q_2 \mid 11 = 2$$

- Propagation to the root:

$$f(q_i, q_j) \rightarrow q_2 \quad \text{if } i = 2 \text{ or } j = 2 \text{ (5 rules)}$$

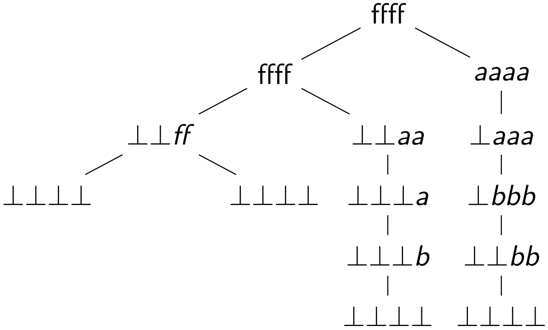
Properties of Reduction Automata

- They can be used to check whether a non-linear pattern matches a tree.
- They enjoy closure under Boolean operations
- Emptiness is decidable for deterministic reduction automata (but undecidable for non-deterministic automata)
- Exercice: Show that the following is undecidable: given an automaton with equality test between brothers, and a reduction automaton, is the intesection of their recognized languages empty?

What about cylindrification and projection?

- Cylindrification of reduction automata would amount to automata with component-wise constraints since no constraints must be applied on the components added by cylindrification.
- Emptyness of automata with component-wise constraints is undecidable even when there is only one application of a constraint, and that application is at the root of the tree.

Automata with component-wise constraints



PCP solution sequence $((\epsilon, \epsilon), (a, aab), (aabb, aabb))$
Automaton, plus tests at the root: $1_3 = \epsilon_1 \wedge 1_4 = \epsilon_2$

The theory of reducibility

- Reduction automata have been used to show decidability of the theory of reducibility (Herbrand, without equality, but with predicates "t matches x" for fixed t.)
- Reduction automata are not closed under cylindrification, and furthermore one cannot close them cylindrification and retain decidability.
- Isn't there a contradiction?

Reducibility is a Monadic structure

- Monadic structure: only unary predicate symbols.
- If the language is monadic, then any FO formula can be rewritten into a Boolean combination of formulas of the form

$$\exists x\Big(P_1(x) \wedge \ldots \wedge P_n(x) \wedge \neg Q_1(x) \ldots \wedge \neg Q_m(x)\Big)$$

  (Löwenheim '15, Ackermann '54)
- Consequences: For monadic structures, Boolean closure plus decidability of emptiness are sufficient! No need for cylindrification and projection.

A Unified Model of Constrained Automata?

- Automata with tests between brothers and reduction automata are incomparable in expressivity.
- Is there a Unified Model of constraint automata that comprises these two classes, has decidable emptiness, and good closure properties (in particular, $\cap$)?
- The answer is no, since it is undecidable whether the intersection of the languages recognized by a reduction automaton and an automaton with constraints between brothers is empty!
- Can we still achieve something?
- Candidate: Tree Automata with One Memory (Comon&Cortier&Mitchell 2001) ?

Is Herbrand Theory Automatic?

- Herbrand: The FO-theory of the structure of finite terms with unification constraints ($x = f(y, z)$ etc.)
- This theory is known to be decidable (Malc'ev '71, Comon&Lescanne '89, Maher '88). Proof: quantifier elimination.
- Is this an automatic structure, for some useful notion of tree automata that allows to conclude decidability?
- Theory of reducibility: is automatic, but does not allow for unification constraints.
- Problem: constraint $x_1 = f(x_2, x_3)$ is not recognized by a standard tree automaton (if decoding function ν is the identity).

Is Herbrand Theory Automatic? (cntd.)

- Reduction Automata ? Are not closed under projection and cylindrification ☹
- The FO-order theory of an RPO has constraints $x = f(y)$, and there strings are encoded as trees! Trick: strings are represented in reverse order, so that the constraint is trivially expressed as an automaton (one just adds an f at the end of one component).
  Can this be used as an encoding trick?
- Problem: the set of meaningful trees L_δ must be recognizable! For that reason, this has so far only succeeded for one binary function symbol and infinitely many constants (Comon&Podelski).