Symbolic Tree Automata

Margus Veanes, Nikolaj Bjørner

Microsoft Research, Redmond, WA, USA

Abstract

We introduce symbolic tree automata as a generalization of finite tree automata with a parametric alphabet over any given background theory. We show that symbolic tree automata are closed under Boolean operations, and that the operations are effectively uniform in the given alphabet theory. This generalizes the corresponding classical properties known for finite tree automata.

Keywords: tree automata, automata algorithms, logic, satisfiability modulo theories

1. Introduction

Finite word automata and finite tree automata provide a foundation for a wide range of applications in software engineering, from regular expressions to compiler technology and specification languages. Despite their immense practical use, explicit representations are not feasible in the presence of finite large alphabets. They require each transition to encode only a single element from the alphabet. For example, string characters in standard programming languages (such as the `char` type in C#) use 16-bit bit-vectors, an explicit representation would thus require an alphabet of size 2^{16}. Moreover, most common forms of finite automata do not support infinite alphabets.

A practical solution to the representation problem is *symbolic tree automata*. They are an extension of classical tree automata that addresses this problem by allowing transitions to be labeled with arbitrary formulas in a specified label theory. While the idea of allowing formulas is straightforward, typical extensions of finite tree automata often lead to either undecidability of the emptiness problem, such as tree automata with equality and disequality constraints [1], or many extensions lead to nonclosure under complement, such as the generalized tree set automata class [1], finite-memory tree automata [2] that generalize finite-memory automata [3] to trees, or unranked data tree automata [4]. We show that this is not the case for symbolic tree automata. The key distinction is that the extension here is with respect to *characters* rather than adding symbolic *states* or adding constraints over whole *subtrees*.

The symbolic extension is practically useful for exploiting efficient symbolic constraint solvers when performing basic automata-theoretic transformations: it enables a separation of concerns. The solver is used as a black box with a clearly defined interface that exposes the label theory as an effective Boolean algebra. The chosen label theory can be specific to a particular problem instance. For example, even when the alphabet is finite, e.g., 16-bit bit-vectors, it may be useful for efficiency reasons to use integer-linear arithmetic rather than bit-vector arithmetic when the solver is more efficient over integers and when only standard arithmetic operations (and no bit-level operations) are being used. Recent work [5, 6] on symbolic string recognizers and transducers takes advantage of this observation.

We here investigate the case of the more expressive class of symbolic *tree* automata. Even though a symbolic tree automaton is a finite object, a key point is that the number of interpretations for symbolic labels does not need to be finite. For example, as a consequence of our main result (Theorem 2) a label theory may itself be the theory of symbolic tree automata (over some basic label theory).

In order to use classical tree automata algorithms, it is possible to reduce a symbolic tree automaton A into a classical finite tree automaton whose alphabet is given by all of the satisfiable Boolean combinations of guards that occur in A. However, such a transformation is in

Email addresses: `margus@microsoft.com` (Margus Veanes), `nbjorner@microsoft.com` (Nikolaj Bjørner)

URL: `http://research.microsoft.com/~margus` (Margus Veanes), `http://research.microsoft.com/~nbjorner` (Nikolaj Bjørner)

general not practical because it introduces an exponential increase in the size of the automaton before the actual algorithm is applied. Moreover, when more than one automaton are involved, this has to be done up front for all predicates that occur in all the automata in order to define the common alphabet. A concrete example of such a blowup is given in [7, Example 2].

2. Definition of symbolic tree automata

We introduce an extension of tree automata with an effective encoding of labels by predicates that denote *sets* of labels, rather than individual labels. We assume a countable *background universe* $\mathscr{B}$. A *predicate* φ over $\mathscr{B}$ is a finite representation of a subset $[\![\varphi]\!]^{\mathscr{B}}$ of $\mathscr{B}$; we write $[\![\varphi]\!]$ when $\mathscr{B}$ is clear from the context. We assume given an effectively enumerable set of predicates Σ such that, for each element $a \in \mathscr{B}$ there is $\hat{a} \in \Sigma$ such that $[\![\hat{a}]\!] = \{a\}$, $\top, \bot \in \Sigma$ such that $[\![\top]\!] = \mathscr{B}$ and $[\![\bot]\!] = \emptyset$, and Σ is effectively closed under Boolean operations: for all $\varphi, \psi \in \Sigma$, we have $\varphi \wedge \psi \in \Sigma$, $\varphi \vee \psi \in \Sigma$, $\neg\varphi \in \Sigma$, where $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$, $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$, and $[\![\neg\varphi]\!] = \mathscr{B} \setminus [\![\varphi]\!]$. We write $\varphi \equiv \psi$ for $[\![\varphi]\!] = [\![\psi]\!]$. We say that $(\Sigma, [\![\cdot]\!]^{\mathscr{B}})$ (or Σ, when $[\![\cdot]\!]^{\mathscr{B}}$ is clear from the context) is an *effective Boolean algebra over* $\mathscr{B}$. We say that Σ is *decidable* if the problem of deciding $\varphi \equiv \bot$ for $\varphi \in \Sigma$ is decidable.

Example 1. An example of a decidable effective Boolean algebra is $(LA(x), [\![\cdot]\!]^{\mathbb{Z}})$ where $[\![\cdot]\!]^{\mathbb{Z}}$ is the standard interpretation of integer arithmetic with , and $LA(x)$ is an effectively enumerable set of all quantifier free integer-linear arithmetic formulas, with one fixed free variable x, e.g., $[\![0 < x \wedge x+1 < 3]\!] = [\![0 < x]\!] \cap [\![x+1 < 3]\!] = \{1\}$. ⊠

In this paper we focus on *binary* trees. This will keep the notational overhead at a minimum, while the results can be generalized to non-binary trees through standard encoding techniques. $\mathscr{T}(\mathscr{B})$ is the smallest set such that the *empty tree* $\epsilon \in \mathscr{T}(\mathscr{B})$ and if $a \in \mathscr{B}$ and $t_1, t_2 \in \mathscr{T}(\mathscr{B})$ then $t = \langle a, t_1, t_2\rangle \in \mathscr{T}(\mathscr{B})$, where a is the *label* of t, denoted $label(t)$, t_1 is the *left subtree* of t, denoted $left(t)$, and t_2 is the *right subtree* of t, denoted $right(t)$.

For example $\langle 1, \langle -2, \epsilon, \langle 3, \epsilon, \epsilon\rangle\rangle, \langle 4, \epsilon, \epsilon\rangle\rangle \in \mathscr{T}(\mathbb{Z})$.

Definition 1. A *symbolic tree automaton (STA) A* is a tuple $(\Sigma, Q, Q^{\text{L}}, Q^{\text{R}}, \Delta)$ where Σ is an effective Boolean algebra called the *label theory* of A, Q is a finite set of *states*, $Q^{\text{L}} \subseteq Q$ is a set of *leaves*, $Q^{\text{R}} \subseteq Q$ is a set of *roots*, and $\Delta \subseteq Q \times \Sigma \times Q \times Q$ is a finite set of *transitions*.

We use A as a subscript to identify a component, unless A is clear from the context. We write $\mathbf{STA}(\Sigma)$ for an effectively enumerable set of all STAs over Σ. Let $A = (\Sigma, Q, Q^{\text{L}}, Q^{\text{R}}, \Delta) \in \mathbf{STA}(\Sigma)$ be fixed. Given a transition $\rho = (p, \varphi, q_1, q_2) \in \Delta$, let $lhs(\rho)$, $\gamma(\rho)$, and $rhs(\rho)$, denote, respectively, the *left-hand-side* p, the *guard* φ, and the *right-hand-side* (q_1, q_2) of ρ. We use $\overline{q}$ as an abbreviation for (q_1, q_2).

Definition 2. The *language of A for* $q \in Q$, denoted by $\mathscr{L}(A, q)$, is the smallest subset of $\mathscr{T}(\mathscr{B})$ such that: if $q \in Q^{\text{L}}$ then $\epsilon \in \mathscr{L}(A, q)$; if $(q, \varphi, q_1, q_2) \in \Delta$, $a \in [\![\varphi]\!]$, and, for $i \in \{1, 2\}$, $t_i \in \mathscr{L}(A, q_i)$, then $\langle a, t_1, t_2\rangle \in \mathscr{L}(A, q)$. The *language of A* is $\mathscr{L}(A) \stackrel{\text{def}}{=} \bigcup_{q \in Q^{\text{R}}} \mathscr{L}(A, q)$.

Two STAs A and B are *equivalent*, denoted $A \equiv B$, when $\mathscr{L}(A) = \mathscr{L}(B)$.

Let $\bot_{\mathbf{STA}(\Sigma)} \stackrel{\text{def}}{=} (\Sigma, \emptyset, \emptyset, \emptyset, \emptyset)$. Thus $\mathscr{L}(\bot_{\mathbf{STA}(\Sigma)}) = \emptyset$. The following example illustrates a representation of valid Unicode character sequences as an STA that uses UTF16 encoding of surrogate pairs.[1]

The particular feature of the representation is that the trees preserve the length of the original Unicode strings as the length of the rightmost branch. The leftmost branch from any node in the tree is either the node itself when the node is not a surrogate, or a surrogate pair otherwise, and encodes thus a single Unicode symbol.

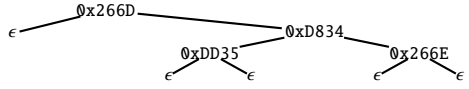
Example 2. Let BV16 stand for quantifier free 16-bit bit-vector arithmetic; BV16 is isomorphic to quantifier free integer linear arithmetic modulo 2^{16}. We use a single fixed free variable x in predicates φ over BV16, thus $[\![\varphi]\!]$ is the set of all values a such that $\varphi[x/a]$ is true. Let

$$\begin{aligned} \mathit{HighSurr} &\stackrel{\text{def}}{=} \texttt{0xD800} \le x \le \texttt{0xDBFF}, \\ \mathit{LowSurr} &\stackrel{\text{def}}{=} \texttt{0xDC00} \le x \le \texttt{0xDFFF}, \end{aligned}$$

Let $A = (\text{BV16}, \{q_{ok}, q_{ls}, q_\epsilon\}, \{q_{ok}, q_\epsilon\}, \{q_{ok}\}, \Delta)$, where

$$\Delta = \left\{ \begin{array}{l} (q_{ok}, \neg \mathit{LowSurr} \wedge \neg \mathit{HighSurr}, q_\epsilon, q_{ok}), \\ (q_{ok}, \mathit{HighSurr}, q_{ls}, q_{ok}), \\ (q_{ls}, \mathit{LowSurr}, q_\epsilon, q_\epsilon) \end{array} \right\}$$

For example, the tree



[1] Complete Unicode alphabet has over one million characters, UTF16 encoding is used to encode the alphabet with 16-bit bit-vectors, where surrogate pairs are used for encoding characters in the upper Unicode range.

is in $\mathscr{L}(A)$ and encodes the Unicode string "♭𝅗𝅥𝅯" of musical symbols, where 𝅗𝅥 is the symbol "cut time" encoded by the surrogate pair (`0xD834`,`0xDD35`). The sequence would be represented by a string `"\u266D\uD834\uDD35\u266E"` (of length 4) in standard programming and scripting languages. ⊠

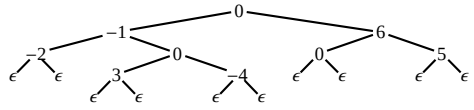
Example 3. We illustrate an STA A that accepts integer-labeled binary trees

$$A = (LA(x), \{q_{root}, q_-, q_0, q_+, q_\epsilon\}, \{q_\epsilon\}, \{q_{root}\}, \Delta),$$

where Δ consists of the transitions

$$(q_{root}, x{=}0, q_-, q_+),\ (q_0, x{=}0, q_+, q_-),$$
$$(q_-, x{<}0, q_-, q_0),\ (q_+, x{>}0, q_0, q_+),$$
$$(q_-, x{<}0, q_\epsilon, q_\epsilon),\ (q_0, x{=}0, q_\epsilon, q_\epsilon),\ (q_+, x{>}0, q_\epsilon, q_\epsilon)$$

The transitions can be understood as a recursive constraint system. The *root* has label 0, its left son is a '-'-node and its right son is a '+'-node. Every '-'-node has a negative label and is either a leaf or its left son is a '-'-node and its right son is an '*0*'-node. Similarly for the other cases. For example, the tree



is in $\mathscr{L}(A)$. ⊠

Classical tree automata theory distinguishes between *top-down* (or *root-to-frontier*) and *bottom-up* (or *frontier-to-root*) recognizers. This classification relates to the intended *direction* of the transitions for accepting or recognizing a tree. The results of the paper are independent of this classification.

A fundamental subclass of STAs is the following, that generalizes the corresponding subclass of deterministic bottom-up (frontier-to-root) tree recognizers [8, Definition 2.1(p. 60)].

Definition 3. A is *deterministic* (*DSTA*) when $|Q^{\mathrm{L}}| = 1$ and, for all $\rho_1, \rho_2 \in \Delta$, if $rhs(\rho_1) = rhs(\rho_2)$ and $\gamma(\rho_1) \wedge \gamma(\rho_2) \not\equiv \bot$ then $lhs(\rho_1) = lhs(\rho_2)$.

For example, the STA in Example 3 is deterministic because the only transitions with equal right-hand-sides are the last three transitions and their guards denote mutually disjoint sets of labels. The STA in Example 2 is not deterministic because $|Q^{\mathrm{L}}| > 1$ (although it is *top-down deterministic*: $|Q^{\mathrm{R}}| = 1$ and, for all $\rho_1, \rho_2 \in \Delta$, if $lhs(\rho_1) = lhs(\rho_2)$ and $\gamma(\rho_1) \wedge \gamma(\rho_2) \not\equiv \bot$ then $rhs(\rho_1) = rhs(\rho_2)$, but it can be converted to a DSTA.

For $\overline{q} \in Q \times Q$ let $\gamma_A(\overline{q})$ denote the disjunction of guards of all transitions in A whose right-hand-side is $\overline{q}$:

$$\gamma_A(\overline{q}) \stackrel{\text{def}}{=} \bigvee\{\gamma(\rho) \mid \rho \in \Delta_A,\ rhs(\rho) = \overline{q}\}$$

Observe that, if there is no transition in A whose right-hand-side is $\overline{q}$ then $\gamma_A(\overline{q})$ is the empty disjunction or $\bot$ (by definition). We use the following subclass of STAs.

Definition 4. A is *total* if, for all $\overline{q} \in Q \times Q$, $\gamma_A(\overline{q}) \equiv \top$.

The STA A in Example 3 is not total because, e.g., $\gamma_A(q_0, q_0) \equiv \bot$. However, any DSTA A can be made total as follows. First, extend A to A_1 by adding a new state q_{sink}. Second, extend A_1 to A_2 by adding the new transition $(q_{sink}, \neg\gamma_{A_1}(\overline{q}), \overline{q})$ for each $\overline{q} \in Q_{A_1} \times Q_{A_1}$. It follows that A_2 is a total DSTA and, for all $q \in Q_A$, $\mathscr{L}(A, q) = \mathscr{L}(A_2, q)$, and thus $\mathscr{L}(A) = \mathscr{L}(A_2)$.

The following basic properties follow for STAs, by structural induction over trees. Let

$$Q_A(t) \stackrel{\text{def}}{=} \{q \in Q_A \mid t \in \mathscr{L}(A, q)\}.$$

Lemma 1. *If A is a DSTA then $|Q_A(t)| \leq 1$ for all $t \in \mathscr{T}(\mathscr{B})$.*

Lemma 2. *If A is total then $|Q_A(t)| \geq 1$ for all $t \in \mathscr{T}(\mathscr{B})$.*

We write *TDSTA* for total DSTA. The construction

$$\complement(A) \stackrel{\text{def}}{=} (\Sigma_A, Q_A, Q_A^{\mathrm{L}}, Q_A \setminus Q_A^{\mathrm{R}}, \Delta_A)$$

defines what the *complement* of a TDSTA A is. For, $X \subseteq \mathscr{T}(\mathscr{B})$, let $\complement(X) \stackrel{\text{def}}{=} \mathscr{T}(\mathscr{B}) \setminus X$.

Lemma 3. *If A is a TDSTA then $\mathscr{L}(\complement(A)) = \complement(\mathscr{L}(A))$.*

Proof. By using Lemmas 1 and 2. ⊠

STAs with ε-moves. An STA A can be extended with a set of *ε-moves* $\Delta^\varepsilon \subseteq Q \times Q$ as an additional (sixth) component. With ε-moves, the definition of $\mathscr{L}(A, q)$ (Definition 2) is extended to include the condition $\mathscr{L}(A, q) \subseteq \mathscr{L}(A, p)$ for all $(p, q) \in \Delta^\varepsilon$.

Similar to finite tree automata, ε-moves can be effectively eliminated and do not affect the expressive power of STAs. The algorithm for ε-moves elimination is independent of the symbolic labels and the standard algorithm for finite tree automata case (see [1, Theorem 1.1.5]), also applies to STAs. However, ε-moves may have practical value because they can increase succinctness among equivalent STA representations, which may improve performance in symbolic analysis, similar to the observations made in [5] regarding the case of SFAs.

3. Determinization of symbolic tree automata

Similar to the case of deterministic frontier-to-root tree recognizers, DSTAs have the same expressive power as general STAs. We lift the classical powerset construction of nondeterministic Rabin-Scott recognizers to STAs. Let $\wp(X)$ denote the powerset of a set X.[2] We write $p \xrightarrow{\varphi} \bar{q}$ for the rule $(p, \varphi, \bar{q})$.

Definition 5. Let $A = (\Sigma, Q, Q^{\mathrm{L}}, Q^{\mathrm{R}}, \Delta)$. The *powerset STA of A* is:

$$\begin{array}{rcl} \wp(A) & \stackrel{\text{def}}{=} & (\Sigma, \wp(Q), \{Q^{\mathrm{L}}\}, \{\mathbf{q} \in \wp(Q) \mid \mathbf{q} \cap Q^{\mathrm{R}} \neq \emptyset\}, \\ & & \{lhs(S) \xrightarrow{\mu(S,\Delta(\bar{\mathbf{q}})\setminus S)} \bar{\mathbf{q}} \mid \\ & & \quad \bar{\mathbf{q}} \in \wp(Q)\times\wp(Q),\ S \subseteq \Delta(\bar{\mathbf{q}})\}) \end{array}$$

$$\begin{array}{rcl} \text{where} \quad \Delta(\mathbf{q}_1, \mathbf{q}_2) & \stackrel{\text{def}}{=} & \{\rho \mid \rho \in \Delta, rhs(\rho) \in \mathbf{q}_1 \times \mathbf{q}_2\} \\ \mu(S, S') & \stackrel{\text{def}}{=} & \bigwedge_{\rho\in S} \gamma(\rho) \wedge \bigwedge_{\rho\in S'} \neg\gamma(\rho) \\ lhs(S) & \stackrel{\text{def}}{=} & \{lhs(\rho) \mid \rho \in S\} \end{array}$$

Note that the role of the state $\emptyset$ in the powerset STA is similar to the role of q_{sink} mentioned after Definition 4.

We obtain the following generalization of [8, Theorem 2.6(p. 65)].

Theorem 1. *For all STAs A:*
(a) $\wp(A)$ is a TDSTA;
(b) for all t, $\{Q_A(t)\} = Q_{\wp(A)}(t)$;
(c) $\wp(A) \equiv A$;
(d) $|Q_{\wp(A)}| = 2^{|Q_A|}$;
(e) $|\Delta_{\wp(A)}| = O(2^{2|Q_A|+|\Delta_A|})$.

Proof. *Proof of (a).* To show that $\wp(A)$ is total, fix a $\bar{\mathbf{q}} = (\mathbf{q}_1, \mathbf{q}_2) \in \wp(Q) \times \wp(Q)$. We need to show that $\gamma_{\wp(A)}(\bar{\mathbf{q}}) \equiv \top$. The empty conjunction is, by definition, $\top$, and thus, when $\mathbf{q}_1 = \emptyset$ or $\mathbf{q}_2 = \emptyset$ then $\Delta(\bar{\mathbf{q}}) = \emptyset$, $\mu(\emptyset, \emptyset) = \top$, and $(\emptyset, \top, \bar{\mathbf{q}}) \in \Delta_{\wp(A)}$, and thus $\gamma_{\wp(A)}(\bar{\mathbf{q}}) \equiv \top$. Assume that $\mathbf{q}_1 \neq \emptyset$ and $\mathbf{q}_2 \neq \emptyset$. Let $\Delta(\bar{\mathbf{q}}) = \{\rho_i\}_{i\in I}$ and let $\varphi_i = \gamma(\rho_i)$ for $i \in I$. We have that

$$\begin{array}{rcl} \gamma_{\wp(A)}(\bar{\mathbf{q}}) & \equiv & \bigvee\{\mu(S, \Delta(\bar{\mathbf{q}}) \setminus S) \mid S \subseteq \Delta(\bar{\mathbf{q}})\} \\ & \equiv & \bigvee_{J\subseteq I}(\bigwedge_{i\in J} \varphi_i \wedge \bigwedge_{i\in I\setminus J} \neg\varphi_i) \quad \equiv \quad \top \end{array}$$

where the last equivalence follows from DeMorgan's laws and simplifications, since *all* possible Boolean combinations of truth assignments of φ_i are included in the disjunction.

To show that $\wp(A)$ is deterministic, let $\bar{\mathbf{q}} \in \wp(Q) \times \wp(Q)$ and let $S_1, S_2 \subseteq \Delta(\bar{\mathbf{q}})$ be such that $S_1 \neq S_2$. Let $S'_i = \Delta(\bar{\mathbf{q}}) \setminus S_i$. It suffices to show that $\mu(S_1, S'_1) \wedge \mu(S_2, S'_2) \equiv \bot$, which follows from $S_1 \neq S_2$ and the definition of $\mu(S, S')$: there exists a transition $\rho \in \Delta(\bar{\mathbf{q}})$ such that $[\![\mu(S_1, S'_1)]\!] \subseteq [\![\gamma(\rho)]\!]$ and $[\![\mu(S_2, S'_2)]\!] \subseteq [\![\neg\gamma(\rho)]\!]$, where, wlog, ρ is such that $\rho \in S_1 \setminus S_2$.

Proof of (b). It follows from (a) and Lemmas 1 and 2 that, for all t, $|Q_{\wp(A)}(t)| = 1$. We prove (b) by induction over trees. The base case $t = \epsilon$ follows immediately from the definitions since $\{Q_A(\epsilon)\} = \{Q^{\mathrm{L}}\} = Q_{\wp(A)}(\epsilon)$. For the induction case suppose $t \neq \epsilon$ and as IH assume that, $\{Q_A(left(t))\} = \{\mathbf{q}_1\} = Q_{\wp(A)}(left(t))$ and $\{Q_A(right(t))\} = \{\mathbf{q}_2\} = Q_{\wp(A)}(right(t))$. Let $\bar{\mathbf{q}} = (\mathbf{q}_1, \mathbf{q}_2)$. The following statements are equivalent by using the definitions and the IH for the equivalence between 2 and 3. Let $p \in Q$.

1. $p \in Q_A(t)$
2. There exists $(p, \varphi, \bar{q}) \in \Delta$ for some $\bar{q} \in \mathbf{q}_1 \times \mathbf{q}_2$ s.t. $label(t) \in [\![\varphi]\!]$.
3. There exists $S \subseteq \Delta(\bar{\mathbf{q}})$ s.t. $label(t) \in [\![\mu(S, \Delta(\bar{\mathbf{q}}) \setminus S)]\!]$ and $p \in lhs(S)$.
4. There exists $\mathbf{q} \in \wp(Q)$ s.t. $p \in \mathbf{q}$ and $Q_{\wp(A)}(t) = \{\mathbf{q}\}$.

The equivalence of 1 and 4 for all p implies that $\{Q_A(t)\} = Q_{\wp(A)}(t)$, that proves (b). Finally, (c) follows from (b) by definition of $Q^{\mathrm{R}}_{\wp(A)}$, and (d) and (e) follow from definitions of $Q_{\wp(A)}$ and $\Delta_{\wp(A)}$. ⊠

Example 4. Recall that $\bot_{\mathbf{STA}(\Sigma)} = (\Sigma, \emptyset, \emptyset, \emptyset, \emptyset)$, so

$$\begin{array}{rcl} \wp(\bot_{\mathbf{STA}(\Sigma)}) & = & (\Sigma, \{\emptyset\}, \{\emptyset\}, \emptyset, \{(\emptyset, \top, \emptyset, \emptyset)\}) \\ \complement(\wp(\bot_{\mathbf{STA}(\Sigma)})) & = & (\Sigma, \{\emptyset\}, \{\emptyset\}, \{\emptyset\}, \{(\emptyset, \top, \emptyset, \emptyset)\}) \end{array}$$

Thus $\mathscr{L}(\complement(\wp(\bot_{\mathbf{STA}(\Sigma)}))) = \mathscr{T}(\mathscr{B})$. ⊠

Similar to the case of classical tree automata, the powerset construction enables us to effectively determinize, and thus complement, STAs.

Determinization of STAs, i.e., Definition 5, can be implemented with bottom-up depth first search. The same basic technique that is used for SFAs [9], that uses minterm generation, has been extended for STAs in [10]. Observe that, in Definition 5, $\mu(S, S')$ (when feasible), defines, at the propositional level, a *minterm* that corresponds intuitively to a nonempty region of a Venn diagram over $\mathscr{B}$ defined by the guards in S and the complements of the guards in S'. Infeasible guards are eliminated eagerly during the bottom-up powerset construction and prune away unreachable left-hand-sides of rules in the constructed powerset STA.

4. Boolean closure of symbolic tree automata

For complete closure under Boolean operations we use the following product construction that is a lifting of the standard product of finite tree automata to STAs.

[2] Recall that $\wp(\emptyset) = \{\emptyset\}$.

Definition 6. Let $A_i = (\Sigma, Q_i, Q_i^{\mathrm{L}}, Q_i^{\mathrm{R}}, \Delta_i)$, for $i = 1, 2$, be STAs. The *product* of A_1 and A_2 is the STA

$$A_1 \times A_2 \stackrel{\text{def}}{=} (\Sigma, Q_1 \times Q_2, Q_1^{\mathrm{L}} \times Q_2^{\mathrm{L}}, Q_1^{\mathrm{R}} \times Q_2^{\mathrm{R}}, \{\rho_1 \times \rho_2 \mid \rho_1 \in \Delta_1, \rho_2 \in \Delta_2\})$$

where, for $i \in \{1, 2\}$ and $\rho_i = (p_i, \varphi_i, q_i, r_i) \in \Delta_i$,

$$\rho_1 \times \rho_2 \stackrel{\text{def}}{=} ((p_1, p_2), \varphi_1 \wedge \varphi_2, (q_1, q_2), (r_1, r_2))$$

Lemma 4 implies that we can effectively intersect languages of STAs. The proof of (a) follows by induction over trees.

Lemma 4. *Let $A_i = (\Sigma, Q_i, Q_i^{\mathrm{L}}, Q_i^{\mathrm{R}}, \Delta_i)$, for $i = 1, 2$, be STAs. Then:*

(a) for all $q_1 \in Q_1, q_2 \in Q_2$,
$\mathscr{L}(A_1 \times A_2, (q_1, q_2)) = \mathscr{L}(A_1, q_1) \cap \mathscr{L}(A_2, q_2)$*;*
(b) $\mathscr{L}(A_1 \times A_2) = \mathscr{L}(A_1) \cap \mathscr{L}(A_2)$;
(c) $|Q_{A_1 \times A_2}| = |Q_1| \cdot |Q_2|$, $|\Delta_{A_1 \times A_2}| = |\Delta_1| \cdot |\Delta_2|$

We use the following definition for constructing the union of tree languages. Assume given two STAs A and B such that $\Sigma_A = \Sigma_B$ and $Q_A \cap Q_B = \emptyset$. The *sum* of A and B, denoted $A + B$, is the STA

$$(\Sigma_A, Q_A \cup Q_B, Q_A^{\mathrm{L}} \cup Q_B^{\mathrm{L}}, Q_A^{\mathrm{R}} \cup Q_B^{\mathrm{R}}, \Delta_A \cup \Delta_B).$$

Trivially, $\mathscr{L}(A_1 + A_2) = \mathscr{L}(A_1) \cup \mathscr{L}(A_2)$.

Assume the *predicates* over the base set $\mathbf{STA}(\Sigma)$ to be defined as the least set that includes $\mathbf{STA}(\Sigma)$, and if φ and ψ are predicates over $\mathbf{STA}(\Sigma)$ then so are $\neg\varphi$, $\psi \wedge \varphi$ and $\psi \vee \varphi$, where for $A, B \in \mathbf{STA}(\Sigma)$, $[\![A]\!] \stackrel{\text{def}}{=} \mathscr{L}(A)$, $[\![\neg A]\!] \stackrel{\text{def}}{=} \mathscr{L}(\complement(\wp(A)))$, $[\![A \wedge B]\!] \stackrel{\text{def}}{=} \mathscr{L}(A \times B)$, and $[\![A \vee B]\!] \stackrel{\text{def}}{=} \mathscr{L}(A+B)$. The denotation is lifted to all predicates over $\mathbf{STA}(\Sigma)$. By slight abuse of notation, we let $\mathbf{STA}(\Sigma)$ also denote the extension with predicates.

A transition $\rho \in \Delta_A$ such that $\gamma(\rho) \equiv \bot$ is called *infeasible*. The size $|\varphi|$ for $\varphi \in \mathbf{STA}(\Sigma)$ is given by the definitions $|\varphi \wedge \varphi'| = |\varphi \vee \varphi'| = |\varphi| + |\varphi'|$, $|\neg\varphi| = |\varphi| + 1$, $|t|$ is the size of term t, and $|A| = 1 + \Sigma_{\rho \in \Delta_A} |\gamma(\rho)|$.

The following lemma is used by Theorem 2.

Lemma 5. *The emptiness problem of an STA A, has complexity $O(n \cdot f(m))$ where $f(m)$ is the complexity of deciding satisfiability of Σ_A for instances of size m and $n = |\Delta_A|$.*

Proof. Using $O(n)$ calls to the decision procedure for Σ for formulas of size at most m we can convert A into an automaton where all infeasiable transitions have been removed. The bound on this step is $O(n \cdot f(m))$.

Next, assume that all transitions in A are feasible and let G be the set of all guards that occur in A and let $\Gamma = \{c_\epsilon\} \cup \{g_\gamma \mid \gamma \in G\}$ be an alphabet, where each g_γ is a binary function symbol and c_ϵ is a constant. Decide emptiness of A as a finite tree automaton (FTA) over Γ where each transition (q, γ, q_1, q_2) corresponds to the FTA transition $q \to g_\gamma(q_1, q_2)$ and each leaf state q corresponds to the transition $q \to c_\epsilon$. If A FTA-recognizes a Γ-term s, then $\mathscr{L}(s) \subseteq \mathscr{L}(A)$, where $\mathscr{L}(c_\epsilon) = \{\epsilon\}$ and $\mathscr{L}(g_\gamma(s_1, s_2)) = \{\langle a, t_1, t_2\rangle \mid a \in [\![\gamma]\!]^{\mathscr{B}}, t_1 \in \mathscr{L}(s_1), t_2 \in \mathscr{L}(s_2)\}$, and $\mathscr{L}(s) \neq \emptyset$ because all guards are feasible. The other direction is immediate: if $t \in \mathscr{L}(A)$ then A FTA-recognizes a Γ-term s such that $t \in \mathscr{L}(s)$. Thus, $[\![\varphi]\!] \neq \emptyset$ iff A FTA-recognizes some Γ-term. Since checking non-emptiness a tree automata is linear in their size, the bound follows. ⊠

Theorem 2. $\mathbf{STA}(\Sigma)$ *is an effective Boolean algebra. If Σ is decidable then so is* $\mathbf{STA}(\Sigma)$*. If the decision complexity of Σ is $O(f(n))$ then the decision complexity of* $\mathbf{STA}(\Sigma)$ *is $O(2^n \cdot f(2^n))$.*

Proof. Given $t \in \mathscr{T}(\mathscr{B})$ we can effectively construct an STA $\hat{t}$ such that $[\![\hat{t}]\!] = \{t\}$. The first statement therefore follows from Lemmas 3 and 4, and Theorem 1.

We now prove decidability and the complexity bound together. Assume we are given an $\varphi \in \mathbf{STA}(\Sigma)$, where φ is of the form $\psi \wedge \psi'$, $\psi \vee \psi'$, $\neg\psi$, t, or A, for symbolic automaton A and $\psi, \psi' \in \mathbf{STA}(\Sigma)$. We can bring φ into negation normal φ_1 form by pushing negations over conjunctions and disjunctions. The size increase of φ is at most n. We can remove negations from φ_1 to form φ_2, by replacing each occurrence $\neg A$ by $\complement(\wp(A))$ in φ. The size increase is $O(\Sigma_{\neg A \in \varphi} 2^{|A|})$, where $\neg A \in \varphi$ means $\neg A$ occurs in φ. We can remove conjunctions and disjunctions from φ_2 by replacing $A \vee B$ by $A + B$ and $A \wedge B$ by $A \times B$. Let the resulting STA be B.

The size of B is at most $\Pi_{A \in \varphi} 2^{|A|} = 2^{\Sigma_{A \in \varphi} |A|} = O(2^n)$. It follows from Definition 2 that $\mathscr{L}(B) = \mathscr{L}(B_1)$. Now check emptiness of B_1. The complexity bound follows from Lemma 5. ⊠

In our implementation [11] the algorithms for product and complement use depth first search, where infeasible guards and unreachable states are never included, e.g., in Definition 6 the product transition $\rho_1 \times \rho_2$ is never added if $\varphi_1 \wedge \varphi_2$ is unsatisfiable. Moreover, all the algorithms are extended to work with arbitrary alphabets, not just binary trees. The product is currently implemented by using top-down (starting from the roots) style of traversal. These algorithms are instrumental in the applications discussed in [10].

Example 5. Complementation and determinization of STAs is needed in the following analysis scenario. Consider a symbolic tree transducer T that is an HTML sanitizer [10, Figure 2]. It is a program that traverses an input HTML document and modifies its nodes by removing attributes and values that may cause malicious code to be executed. The description of what is a safe HTML document can be given by an STA A, in general A may be nondeterministic. The sanitizer T is *secure for* A if for all valid input trees t, $T(t) \in \mathscr{L}(A)$, where a tree t is valid if it is accepted by the domain STA $dom(T)$ of T. The *inverse image* of T with respect to an STA B, $InvImg(T, B)$ is also an STA that can be computed effectively from T and B, and is such that

$$\mathscr{L}(InvImg(T, B)) = \{t \mid t \in \mathscr{L}(dom(T)), T(t) \in \mathscr{L}(B)\}.$$

We have that

$$\begin{array}{ll} & \mathscr{L}(InvImg(T, \complement(\wp(A)))) \\ = & \{t \mid t \in \mathscr{L}(dom(T)), T(t) \in \mathscr{L}(\complement(\wp(A)))\} \\ = & \{t \mid t \in \mathscr{L}(dom(T)), T(t) \in \complement(\mathscr{L}(A))\} \\ = & \{t \mid t \in \mathscr{L}(dom(T)), T(t) \notin \mathscr{L}(A)\} \end{array}$$

So T is secure for A iff $\mathscr{L}(InvImg(T, \complement(\wp(A)))) = \emptyset$. ⊠

Another basic decision problem of STAs is the *membership* problem: given $t \in \mathscr{T}(\mathscr{B})$ and STA A, decide if $t \in \mathscr{L}(A)$. An equivalent formulation is $\mathscr{L}(\hat{t}) \cap \mathscr{L}(A) \neq \emptyset$, i.e., $\hat{t} \times A \not\equiv \bot_{\mathbf{STA}(\Sigma)}$. A more direct and computationally less expensive method (that avoids the product construct) is to decide $Q_A(t) \cap Q_A^{\mathrm{R}} \neq \emptyset$. The definition of $Q_A(t)$ can be given by induction over trees:

$$\begin{array}{l} Q_A(\epsilon) \stackrel{\text{def}}{=} Q_A^{\mathrm{L}} \\ Q_A(\langle a, t_1, t_2\rangle) \stackrel{\text{def}}{=} \\ \quad \bigcup_{\rho \in \Delta_A} \{lhs(\rho) \mid rhs(\rho) \in Q_A(t_1) \times Q_A(t_2), a \in [\![\gamma(\rho)]\!]\} \end{array}$$

When formulated as a depth first search procedure, the membership problem reduces to a linear number of membership problems in the label theory.

Theorem 3. *The membership problem for STAs, given $t \in \mathscr{T}(\mathscr{B})$ and STA A, has complexity $O((|t| + |\Delta_A|) \cdot f(m))$ where $f(m)$ is the complexity of the membership problem of Σ_A for instances of size m.*

5. Related work

Our interest in automata and transducers with *symbolic* alphabets originally surfaced in the context of security analysis of string sanitization routines [6]. Sanitizers transform untrusted data to trusted data as a first line of defense against cross site scripting (XSS) attacks in web browsers. Symbolic transducers were generalized to symbolic *tree* transducers (STTs) in [12]. Boolean closure operations of STAs were initially studied in [13] where preliminary results corresponding to Theorem 1 and Theorem 2 are stated as [13, Theorem 1] and [13, Theorem 3], respectively. Explicit complexity bounds are not investigated in [13]. Unfortunately, some of the results in [13] are unclear and wrong, which is also noted in [14]. In particular, closure under complement is unclear because unbounded tree ranks are allowed in tree languages in [13]. Also, closure of STTs under composition [13, Theorem 5] is wrong. Properties of STTs are studied and analyzed further in [14].

Analysis of string sanitizers is lifted to trees by use of symbolic tree transducers with *regular lookahead*, that are introduced in the context of the Fast project [10, 11]. Many tree transducer analysis algorithms depend on STA algorithms, some of which require that the STA is deterministic. Some applications are: HTML Sanitization, augmented reality conflict analysis, deforestation, and CSS analysis. An online tutorial is avilable in [11]. The notion of an *alternating symbolic tree automaton* is also introduced in [10]. Alternating STAs arise naturally as domain automata of nonlinear symbolic tree transducers. Alternating STAs must be *normalized* to STAs prior to determinization.

Symbolic generalizations of classical (Rabin-Scott) word automata have been studied and used in various contexts. In the context of *finite automata algorithm* design, use of predicates is mentioned in [15]. In [16] the motivation comes from *computational linguistics*. In [17] a variant of symbolic automata, called $\mathfrak{M}$-automata, are studied from a *model-theoretic* perspective. In [18] the study of SFAs is primarily motivated by *string analysis*. A symbolic extension seems straightforward at first glance, but raises many challenging questions about algorithm design [9, 7].

There has been considerable interest in word automata over infinite alphabets [19], starting with the work on *finite memory automata* [3]. Other automata models over data words are *pebble automata* [20] and *data automata* [21]. Recent work in [22] introduces *automata with group actions* that, using category theory, provides a generalization of finite memory automata to a richer set of alphabet theories.

Generalizations of automata over data words to *tree automata* over data trees has received quite a bit of attention in XML research, with generalizations such as finite-memory tree automata [2], and, more recently, unranked data tree automata [4]. The paper [4] includes an up-to-date overview of the state-of-the-art. Overall, the

extensions are largely orthogonal to STAs and address different problems. Data tree automata in [4] primarily target analysis of satisfiability of *unranked* data tree automata with respect to constraints that can, e.g., reflect XML integrity, consistency and denial constraints, while providing an NP upper bound [4, Theorem 4.1] by reduction to integer linear programming. STAs on the other hand are defined over a *ranked* alphabet, and are not tied to any particular label theory, e.g., $\mathscr{B}$ could be $F \times \mathbb{Z}$ where F is a finite labeling alphabet as in data tree automata. To the best of our knowledge, STAs are the first extension of tree automata over infinite alphabets that itself forms an effective Boolean algebra, in particular, unlike the extensions in [2, 4], STAs are closed under *complement*. Moreover, this holds uniformly for all label theories that are Boolean algebras.

MONA [23, 24] provides decision procedures for several varieties of monadic second-order logic and supports encoding and reasoning over trees. MONA uses multi-terminal BDDs for encoding of transitions. A core difference compared to **STA**(Σ) is that the label theory Σ is not modular but an integrated part of the MTBDD encoding. VATA is a tool that enables tree automata analysis [25]. Similar to MONA, transitions are represented symbolically using BDDs and VATA is limited to nondeterministic tree automata over finite (although large) alphabets.

References

[1] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree automata techniques and applications, Available on: `http://www.grappa.univ-lille3.fr/tata`, release October, 12th 2007 (2007).

[2] M. Kaminski, T. Tan, Tree automata over infinite alphabets, in: Pillars of Computer Science, Springer, 2008, pp. 386–423.

[3] M. Kaminski, N. Francez, Finite-memory automata, in: 31st Annual Symposium on Foundations of Computer Science (FOCS 1990), Vol. 2, IEEE, 1990, pp. 683–688.

[4] C. David, L. Libkin, T. Tan, Efficient reasoning about data trees via integer linear programming, in: ICDT'11, ACM, 2011, pp. 18–29.

[5] M. Veanes, N. Bjørner, L. de Moura, Symbolic automata constraint solving, in: C. Fermüller, A. Voronkov (Eds.), LPAR-17, Vol. 6397 of LNCS, Springer, 2010, pp. 640–654.

[6] M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, N. Bjorner, Symbolic finite state transducers: Algorithms and applications, in: Proceedings of the Symposium on Principles of Programming Languages (POPL'12), 2012.

[7] L. D'Antoni, M. Veanes, Minimization of symbolic automata, in: POPL'14, ACM, 2014, pp. 541–553.

[8] F. Gécseg, M. Steinby, Tree Automata, Akadémiai Kiadó, Budapest, 1984.

[9] P. Hooimeijer, M. Veanes, An evaluation of automata algorithms for string analysis, in: VMCAI'11, LNCS, Springer, 2011.

[10] L. D'Antoni, M. Veanes, B. Livshits, D. Molnar, Fast: a transducer-based language for tree manipulation, in: PLDI'14, ACM, 2014.

[11] FAST tutorial, `http://www.rise4fun.com/Fast/tutorial`.

[12] M. Veanes, N. Bjørner, Symbolic tree transducers, in: Perspectives of System Informatics (PSI'11), 2011.

[13] M. Veanes, N. Bjørner, Foundations of finite symbolic tree transducers, in: Y. Gurevich (Ed.), Bulletin of the EATCS, The Logic in Computer Science Column, no. 105, 2011, pp. 141–173.

[14] Z. Fülöp, H. Vogler, Forward and backward application of symbolic tree transducers, Acta Informatica 51 (5) (2014) 297–325.

[15] B. W. Watson, Cambridge U. Press, 1999, Ch. Implementing and using finite automata toolkits, pp. 19–36.

[16] G. V. Noord, D. Gerdemann, Finite state transducers with predicates and identities, Grammars 4 (2001) 263–286.

[17] A. Bés, An application of the Feferman-Vaught theorem to automata and logics for words over an infinite alphabet, Logical Methods in Computer Science 4 (2008) 1–23.

[18] M. Veanes, P. de Halleux, N. Tillmann, Rex: Symbolic Regular Expression Explorer, in: Third International Conference on Software Testing, Verification and Validation (ICST'10), IEEE, 2010.

[19] L. Segoufin, Automata and logics for words and trees over an infinite alphabet, in: Z. Ésik (Ed.), CSL, Vol. 4207 of LNCS, 2006, pp. 41–57.

[20] F. Neven, T. Schwentick, V. Vianu, Finite state machines for strings over infinite alphabets, ACM Trans. CL 5 (2004) 403–435.

[21] M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, C. David, Two-variable logic on words with data, in: LICS, IEEE, 06, pp. 7–16.

[22] M. Bojańczyk, B. Klin, S. Lasota, Automata with group actions, in: 26th Annual IEEE Symposium on Logic in Computer Science, IEEE, 2011, pp. 355–364.

[23] J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, A. Sandholm, Mona: Monadic second-order logic in practice, in: Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95, LNCS 1019, 1995.

[24] N. Klarlund, A. Møller, M. I. Schwartzbach, MONA implementation secrets, International Journal of Foundations of Computer Science 13 (4) (2002) 571–586.

[25] O. Lengal, J. Šimáček, T. Vojnar, Vata: A library for efficient manipulation of non-deterministic tree automata, in: TACAS'12, Vol. 7214 of LNCS, Springer, 2012, pp. 79–94.