Minimization of Symbolic Tree Automata

Loris D'Antoni
University of Wisconsin — Madison
loris@cs.wisc.edu

Margus Veanes
Microsoft Research
margus@microsoft.com

Abstract

Symbolic tree automata allow transitions to carry predicates over rich alphabet theories, such as linear arithmetic, and therefore extend finite tree automata to operate over infinite alphabets, such as the set of rational numbers. Existing tree automata algorithms rely on the alphabet being finite, and generalizing them to the symbolic setting is not a trivial task.

In this paper we study the problem of minimizing symbolic tree automata. First, we formally define and prove the basic properties of minimality in the symbolic setting. Second, we lift existing minimization algorithms to symbolic tree automata. Third, we present a new algorithm based on the following idea: the problem of minimizing symbolic tree automata can be reduced to the problem of minimizing symbolic (string) automata by encoding the tree structure as part of the alphabet theory. We implement and evaluate all our algorithms against existing implementations and show that the symbolic algorithms scale to large alphabets and can minimize automata over complex alphabet theories.

1. Introduction

Tree automata are used in a variety of applications in software engineering, including analysis of XML programs [24], software verification [2], and natural language processing [27]. While tree automata are of immense practical use, they suffer from a major drawback: in the most common forms they can only handle finite and small alphabets.

Symbolic automata allow transitions to carry predicates over a specified alphabet theory, such as linear arithmetic, and therefore extend finite tree automata to operate over infinite alphabets, such as the set of rational numbers [14, 16]. Symbolic automata are therefore more general and succinct than their finite-alphabet counterparts. Traditional algorithms for finite string and tree automata do not immediately generalize to the symbolic setting, making the design of algorithms for symbolic automata challenging. A notable example appears in [15]: while allowing finite state automata transitions to read multiple adjacent inputs does not add expressiveness, in the symbolic case this extension makes problems such as checking equivalence undecidable.

Symbolic tree automata (s-TA) are closed under Boolean operations and enjoy decidable equivalence if the alphabet theory is decidable and forms a Boolean algebra [32]. s-TAs have been used in combination with symbolic tree transducers to analyze complex tree-manipulating programs such as HTML sanitizers and augmented reality taggers. In these applications keeping the automata "small" is crucial for scalability, but to the best of our knowledge, no algorithms have been proposed to minimize s-TAs. Minimization of symbolic *bottom-up* tree automata is the topic of this paper.

Minimization of tree automata. Minimization of tree automata has been studied extensively [5, 12, 19, 22], although not as thoroughly as minimization of finite string automata. Recently the topic has received new interest [3]. For a deterministic bottom-up tree automaton A to be minimal, all distinct states p and q of A have to be *distinguishable*: for some term $t(x)$ with a single variable x, and two trees t_p and t_q accepted by the states p and q respectively, the tree $t(t_p)$ is accepted by A if and only if the tree $t(t_q)$ is not accepted by A. Most minimization algorithms start with an under-approximation of the set of distinguishable states and iteratively refine it using a fix-point computation.[1] In the case of finite alphabets, minimization algorithms refine the under-approximation by looping over the alphabet and checking whether on the same symbol two transitions lead to two distinguishable states. Since symbolic finite automata operate over infinite alphabets, this operation cannot be performed.

One solution is to *finitize* the alphabet and then use existing minimization algorithms. This is done by treating the set of all inequivalent satisfiable Boolean combinations of the predicates appearing in the automaton as the alphabet. Unfortunately, this *finitization* procedure is exponential in the number of transitions. Efficient algorithms for minimizing symbolic (string) automata that avoid the alphabet finitization are proposed in [14], but they do not generalize to symbolic tree automata.

Minimization of symbolic tree automata. We propose two new algorithms for minimizing symbolic tree automata and prove their correctness.

Our first algorithm, STAPart, is a symbolic extension of the algorithm presented in [8] for minimizing tree automata. The algorithm builds on the following property: if two states q and q' are distinguishable, and there exists a symbol a and transitions $(q_1, \ldots, q_i, \ldots, q_n) \xrightarrow{a} q$ and $(q_1, \ldots, q'_i, \ldots, q_n) \xrightarrow{a} q'$, then q_i and q'_i are distinguishable. This notion nicely translates to the symbolic setting: if two states q and q' are distinguishable, and there exist transitions $(q_1, \ldots, q_i, \ldots, q_n) \xrightarrow{\varphi} q$ and $(q_1, \ldots, q'_i, \ldots, q_n) \xrightarrow{\psi} q'$, such that $\mathit{Sat}(\varphi \wedge \psi)$, then q_i and q'_i are distinguishable The algorithm uses a fixpoint computation based on this definition to compute all the pairs of distinguishable states. Like most algorithms for symbolic automata, the algo-

[1] A notable exception is Brzozowski's algorithm, which does not explicitly build the set of distinguishable states [7].

CONF 'yy, Month d–d, 20yy, City, ST, Country.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

rithm requires that checking satisfiability of the predicate $\varphi \wedge \psi$ is decidable. Practically, this is checked using a decidable effective Boolean algebra as a solver. The algorithm operates on *partial* automata and avoids computing the costly automata completion. The algorithm is based on the idea that only a few transitions added by the completion operation are used when refining the set of distinguishable states.

Our second algorithm, SFARed, is based on the following idea: the problem of minimizing s-TAs can be reduced to the problem of minimizing symbolic finite automata over strings. This intuition is inspired by Abdulla et al. [2], where a similar notion is used to compute bisimulations of non-deterministic tree automata. Thanks to this reduction we can directly use the efficient s-FA minimization algorithm $Min^{\mathbf{N}}_{\text{SFA}}$ presented in [14]. A novel aspect of this reduction is that the reduced s-FA operates over a different alphabet theory from the one used by the input s-TA. However, we show that the resulting alphabet theory is not only decidable, but can also be efficiently implemented with an off-the-shelf SMT [18] solver. On the other hand, if one did not use a symbolic representation of the reduced alphabet the reduction would cause an exponential blow-up. What is remarkable is that the separation of concerns between the alphabet theory and the automata structure allows us to use symbolic automata not only to model string and tree languages over complex alphabets, but also to design new elegant algorithms.

We compare the performance of the two algorithms using: 1) a large set of tree automata over small finite alphabets taken from the VATA library for non-deterministic tree automata [26]; 2) a set of s-TAs over large finite alphabets aimed at showing the exponential blow-up caused by the alphabet finitization; and 3) a set of randomly generated s-TAs over a complex alphabet theory. In experiments 1 and 2 we also compare the performance of our algorithms against the Lethal library [11] and the tree automata minimization implementation described in Carrasco et al. [8]. Our experiments show that our algorithms are comparable to state-of-the-art finite-alphabet implementations in the case of small finite alphabets and scale to large alphabets that existing tree automata implementations cannot handle. In the case of complex alphabet theories our two algorithms have comparable performances.

Contributions. Our contributions are the following.

- A formal study of the notion of minimality of symbolic tree automata (s-TAs) (§ 3);
- A new algorithm for minimizing s-TAs, STAPart, which is based on existing algorithms for minimizing tree automata over finite alphabets (§ 4);
- A new efficient algorithm, SFARed, based on a reduction to symbolic (string) automata minimization (§ 5);
- An implementation and a comprehensive evaluation of the algorithms over a variety of benchmarks (§ 6).

2. Motivating Example: Analysis of Tree-Manipulating Programs

In this section we introduce s-TAs using informal examples and show how minimization can be used to speed-up the analysis of tree-manipulating programs written in FAST, a language that has been used to prove properties of HTML sanitizers, functional programs over trees, and augmented reality taggers [16]. In all these applications the alphabets are infinite and using symbolic tree automata is therefore necessary. To analyze programs, FAST uses a variety of symbolic tree automata and transducer algorithms including intersection, complement, functional composition, and domain automaton computation. When many of these algorithms are applied in sequence, the s-TAs tend to become too large, causing the running time to be impractical. The minimization techniques proposed in this paper are vital in keeping the s-TAs small and making the analyses scalable. The following example adapted from the case study in [16, Fig. 2] and it is used throughout the paper to illustrate definitions and concepts.[2]

2.1 Symbolic Tree Automata in FAST

Consider the following algebraic datatype `Html` that describes (a simplified form of) abstract syntax of HTML documents where `str` is a predefined type for strings:

```
Html   = empty | node str Attrs Html Html
Attrs  = none  | attr str Values Attrs
Values = nil   | cons str Values
```

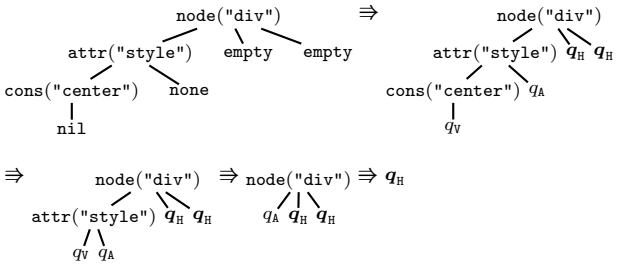
We group all the *non-tree-type* arguments together into a single *label* type ℓ as a disjoint union of named tuples:

$$\ell = \texttt{empty|none|nil|node str|attr str|cons str}$$

and define testers over ℓ, e.g., $\texttt{IsNode}(\texttt{node}(\texttt{"ab"}))$ is true. Let `HTML` be an STA that accepts the set of trees representing correct HTML documents using the following transitions,

$$\Delta_{\texttt{HTML}} = \{() \xrightarrow{\texttt{IsNil}} q_{\texttt{V}}, () \xrightarrow{\texttt{IsNone}} q_{\texttt{A}}, () \xrightarrow{\texttt{IsEmpty}} \boldsymbol{q}_{\texttt{H}},$$
$$(q_{\texttt{V}}) \xrightarrow{\texttt{IsCons}} q_{\texttt{V}}, (q_{\texttt{V}}, q_{\texttt{A}}) \xrightarrow{\texttt{IsAttr}} q_{\texttt{A}}, (q_{\texttt{A}}, \boldsymbol{q}_{\texttt{H}}, \boldsymbol{q}_{\texttt{H}}) \xrightarrow{\texttt{IsNode}} \boldsymbol{q}_{\texttt{H}}\}$$

where *states* $q_{\texttt{V}}, q_{\texttt{A}}, \boldsymbol{q}_{\texttt{H}}$ represent types `Values`, `Attrs`, `Html` with $\boldsymbol{q}_{\texttt{H}}$ as a *final* state. Intuitively, $\Delta_{\texttt{HTML}}$ is a conditional term rewrite system. A tree t is *accepted* by `HTML` (in symbols $t \in \mathcal{L}(\texttt{HTML})$) iff t can be reduced to $\boldsymbol{q}_{\texttt{H}}$, e.g.,



2.2 The Importance of Minimization in FAST

FAST allows the user to define arbitrary s-TAs and to compose them. This can be seen in action when writing a FAST program that analyzes an HTML sanitizer. An HTML *sanitizer* is a program that parses rich HTML markup and removes executable content. When analyzing an HTML sanitizer S, we check that S can only produce safe HTML trees (for example, those that do not contain script nodes). To check this property the user can define an s-TA `IsSafe` that accepts the set of all *safe* HTML trees that S should output. Keeping the s-TA `IsSafe` minimal is crucial for scalability. We show how a typical version of the s-TA `IsSafe` can be large in practice. A simple set of safe HTML trees is defined by the s-TA $\texttt{IsSafe}_1$ which accepts all the trees that do not contain a `"script"` node. The s-TA $\texttt{IsSafe}_1$ has a single state q (that is also final) and the transitions:

$$\{() \xrightarrow{\top} q, \quad (q) \xrightarrow{\top} q, \quad (q,q) \xrightarrow{\top} q, \quad (q,q,q) \xrightarrow{\textit{safe}} q\}$$

where *safe* $\stackrel{\text{def}}{=} \lambda x{:}\ell.(\texttt{IsNode}(x) \wedge x[1] \neq \texttt{"script"})$ and $\top \stackrel{\text{def}}{=} \lambda x{:}\ell.true$. As new possible vulnerabilities arise, the user will create new s-TAs $\texttt{IsSafe}_i$, $1 < i \leq n$, aimed at further restricting the set of HTML trees. The final definition of `IsSafe` is `HTML` $\cap$

[2] The reader may also consult this example using the online version of FAST at `http://rise4fun.com/Fast/Lxu`.

$\bigcap_{i=1}^{n}$ `IsSafe`$_i$. In a typical setting, n can be up to 20. Because of the repeated intersections, the s-TA `IsSafe` can quickly grow in size, making the analysis intractable. Minimization drastically reduces the size.

3. Symbolic Tree Automata

In this section we formally define symbolic (bottom-up) tree automata and their theory of minimality.

3.1 Effective Boolean Algebras

We use effective Boolean algebras in place of concrete alphabets. An *effective Boolean algebra* $\mathcal{A}$ is a tuple $(U, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$ where U is a recursively enumerable (r.e.) set called the *universe* of $\mathcal{A}$. Ψ is a r.e. set of *predicates* closed under the Boolean connectives, $\vee, \wedge : \Psi \times \Psi \to \Psi$, $\neg : \Psi \to \Psi$, and $\bot, \top \in \Psi$. The *denotation function* $[\![\_]\!] : \Psi \to 2^U$ is r.e. and is such that, $[\![\bot]\!] = \emptyset$, $[\![\top]\!] = U$, for all $\varphi, \psi \in \Psi$, $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$, $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$, and $[\![\neg\varphi]\!] = U \setminus [\![\varphi]\!]$.[3] For $\varphi \in \Psi$, we write *Sat*(φ) when $[\![\varphi]\!] \neq \emptyset$ and say that φ is *satisfiable*. The algebra $\mathcal{A}$ is *decidable* if *Sat* is decidable. We say that $\mathcal{A}$ is *infinite* if U is infinite. In practice, an (effective) Boolean algebra can be implemented as an API with corresponding methods implementing the operations. The following are examples of Boolean algebras that we use in practice.

Example 1. We illustrate use of an SMT solver as an effective Boolean algebra. Let *SMT*$_\tau$ denote $(U, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$, where τ is a fixed type, U is the set of all elements of type τ. Ψ is the set of all quantifier free formulas of the solver containing a single uninterpreted constant (or variable) $x : \tau$. $\top$ is $x = x$ and $\bot$ is $x \neq x$. The Boolean operations are the corresponding logical connectives of the solver. The interpretation function $[\![\varphi]\!]$ is defined using satisfiability checking and model generation features of the solver: test if φ is satisfiable, if not then terminate else construct a model $x \mapsto v \models \varphi$, yield v, and continue with $\varphi \wedge x \neq v$. ⊠

Example 2. *The BDD algebra* $\mathcal{B} = (\mathbb{N}, \Psi, [\![\_]\!], \bot, \top, |, \&, \overline{\cdot})$ has the set of natural numbers $\mathbb{N}$ as its universe and Ψ is the Boolean closure of BDDs [6] β_i s.t. $[\![\beta_i]\!] = \{n | \text{“the } i\text{'th bit of } n \text{ is 1”}\}$. $\beta_3 \& \overline{\beta_0}$ means “*if bit*$_3$=1 *then (if bit*$_0$=1 *then* $\bot$ *else* $\top$*) else* $\bot$” that denotes the set of numbers matching the binary bitpattern $*\cdots*1{*}{*}0$, e.g., $8, 26 \in [\![\beta_3 \& \overline{\beta_0}]\!]$.[4] We use $\mathcal{B}$ in Example 4. ⊠

3.2 Symbolic Tree Automata

A *signature* Σ is an r.e. set of ranked function symbols. For $f \in \Sigma$ we let $\natural(f)$ denote its rank. The set of all symbols of rank k in Σ is denoted by $\Sigma(k)$. The set of all terms over Σ is $\mathcal{T}(\Sigma)$. We assume that $\Sigma(0) \neq \emptyset$ so that $\mathcal{T}(\Sigma) \neq \emptyset$. Given a set S and rank $k \geq 0$, we write S^k for the cross product $\prod_{i=1}^{k} S$ where $S^0 \stackrel{\text{def}}{=} \{()\}$.

Definition 1. A *symbolic tree automaton* (*s-TA*) M is a tuple $(\mathcal{A}, \Gamma, Q, F, \Delta)$ where $\mathcal{A}$ is an effective Boolean algebra called the *label alphabet*, Γ is a finite signature of *constructors*, Q is a finite set of *states*, $F \subseteq Q$ is the set of *final states*, and $\Delta \subseteq \bigcup_{k \geq 0} \Gamma(k) \times Q^k \times \Psi_{\mathcal{A}} \times Q$ is a finite set of *transitions*.

We fix $M = (\mathcal{A}, \Gamma, Q, F, \Delta)$, $\mathcal{A} = (U, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$.

Definition 2. M is *deterministic* if for all $(g_1, \bar{p}_1, \psi_1, q_1)$ and $(g_2, \bar{p}_2, \psi_2, q_2)$ in Δ, if $g_1 = g_2$ and $\bar{p}_1 = \bar{p}_2$ and *Sat*$(\psi_1 \wedge \psi_2)$ then $q_1 = q_2$.

[3] The underlying Boolean algebra of $\mathcal{A}$ corresponds to the *field of sets* $(U, \{[\![\psi]\!] \mid \psi \in \Psi\})$ where elements of U are called *points*, using the *Representation Theorem of Boolean Algebras* (cf. [9, Proposition 1.4.4]).

[4] The underlying Boolean algebra of the BDD algebra $\mathcal{B}$ is isomorphic to the *countable atomless Boolean algebra* (cf. [9, Proposition 1.4.5]).

In this paper we are only concerned with deterministic s-TAs. We write $g(\bar{p}) \xrightarrow{\psi}_M q$, or $g(\bar{p}) \xrightarrow{\psi} q$ when M is clear, for a transition $(g, \bar{p}, \psi, q) \in \Delta$ and say that $(g, \bar{p}, \psi, q)$ has *rank* $\natural(g)$; g is the *constructor*, $\bar{p}$ the *source*, q the *target*, ψ the *guard*, and $g(\bar{p})$ the *trigger* of $(g, \bar{p}, \psi, q)$. We further abbreviate $g(\bar{p}) \xrightarrow{\psi} q$ by $\bar{p} \xrightarrow{\psi} q$ when g is the only constructor in $\Gamma(\natural(g))$.

Example 3. Consider the type ℓ from Section 2 and let terms be in the Boolean algebra *SMT*$_\ell$ from Example 1 by using corresponding datatype constructors. Let `HTML` = (*SMT*$_\ell$, $\{f_0, f_1, f_2, f_3\}$, $\{q_{\mathtt{V}}, q_{\mathtt{A}}, \boldsymbol{q}_{\mathtt{H}}\}$, $\{\boldsymbol{q}_{\mathtt{H}}\}$, $\Delta_{\mathtt{HTML}}$) where f_i has rank i. `HTML` is deterministic because all transitions have disjoint guards. ⊠

Definition 3. M is *complete* if for all $g \in \Gamma$, $a \in U$, and $\bar{p} \in Q^{\natural(g)}$, there is $g(\bar{p}) \xrightarrow{\psi} q$ with $a \in [\![\psi]\!]$. Else, M is *partial*.

Completeness ensures that for all labels and all constructors there exists a transition from each state combination. Let $q^\bot \notin Q$ be a new *sink* state and let $Q^\bot = Q \cup \{q^\bot\}$.

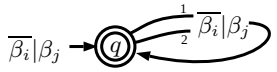
Definition 4. If M is partial, then the *completion of* M is the s-TA $M^\bot \stackrel{\text{def}}{=} (\mathcal{A}, \Gamma, Q^\bot, F, \Delta \cup \{(g, \bar{p}, \nu_{g(\bar{p})}, q^\bot) \mid g \in \Gamma, \bar{p} \in (Q^\bot)^{\natural(g)}, \mathit{Sat}(\nu_{g(\bar{p})})\})$ where $\nu_{g(\bar{p})} \stackrel{\text{def}}{=} \bigwedge_{\exists q : (\bar{p}, g, \psi, q) \in \Delta} \neg\psi$.

Definition 5. We define $\Sigma_M \stackrel{\text{def}}{=} \Gamma \times U$, and use $g[a]$ to denote the element $(g, a) \in \Sigma_M$, and let $\natural(g[a]) \stackrel{\text{def}}{=} \natural(g)$.

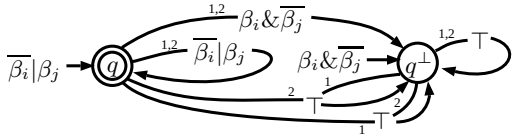
We write Σ for Σ_M when M is clear. We use the notions from [9] regarding *universe*, *interpretation*, and *model for* Σ (or Σ-*model*). We view a deterministic and complete s-TA M as a Σ-model $\mathfrak{M}$ whose universe is Q and whose interpretation for $g[a] \in \Sigma$ is the function $g[a]^{\mathfrak{M}}$ that maps $\bar{p} \in Q^{\natural(g)}$ to the state $q \in Q$ such that $g(\bar{p}) \xrightarrow{\psi}_M q$ and $a \in [\![\psi]\!]$. For deterministic *partial* M we let the Σ-model of M be the Σ-model of $M^\bot$.

Definition 6. Let M be deterministic. For $q \in Q$, the *down-language of* q *in* M is $\mathcal{L}^{\downarrow}_M(q) \stackrel{\text{def}}{=} \{t \in \mathcal{T}(\Sigma) \mid t^{\mathfrak{M}} = q\}$. The *language of* M is the set $\mathcal{L}(M) \stackrel{\text{def}}{=} \bigcup_{q \in F} \mathcal{L}^{\downarrow}_M(q)$. Let N be a deterministic s-TA. Then M is *equivalent* with N, $M \simeq N$, if $\mathcal{L}(M) = \mathcal{L}(N)$ and $\mathcal{A}_M = \mathcal{A}_N$ and $\Gamma_M = \Gamma_N$.

Example 4. We use the BDD algebra $\mathcal{B}$ from Example 2 and let $\Gamma = \{c, f\}$, $\natural(c) = 0$, $\natural(f) = 2$. Let $M_{i,j}$ be the s-TA



$M_{i,j} = (\mathcal{B}, \{c, f\}, \{q\}, \{q\}, \{c() \xrightarrow{\overline{\beta_i}|\beta_j} q, f(q, q) \xrightarrow{\overline{\beta_i}|\beta_j} q\})$. The figure uses 1 and 2 to identify the respective arguments of the constructor $f[a] \in \Sigma_M$. Recall from Example 2 that the predicate $\overline{\beta_i}|\beta_j$ says that if the i'th bit of the label is 1 then the j'th bit must also be 1. $M_{i,j}$ accepts a tree iff all of its labels satisfy this condition. To be concrete let $t = f[8](c[4], f[6](c[3], c[0]))$. For example $t \in \mathcal{L}(M_{0,1})$ but $t \notin \mathcal{L}(M_{1,0})$ because of label 6. The completion $M^\bot_{i,j}$ of $M_{i,j}$ looks as follows



$M^\bot_{i,j}$ can be used to construct the s-TA accepting the complement of $\mathcal{L}(M_{i,j})$ by making $q^\bot$ final and q nonfinal.[5] ⊠

[5] The example arises when mapping S2S *subset* relations to tree automata [21].

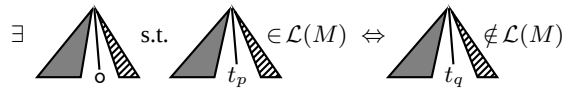


Figure 1. Distinguishability of states p and q in M.

3.3 Minimality of Symbolic Tree Automata

Our notion of minimality is based on the generalization of the *Myhill-Nerode theorem* to trees [13, Section 1.5]. The following notion of context is central to our definition of minimality. Recall Definition 5.

Definition 7. Let $\mathtt{o}$ be a fixed constant, $\mathtt{o} \notin \Sigma$. A Σ-*context* is a tree in $\mathcal{T}(\Sigma \cup \{\mathtt{o}\})$ with exactly one occurrence of $\mathtt{o}$.

If $t(\mathtt{o})$ is a Σ-context and u is a term then $t(u)$ is the term obtained by substituting $\mathtt{o}$ in $t(\mathtt{o})$ by u.

Definition 8. A state q is *accessible* if $\mathcal{L}^{\downarrow}_M(q) \neq \emptyset$; t_q denotes a fixed term in $\mathcal{L}^{\downarrow}_M(q)$. M is *reduced* if all states in Q are accessible. A state q is *useful* if q is accessible and there exists a context $t(\mathtt{o})$ such that $t(t_q) \in \mathcal{L}(M)$. M is *clean* if all states in Q are useful.

Useless states can be eliminated using standard algorithms. The prerequisite is that there are no transitions with unsatisfiable guards, which requires use of $\mathcal{A}$. Observe that if M is partial then in $M^{\perp}$ the sink state $q^{\perp}$ is useless although it is accessible. Minimality uses the following context based indistiguishabililty relation.

Definition 9. For all $u, v \in \mathcal{T}(\Sigma)$, define $u \equiv_{\mathcal{L}(M)} v$ iff for all Σ-contexts $t(\mathtt{o})$: $t(u) \in \mathcal{L}(M) \Leftrightarrow t(v) \in \mathcal{L}(M)$.

Definition 10. Assume M is deterministic and reduced. For $p, q \in Q$ define $p \equiv_M q$ iff $t_p \equiv_{\mathcal{L}(M)} t_q$; we say that p and q are *distinguishable* in M when $p \not\equiv_M q$.

The definition of $\equiv_M$ is well-defined because M is reduced (t_p and t_q exist) and defines an equivalence relation over Q because M is deterministic. Distinguishability is illustrated in Figure 1. We write $\equiv$ for $\equiv_M$ if M is clear.

Definition 11. M is *normalized* if it has no two distinct transitions with equal triggers and targets.

Normalize takes a set T of transitions and combines any distinct $(g, \bar{p}, \varphi, q)$ and $(g, \bar{p}, \psi, q)$ in T into $(g, \bar{p}, \varphi \vee \psi, q)$.

Definition 12. $M_{/\equiv} \stackrel{\text{def}}{=} (\mathcal{A}, \Gamma, Q_{/\equiv}, F_{/\equiv}, \mathit{Normalize}(\Delta_{/\equiv}))$ where $\Delta_{/\equiv} = \{(g, \bar{q}_{/\equiv}, \varphi, p_{/\equiv}) \mid (g, \bar{q}, \varphi, p) \in \Delta\}$.

Proposition 1. *Assume M is deterministic and reduced. Then $M \simeq M_{/\equiv}$ and $M_{/\equiv}$ represents the unique (up to isomorphism) Σ-model with the smallest number of states that accepts $\mathcal{L}(M)$.*

A direct proof of Proposition 1 can be given by generalizing Brainerd's theorem [5, Theorem 5.7] to an infinite alphabet Σ. The tree version of Myhill-Nerode theorem [13] can also be generalized to an infinite Σ to prove Proposition 1.

Definition 13. Assume M is deterministic. Then M is *minimal* if $|Q_M| \leq |Q_N|$ and $|\Delta_M| \leq |\Delta_N|$ for all deterministic s-TAs N such that $M \simeq N$.

Observe that if M is partial and clean then the induced Σ-model always has the extra sink state $q^{\perp}$. Then $M^{\perp}_{/\equiv}$ has also the extra state $q^{\perp}_{/\equiv}$, while $M_{/\equiv}$ does not.

Proposition 2. *Assume M is deterministic and clean. Then $M_{/\equiv}$ is minimal.*

Proposition 2 follows from Proposition 1 and normalization. While for finite alphabets the definitions of minimality directly lead to minimization algorithms, this is not true when dealing with infinite alphabets such as $\mathcal{B}$ in Example 2. We discuss our minimization algorithms for symbolic tree automata in the next sections and we will use the following additional definitions. Given a non-empty sequence $\bar{x}$ of length k, x_i stands for the i'th element of $\bar{x}$ for $1 \leq i \leq k$, and, given element y, $\bar{x}_{i:y} \stackrel{\text{def}}{=} (x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_k)$.

Definition 14. Given $k \geq 1$ and two sequences $\bar{x}$ and $\bar{y}$ of length k, and i, $1 \leq i \leq k$, then, $\bar{x} \doteq_i \bar{y}$ is defined as $\bar{x}_{i:y_i} = \bar{y}$ and $x_i \neq y_i$. And $\bar{x} \doteq \bar{y}$ means $\bar{x} \doteq_i \bar{y}$ for some i.

In other words, $\bar{x} \doteq \bar{y}$ means that $\bar{x}$ and $\bar{y}$ are equal in all but one position. For example, $(1, 2, 3, 4) \doteq_3 (1, 2, 5, 4)$.

Definition 15. For a context $t(\mathtt{o})$ let the *o-depth*, $\lfloor t(\mathtt{o}) \rfloor$, of $t(\mathtt{o})$ be the distance of $\mathtt{o}$ in $t(\mathtt{o})$ from the root of $t(\mathtt{o})$. For a pair $(p, q) \in \not\equiv_M$ define $\lfloor p, q \rfloor$ as the smallest $\mathtt{o}$-depth $\lfloor t(\mathtt{o}) \rfloor$ such that $t(t_p) \in \mathcal{L}(M) \Leftrightarrow t(t_q) \notin \mathcal{L}(M)$

E.g., $\lfloor f(t, g(u, \mathtt{o}), v) \rfloor = 2$ and $\lfloor p, q \rfloor = 0$ when $p \in F \Leftrightarrow q \notin F$.

4. Minimization of s-TAs

Completion of s-TAs is expensive and should be avoided if possible. It may be infeasible to represent Δ so that for all ranks and $\bar{q}$ there is a transition from $\bar{q}$. A similar problem arises when minimizing *unranked* tree automata over finite alphabets where the completion is not computable [8]. In the following we introduce an algorithm that can avoid completion by computing an underapproximation of $M^{\perp}$ that suffices for distinguishability.

Let M be deterministic and clean. We let $q^{\perp} \notin Q$ be a new *sink* state. (Recall the definition $\nu_{g(\bar{x})}$ from Definition 4.) We approximate $M^{\perp}$ by using the following set $\Delta^{\perp}$ of transitions. We write $\mathit{lhs}(\Delta)$ for the set of all triggers of transitions in Δ.

$$\Delta^{\perp} \stackrel{\text{def}}{=} \Delta^{\perp}_1 \cup \Delta^{\perp}_2$$
$$\Delta^{\perp}_1 \stackrel{\text{def}}{=} \{g(\bar{x}) \xrightarrow{\nu_{g(\bar{x})}} q^{\perp} \mid g(\bar{x}) \in \mathit{lhs}(\Delta) \wedge \mathit{Sat}(\nu_{g(\bar{x})})\}$$
$$\Delta^{\perp}_2 \stackrel{\text{def}}{=} \{g(\bar{x}_{i:y}) \xrightarrow{\top} q^{\perp} \mid g(\bar{x}) \in \mathit{lhs}(\Delta) \wedge i \in [1..\natural(g)] \wedge y \in Q \wedge x_i \neq y \wedge g(\bar{x}_{i:y}) \notin \mathit{lhs}(\Delta) \wedge (y \in F \Leftrightarrow x_i \in F)\}$$

Observe that if M is complete then $\Delta^{\perp} = \emptyset$. Else, the set $\Delta^{\perp}_1$ contains the completion of all the triggers appearing in Δ, and the set $\Delta^{\perp}_2$ contains transitions to the sink state $q^{\perp}$ from all the new possible triggers that can be generated by replacing exactly one state in an existing trigger. A final state can only be replaced by a final state, and similarly for non-final states. Let $F^c = Q \setminus F$. Treat $\langle p, q \rangle$ below as an unordered pair.

STAPart (Partial s-TA state distinguishability).

Input: deterministic clean s-TA M, *Output*: $\not\equiv_M$.

1. $D := \{\langle p, q \rangle \mid p \in F \wedge q \in F^c\}$; *Frontier* $:= D$;
   if $\Delta^{\perp} \neq \emptyset$ add $\{\langle q, q^{\perp} \rangle \mid q \in Q\}$ to *Frontier*;
2. while *Frontier* $\neq \emptyset$:
   (a) pop $\langle p', q' \rangle$ from *Frontier*;
   (b) for all $g(\bar{p}) \xrightarrow{\varphi} p'$, $g(\bar{q}) \xrightarrow{\psi} q' \in \Delta \cup \Delta^{\perp}$ such that $\bar{p} \doteq_i \bar{q}$ and $\langle p_i, q_i \rangle \notin D$ and $\mathit{Sat}(\varphi \wedge \psi)$:
   add $\langle p_i, q_i \rangle$ to D and push $\langle p_i, q_i \rangle$ to *Frontier*;
3. return D.

If $m = \mathrm{maxrank}(\Gamma)$, the size of $\Delta^{\perp}$ is $O(m \cdot |Q| \cdot |\Delta|)$, which is in sharp contrast to $O(|Q|^m \cdot |\Delta|)$ if Δ would be completed.

Theorem 3. *If $\mathcal{A}$ is decidable, and M is deterministic and clean, then STAPart computes $\not\equiv_M$.*

Proof. First we show termination. The effectiveness of step 2(b) depends on $\mathcal{A}$ being decidable and on the number of choices for given p' and q' in 2(b) being finite. Latter follows from finiteness of $\Delta \cup \Delta^\perp$. Effective construction of $\Delta^\perp$ depends on finiteness of Δ and finiteness of Q and effective use of $\mathcal{A}$ for the construction and satisfiability checking of $\nu_{g(\bar{x})}$ (recall Definition 4). Termination follows because Q is finite and only new elements from $Q \times Q$ that have not been in *Frontier* before are added to *Frontier* and one element is removed from *Frontier* at each iteration of step 2.

Next, we show that D in step 3 equals $\not\equiv_M$. We proceed in two stages: A) we prove the statement under the assumption that M is *complete and reduced*; B) we extend the proof to the case when M is *partial and clean*.

A) Assume M is complete and reduced. (M does not have to be clean.) Note that $\Delta^\perp = \emptyset$. Let D_f be the value of variable D in step 3. We prove that $D_f = \not\equiv_M$. Given $D_n = D$ before executing step 2(b) let D_{n+1} be the value after D has been updated. Initially D_0 is D. Let $L = \mathcal{L}(M)$.

Case $D_f \subseteq \not\equiv_M$**:** We show $D_n \subseteq \not\equiv_M$ by induction over n.
Base case: $D_0 \subseteq \not\equiv_M$ by definition.
Induction case: We show that $D_{n+1} \subseteq \not\equiv_M$. The *IH* is $D_n \subseteq \not\equiv_M$. From 2(a) and IH it follows that $p' \not\equiv_M q'$, so there exists a Σ-context $t(\mathtt{o})$ such that $t(t_{p'}) \in L \Leftrightarrow t(t_{q'}) \notin L$, where $t_{p'}$ and $t_{q'}$ exist because M is reduced. Fix $t(\mathtt{o})$. Assume D_n is updated in Step 2(b). So there are transitions $g(\bar{p}) \xrightarrow{\varphi} p'$ and $g(\bar{q}) \xrightarrow{\psi} q' \in \Delta$ such that $\bar{p} \doteq_i \bar{q}$ and $\langle p_i, q_i\rangle \notin D_n$ and $\mathit{Sat}(\varphi \wedge \psi)$. Fix such transitions. It follows that there exists a Σ-context $g[a](\bar{u} \cdot \mathtt{o} \cdot \bar{v})$ for some $a \in [\![\varphi \wedge \psi]\!]$, and where $\bar{u} = (t_{p_j})_{j=1}^{i-1}$, and $\bar{v} = (t_{p_j})_{j=i+1}^{h(g)}$ since all states in $\bar{p}$ are accessible.
Now let $w(\mathtt{o})$ be the composed Σ-context $t(g[a](\bar{u} \cdot \mathtt{o} \cdot \bar{v}))$. So $w(\mathtt{o})$ is such that, using M as a Σ-model $\mathfrak{M}$, let $f = g[a]$,
$$w(t_p)^{\mathfrak{M}} = t(f(\bar{u} \cdot t_p \cdot \bar{v}))^{\mathfrak{M}} = t(t_{p'})^{\mathfrak{M}} \in F$$
$$\Leftrightarrow \quad w(t_q)^{\mathfrak{M}} = t(f(\bar{u} \cdot t_q \cdot \bar{v}))^{\mathfrak{M}} = t(t_{q'})^{\mathfrak{M}} \notin F$$
So $D_{n+1} \subseteq \not\equiv_M$ where $D_{n+1} = D_n \cup \{\langle p, q\rangle\}$.

Case $D_f \supseteq \not\equiv_M$**:** We show the statement by induction over $\lfloor p, q\rfloor$ for $\langle p, q\rangle \in \not\equiv_M$.
Base case: If $\lfloor p, q\rfloor = 0$ then $p \in F \Leftrightarrow q \notin F$, so $\langle p, q\rangle \in D_0$ and thus $\langle p, q\rangle \in D_f$.
IH: $\forall \langle r, s\rangle \in \not\equiv_M : (\lfloor r, s\rfloor \leq i \Rightarrow \langle r, s\rangle \in D_f)$.
Induction case: We show that the statement holds for $i + 1$. Assume $\lfloor p, q\rfloor = i + 1$. So there is a context $w(\mathtt{o})$ such that $\lfloor w(\mathtt{o})\rfloor = i + 1$, using M as a Σ-model $\mathfrak{M}$,
$w(t_p)^{\mathfrak{M}} \in F \Leftrightarrow w(t_q)^{\mathfrak{M}} \notin F, \;\; w(\mathtt{o}) = t(f(\bar{u} \cdot \mathtt{o} \cdot \bar{v}))$
for some contexts $t(\mathtt{o})$ and $g[a](\bar{u} \cdot \mathtt{o} \cdot \bar{v})$ (let $f = g[a]$), where $\lfloor t(\mathtt{o})\rfloor = i$ and $\lfloor f(\bar{u} \cdot \mathtt{o} \cdot \bar{v})\rfloor = 1$. Let $q', p' \in Q$ be such that $f(\bar{u} \cdot t_p \cdot \bar{v})^{\mathfrak{M}} = p'$ and $f(\bar{u} \cdot t_q \cdot \bar{v})^{\mathfrak{M}} = q'$. It follows from the choice of w, that
$t(t_{p'})^{\mathfrak{M}} = w(t_p)^{\mathfrak{M}} \in F \Leftrightarrow w(t_q)^{\mathfrak{M}} = t(t_{q'})^{\mathfrak{M}} \notin F$
and thus, since M is complete,

$$g(\bar{p}) \xrightarrow{\varphi}_M p', g(\bar{q}) \xrightarrow{\psi}_M q', \; a \in [\![\varphi \wedge \psi]\!], \; \bar{p} \doteq_i \bar{q} \quad (*)$$

where $p_i = p$ and $q_i = q$. So $p' \not\equiv_M q'$ and by IH we have $\langle p', q'\rangle \in D_f$ because $\lfloor p', q'\rfloor \leq \lfloor t(\mathtt{o})\rfloor = i$. In other words, at some point $\lfloor p', q'\rfloor$ was added to *Frontier* because all newly discovered distinguishable pairs are added. Consider the step n at which $\lfloor p', q'\rfloor$ is *removed* from *Frontier*, so $\langle p', q'\rangle \in D_n$ and assume that $\langle p, q\rangle \notin D_n$ (or else we are done with the induction case). Then the for-all loop is enabled for the choices in $(*)$ and so $\langle p, q\rangle \in D_m$ for some $m > n$. Thus, $\langle p, q\rangle \in D_f$, that proves the induction case.

The two cases imply that $D_f = \not\equiv_M$.

B) Assume M is partial and clean. Since $M^\perp$ is complete and reduced we can use (A) to show that the algorithm computes $\not\equiv_{M^\perp}$ given $M^\perp$ as input. We show that we can lift that proof to M. Let $D^\perp = \{\langle q, q^\perp\rangle \mid q \in Q\}$.

Here we let D_n stand for the value of $D \cup D^\perp$ during iteration n and we let D_f be the value of $D \cup D^\perp$ in step 3. Initially, D_0 is the value of *Frontier*.

We prove that $D_f = \not\equiv_{M^\perp}$. M must be clean. If Q would include an accessible but useless state this would violate the base case $D_0 \subseteq \not\equiv_{M^\perp}$. The proof for the case $D_f \subseteq \not\equiv_{M^\perp}$ is otherwise identical for the proof in (A).

For the case $D_f \supseteq \not\equiv_{M^\perp}$ we need to show that it suffices to use the transitions in $\Delta \cup \Delta^\perp$. Consider the induction case in the proof of case '$\supseteq$' in (A). Assume that $q' = q^\perp$ in $(*)$, or else, if $q', p' \in Q$ then both transitions in $(*)$ are in Δ. It is enough to show that the transition with target $q' = q^\perp$ exists in $\Delta^\perp$. Since $\bar{p} \doteq_i \bar{q}$ and $p_i \in Q$ and $\langle p_i, q_i\rangle \notin D$ we know that $q_i \neq p_i$ and $q_i \neq q^\perp$, i.e., $q_i \in Q$. We know that $\nu_{g(\bar{q})} = \bigwedge_{j=1}^{k} \neg\gamma_j$ for some $k \geq 0$ where the γ_j are the guards of the transitions whose trigger in Δ is $g(\bar{q})$. There are two sub-cases:

1. If $k = 0$ we have that $g(\bar{q})$ does not occur as a trigger in Δ but $g(\bar{p}) = g(\bar{q}_{i:p_i})$ does occur because $p' \in Q$. We also know that $(q_i \in F \Leftrightarrow p_i \in F)$ or else $\langle q_i, p_i\rangle \in D_0$. So, by definition of $\Delta_2^\perp$, $g(\bar{q}) \xrightarrow{\top} q' \in \Delta_2^\perp$.
2. If $k > 0$ then $g(\bar{q})$ occurs as a trigger in Δ. This means that $\mathit{Sat}(\nu_{g(\bar{q})})$, $\psi = \nu_{g(\bar{q})}$, and $g(\bar{q}) \xrightarrow{\psi} q' \in \Delta_1^\perp$ above.

It follows that the use of $\Delta \cup \Delta^\perp$ covers all the transitions from $\Delta_{M^\perp}$ that are needed to complete the induction step in $(*)$. Thus, the proof of $D_f \supseteq \not\equiv_{M^\perp}$ is complete.

We have shown that $D_f = \not\equiv_{M^\perp}$ in case (B). Therefore the final value of D is $(D_f \setminus D^\perp) = \not\equiv_M$ in case (B).

This completes the proof of the theorem. □

Complexity. We assume that M is normalized. Let $m = |\Delta|$ be the number of transitions, $k = |\Gamma|$ be the number of constructors, r be the maximum rank, and ℓ be the size of the largest guard appearing in any transition in Δ. Given a predicate φ of size l in the Boolean algebra $\mathcal{A}$, let $f(l)$ be the complexity of deciding satisfiability of φ. $\Delta^\perp$ contains $\mathcal{O}(mkr)$ transitions and each transition has a guard of size $\mathcal{O}(m\ell)$. STAPart has complexity $\mathcal{O}(m^2k^2r^2f(m\ell))$.

5. Reduction to Minimization of s-FAs

In this section we present the most important contribution of this paper: the problem of minimizing s-TAs can be reduced to the problem of minimizing symbolic finite automata over words. Thanks to this reduction we can use existing s-FA minimization algorithms from [14].

5.1 Review of Symbolic Finite Automata

Definition 16. A *symbolic finite automaton* (*s-FA*) S is a tuple $(\mathcal{A}, Q, q_0, F, \delta)$ where $\mathcal{A}$ is an effective Boolean algebra, Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *final states*, and $\delta \subseteq Q \times \Psi_{\mathcal{A}} \times Q$ is a finite set of *transitions*.

We also denote a transition $(p, \varphi, q) \in \delta$ by $p \xrightarrow{\varphi} q$. A word $w = a_1 \ldots a_k \in U_{\mathcal{A}}^*$ is *accepted from a state* $q \in Q$, denoted $w \in \mathscr{L}_S^{\mathrm{r}}(q)$, if there exists $p_{i-1} \xrightarrow{\varphi_i} p_i$ for $1 \leq i \leq k$, and $k \geq 0$,

such that $a_i \in [\![\varphi_i]\!]$, $p_0 = q$ and $p_k \in F$. The *language accepted by* S is $\mathscr{L}(S) \stackrel{\text{def}}{=} \mathscr{L}^{\mathrm{r}}_S(q_0)$. S is *deterministic* if for any two transitions $p_1 \xrightarrow{\varphi_1} q_1$ and $p_2 \xrightarrow{\varphi_2} q_2$ in δ, if $p_1 = p_2$ and $\mathit{Sat}(\varphi_1 \wedge \varphi_2)$ then $q_1 = q_2$. We refer the reader to [14] for more details.

5.2 Reduction to s-FA Minimization

In this section we show how s-TA transitions can be modeled as s-FA transitions without compromising state distinguishability. Our reduction is inspired by Abdulla et al. [2] where it is shown that the computation of bisimulation equivalence for non-deterministic tree automata can be reduced to computing *word* bisimulation equivalence on transition systems derived from the automata.

The reduction is, in part, based on the relation $\doteq$ that we presented in Definition 14. Observe that step 2(b) of STAPart does the following: when applying the step from a given pair of targets $\langle p', q'\rangle$, find two triggers $g(\bar{q})$ and $g(\bar{p})$ such that $\bar{q} \doteq \bar{p}$ having two transitions $g(\bar{q}) \xrightarrow{\varphi} q'$ and $g(\bar{p}) \xrightarrow{\psi} p'$ with $\mathit{Sat}(\varphi \wedge \psi)$. The key point behind the reduction is to represent the condition that $\bar{p} \doteq \bar{q}$ and $\mathit{Sat}(\varphi \wedge \psi)$ as a combined predicate over an extension of $\mathcal{A}$ that is able to symbolically describe the possible tree contexts. In other words, we want to represent the s-TA transitions

$$g(\bar{q}) \xrightarrow{\varphi} q' \qquad g(\bar{p}) \xrightarrow{\psi} p'$$

where $\bar{q} \doteq_i \bar{p}$ as corresponding s-FA transitions

$$q_i \xrightarrow{\boldsymbol{\phi}^i_{\langle\varphi,\langle g,\bar{q}\rangle\rangle}} q' \qquad p_i \xrightarrow{\boldsymbol{\phi}^i_{\langle\psi,\langle g,\bar{p}\rangle\rangle}} p'$$

where $\boldsymbol{\phi}^i_{\langle\varphi,\langle g,\bar{q}\rangle\rangle}$ and $\boldsymbol{\phi}^i_{\langle\psi,\langle g,\bar{p}\rangle\rangle}$ are predicates over an extended Boolean algebra $\hat{\mathcal{A}}$ of $\mathcal{A}$ so that $\mathit{Sat}(\boldsymbol{\phi}^i_{\langle\varphi,\langle g,\bar{q}\rangle\rangle} \wedge \boldsymbol{\phi}^i_{\langle\psi,\langle g,\bar{p}\rangle\rangle})$ iff the condition 2(b) of STAPart holds (i.e. $\bar{p} \doteq_i \bar{q}$, and $\mathit{Sat}(\varphi \wedge \psi)$). If $p_i \neq q_i$ then the combined condition $\boldsymbol{\phi}^i_{\langle\varphi,\langle g,\bar{q}\rangle\rangle} \wedge \boldsymbol{\phi}^i_{\langle\psi,\langle g,\bar{p}\rangle\rangle}$ should be equivalent to $\bar{p}_{i:\mathsf{o}} = \bar{q}_{i:\mathsf{o}} \wedge \varphi \wedge \psi$, where o is a new constant.[6]

We illustrate one possible way to define the extended effective Boolean algebra $\hat{\mathcal{A}}$ to achieve the effect described above. We use the property that a *Cartesian product algebra* of two effective Boolean algebras is also an effective Boolean algebra. First, let $\hat{Q} = Q \cup \{q^0, \mathsf{o}\}$ where $q^0, \mathsf{o} \notin Q$ are new states[7] and let $G = \{\langle g, \bar{q}\rangle \mid g \in \Gamma, \bar{q} \in \hat{Q}^{\natural(g)}\}$. Let the power set algebra 2^G be represented by an effective Boolean algebra $\mathcal{G}$.[8] For $g \in \Gamma, \bar{q} \in \hat{Q}^{\natural(g)}$, let $\mathit{Is}_{\langle g,\bar{q}\rangle} \in \Psi_{\mathcal{G}}$ be such that

$$[\![\mathit{Is}_{\langle g,\bar{q}\rangle}]\!]_{\mathcal{G}} = \{\langle g, \bar{q}\rangle\}.$$

When $\natural(g) = 0$ we write Is_g for $\mathit{Is}_{\langle g,()\rangle}$. $\hat{\mathcal{A}}$ is defined as the Cartesian product algebra $\mathcal{A} \times \mathcal{G}$ of $\mathcal{A}$ and $\mathcal{G}$. The universe of $\hat{\mathcal{A}}$ is $U \times G$. For $x = \langle a, b\rangle \in U \times G$ let $x[1] = a$ and $x[2] = b$. $\Psi_{\hat{\mathcal{A}}}$ consists of all Boolean combinations of $\lambda x.\alpha(x[1])$ and $\lambda x.\gamma(x[2])$ for $\alpha \in \Psi_{\mathcal{A}}$ and $\gamma \in \Psi_{\mathcal{G}}$. For $\alpha \in \Psi_{\mathcal{A}}$ (similarly for $\gamma \in \Psi_{\mathcal{G}}$) let

$$\neg_{\hat{\mathcal{A}}}(\lambda x.\alpha(x[1])) \stackrel{\text{def}}{=} \lambda x.\neg_{\mathcal{A}}\alpha(x[1]).$$

Given $\psi \in \Psi_{\hat{\mathcal{A}}}$, ψ has, without loss of generality, an equivalent disjunctive normal form (DNF)

$$\bigvee_{i=1}^{k}((\lambda x.\alpha_i(x[1])) \wedge (\lambda x.\gamma_i(x[2])))$$

where all $\alpha_i \in \Psi_{\mathcal{A}}$ and $\gamma_i \in \Psi_{\mathcal{G}}$. The denotation of ψ in $\hat{\mathcal{A}}$ is

$$[\![\psi]\!]_{\hat{\mathcal{A}}} = \bigcup_{i=1}^{k} [\![\alpha_i]\!]_{\mathcal{A}} \times [\![\gamma_i]\!]_{\mathcal{G}}.$$

Proposition 4. *$\hat{\mathcal{A}}$ is an effective Boolean algebra and if $\mathcal{A}$ is decidable then $\hat{\mathcal{A}}$ is decidable.*

Proof. The property that $\hat{\mathcal{A}}$ is an effective Boolean algebra follows from Morgan's laws, laws of distributivity of Boolean connectives, and that $\mathcal{A}$ and $\mathcal{G}$ are effective Boolean algebras. Assume $\mathcal{A}$ is decidable. $\mathcal{G}$ is trivially decidable because it is finite. Deciding satisfiability of $\psi \in \Psi_{\hat{\mathcal{A}}}$ reduces effectively to satisfiability checks in $\mathcal{A}$ and $\mathcal{G}$ as follows. Take the DNF of ψ as above. Then $\mathit{Sat}_{\hat{\mathcal{A}}}(\psi)$ holds iff for some i both $\mathit{Sat}_{\mathcal{A}}(\alpha_i)$ and $\mathit{Sat}_{\mathcal{G}}(\gamma_i)$ hold. □

The following key construction is used to create the predicates of the transitions in the s-FA.

Definition 17. For $g \in \Gamma$, $\bar{q} \in Q^{\natural(g)}$, and $\varphi \in \Psi_{\mathcal{A}}$, let

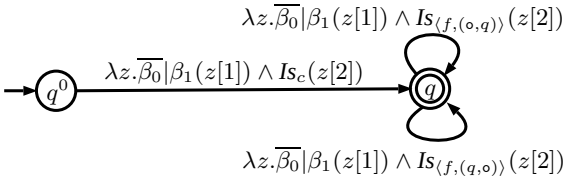
$$\boldsymbol{\phi}^0_{\langle\varphi,g\rangle} \stackrel{\text{def}}{=} \lambda z.\varphi(z[1]) \wedge \mathit{Is}_g(z[2]),\ \text{if } \natural(g) = 0;$$
$$\boldsymbol{\phi}^i_{\langle\varphi,\langle g,\bar{q}\rangle\rangle} \stackrel{\text{def}}{=} \lambda z.\varphi(z[1]) \wedge \mathit{Is}_{\langle g,\bar{q}_{i:\mathsf{o}}\rangle}(z[2]),\ \text{for } 1 \leq i \leq \natural(g).$$

Definition 18. The *s-FA of* M is $\hat{M} \stackrel{\text{def}}{=} (\hat{\mathcal{A}}, \hat{Q}, q^0, F, \delta)$, where

$$\begin{aligned}\delta \;=\; & \{(q^0, \boldsymbol{\phi}^0_{\langle\varphi,g\rangle}, p) \mid g() \xrightarrow{\varphi}_M p\} \cup \\ & \{(q_i, \boldsymbol{\phi}^i_{\langle\varphi,\langle g,\bar{q}\rangle\rangle}, p) \mid g(\bar{q}) \xrightarrow{\varphi}_M p, 1 \leq i \leq \natural(g)\}\end{aligned}$$

Notice the duality between the transitions of M and the transitions of $\hat{M}$. Recall that the concrete alphabet is Σ. There is a close connection between Σ-contexts and words over $U_{\hat{\mathcal{A}}}$. Namely, a word over $U_{\hat{\mathcal{A}}}$ describes a path from o to the root of the Σ-context with precisely enough information to preserve state (in)distinguishability.

Example 5. Consider the s-TA $M = M_{0,1}$ with $M_{i,j}$ as in Example 4. In this case $\hat{M}$ is as follows:



The transition from q^0 to q represents the transitions of M that read constructors of rank 0. Since c is the only constructor with rank 0, the predicate on the transition checks that the constructor component $z[2]$ is c ($\mathit{Is}_c(z[2])$) and that the label component $z[1]$ of the input symbol satisfies $\overline{\beta_0}|\beta_1$ — i.e., the predicate of the original transition in M. The looping transitions on q represent the original transition from q to q appearing in M. Since f has rank 2, there are two transitions from q to q in $\hat{M}$: one places the context on the first child ($\mathit{Is}_{\langle f,(\mathsf{o},q)\rangle}(z[2])$) and one places the context on the second child ($\mathit{Is}_{\langle f,(q,\mathsf{o})\rangle}(z[2])$). Since M is already minimal $\hat{M}$ is also minimal up to normalization. ⊠

Observe that q^0 is always distinguishable from all the other states in $\hat{M}$ when M is clean. This is because the predicate Is_c, where c is a constant, is false for all transitions whose source is different from q^0.

SFARed (s-TA state distinguishability via s-FA reduction)

Input: deterministic clean s-TA M, *Output*: $\not\equiv_M$.

Return $\not\equiv_{\hat{M}} \setminus \{\langle q^0, q\rangle \mid q \in Q_M\}$.

[6] The definition of $\bar{q}_{i:\mathsf{o}}$ is similar to the notion of *environment* in [2].

[7] The state q^0 is used below as the *initial state* in the s-FA construction and the state o is used as a *context state* as explained in Definition 17 below. Assuming that $q^0 \neq \mathsf{o}$ is not necessary but intuitively useful.

[8] There are several ways in which $\mathcal{G}$ can be defined, one way is to use a finite restriction of the BDD algebra $\mathcal{B}$ and encode $\langle g, \bar{q}\rangle$ as a number.

Lemma 5. *If M is deterministic then $\hat{M}$ is deterministic.*

Proof. Assume that M is deterministic. Consider two transitions of $\hat{M}$ with same source states. If the guards use testers of distinct constructors then their conjunction is unsatisfiable. Assume that they use testers for the same constructor g and that the conjunction of their guards is satisfiable. We must show that the target states are the same. Let $k = \natural(g)$.

Assume first that $k = 0$. Then the $\hat{M}$ transitions have the form $(q^0, \boldsymbol{\phi}^0_{\langle\varphi,g\rangle}, p)$ and $(q^0, \boldsymbol{\phi}^0_{\langle\psi,g\rangle}, r)$ for some φ,ψ, p and r such that $g() \xrightarrow{\varphi}_M p$ and $g() \xrightarrow{\psi}_M r$. From $\mathit{Sat}(\boldsymbol{\phi}^0_{\langle\varphi,g\rangle} \wedge \boldsymbol{\phi}^0_{\langle\psi,g\rangle})$ follows that $\mathit{Sat}(\varphi \wedge \psi)$, and since M is deterministic it follows that $p = r$.

Assume now that $k > 0$. Then the $\hat{M}$ transitions have the form $(q_i, \boldsymbol{\phi}^i_{\langle\varphi,\langle g,\bar{q}\rangle\rangle}, r)$ and $(p_j, \boldsymbol{\phi}^j_{\langle\psi,\langle g,\bar{p}\rangle\rangle}, s)$ for some i and j, $1 \leq i, j \leq k$, where $g(\bar{q}) \xrightarrow{\varphi} r$ and $g(\bar{p}) \xrightarrow{\psi} s$ are transitions in Δ. We are assuming that $q_i = p_j$ and

$$\mathit{Sat}(\boldsymbol{\phi}^i_{\langle\varphi,\langle g,\bar{q}\rangle\rangle} \wedge \boldsymbol{\phi}^j_{\langle\psi,\langle g,\bar{p}\rangle\rangle})$$

and need to show $r = s$. If we expand the definitions we get

$$\mathit{Sat}(\lambda z.\varphi(z[1]) \wedge \mathit{Is}_{\langle g,\bar{q}_{i:\mathtt{o}}\rangle}(z[2]) \wedge \psi(z[1]) \wedge \mathit{Is}_{\langle g,\bar{p}_{j:\mathtt{o}}\rangle}(z[2]))$$

from which follows that $\mathit{Sat}(\varphi \wedge \psi)$ and $\bar{q}_{i:\mathtt{o}} = \bar{p}_{j:\mathtt{o}}$. The latter is only possible if $i = j$ because $\mathtt{o}$ does not occur in Q_M. So $\bar{q}_{i:\mathtt{o}} = \bar{p}_{i:\mathtt{o}}$, and by using $q_i = p_i$, we have $\bar{q} = \bar{p}$. It follows by determinism of M that $r = s$. □

We are now ready to state our main theorem that relates the minimality of $\hat{M}$ to the minimality of M. First we recall the definition of indistinguishability of states in an s-FA, which is a simplified version of the definition used for s-TAs. Two states p and q are *indistinguishable* in $\hat{M}$, $p \equiv_{\hat{M}} q$, if $\mathscr{L}^{\mathrm{r}}_{\hat{M}}(p) = \mathscr{L}^{\mathrm{r}}_{\hat{M}}(q)$. The analogue of $\mathscr{L}^{\mathrm{r}}_{\hat{M}}(q)$ in M is the *up-language of a state* q:

$$\mathcal{L}^{\uparrow}_M(q) \stackrel{\text{def}}{=} \{\Sigma\text{-context } t(\mathtt{o}) \mid t(t_q) \in \mathcal{L}(M)\}.$$

Thus, $p \equiv_M q$ iff $\mathcal{L}^{\uparrow}_M(p) = \mathcal{L}^{\uparrow}_M(q)$. We write $\mathcal{L}^{\uparrow}(q)$ for $\mathcal{L}^{\uparrow}_M(q)$ if M is clear. It is technically convenient to work with up-languages in the proof of Theorem 6.

Theorem 6. *Assume $\mathcal{A}$ is decidable and M is deterministic and clean. For all $p, q \in Q_M$: $p \equiv_M q \Leftrightarrow p \equiv_{\hat{M}} q$.*

Proof. The indistinguishability relation is well-defined and effectively computable for $\hat{M}$ due to Proposition 4 and Lemma 5. We show $p \equiv_{\hat{M}} q \Leftrightarrow p \equiv_M q$ for all $p, q \in Q$. We do not need to consider the completed versions of M or $\hat{M}$ here, M can be assumed to be partial.

Case $p \equiv_{\hat{M}} q \Rightarrow p \equiv_M q$**:** By way of contradiction, assume there exist p and q such that $p \equiv_{\hat{M}} q$ but $p \not\equiv_M q$. Then we can select p, q, and a Σ-context $t(\mathtt{o})$ such that $\mathit{Hyp}(p, q, t)$ holds with $\lfloor t(\mathtt{o})\rfloor$ being *smallest*:

$$\mathit{Hyp}(p, q, t)\text{: } p \equiv_{\hat{M}} q,\ p \not\equiv_M q,\ t(\mathtt{o}) \in \mathcal{L}^{\uparrow}(q) \setminus \mathcal{L}^{\uparrow}(p).$$

If $\lfloor t(\mathtt{o})\rfloor = 0$ then $\mathtt{o} \in \mathcal{L}^{\uparrow}(q)$ and thus $q \in F$. But then, by $p \equiv_{\hat{M}} q$, we have $p \in F$ and so $\mathtt{o} \in \mathcal{L}^{\uparrow}(p)$, *contradicting* that $t(\mathtt{o}) \notin \mathcal{L}^{\uparrow}(p)$.

Assume $\lfloor t(\mathtt{o})\rfloor > 0$. Then, for some context $t'(\mathtt{o})$, $t(\mathtt{o}) = t'(g[a](\bar{u}_{i:\mathtt{o}}))$ and, because $t(\mathtt{o}) \in \mathcal{L}^{\uparrow}(q)$, there is a transition $g(\bar{q}) \xrightarrow{\varphi}_M r$ where $q_i = q$ and $a \in [\![\varphi]\!]$ and $u_j \in \mathcal{L}^{\downarrow}(q_j)$ for $j \neq i$ and $t'(\mathtt{o}) \in \mathcal{L}^{\uparrow}(r)$. So there is a transition instance $(q, \langle a, \langle g, \bar{q}_{i:\mathtt{o}}\rangle\rangle, r)$ in $\hat{M}$ and from $p \equiv_{\hat{M}} q$ follows that there is a transition instance $(p, \langle a, \langle g, \bar{q}_{i:\mathtt{o}}\rangle\rangle, s)$ in $\hat{M}$ for some s such that $r \equiv_{\hat{M}} s$. So there is a corresponding transition $(p, \boldsymbol{\phi}^i_{\langle\psi,\langle g,\bar{q}_{i:p}\rangle\rangle}, s)$ in $\hat{M}$ where $a \in [\![\psi]\!]$, which means that there is a transition $g(\bar{q}_{i:p}) \xrightarrow{\psi}_M s$. We have two sub-cases:

1. Suppose $t'(\mathtt{o}) \notin \mathcal{L}^{\uparrow}(s)$. This contradicts the choice of p, q, and t in $\mathit{Hyp}(p, q, t)$ because $\mathit{Hyp}(s, r, t')$ holds with $\lfloor t'(\mathtt{o})\rfloor < \lfloor t(\mathtt{o})\rfloor$, *contradicting* that $\lfloor t(\mathtt{o})\rfloor$ is smallest.
2. Suppose $t'(\mathtt{o}) \in \mathcal{L}^{\uparrow}(s)$. Then, by using the transition $g(\bar{q}_{i:p}) \xrightarrow{\psi}_M s$ we get that $t'(g[a](\bar{u}_{i:\mathtt{o}})) \in \mathcal{L}^{\uparrow}(p)$, but this *contradicts* that $t(\mathtt{o}) \notin \mathcal{L}^{\uparrow}(p)$.

Case $p \equiv_M q \Rightarrow p \equiv_{\hat{M}} q$**:** By way of contradiction, assume there exist p and q such that $p \equiv_M q$ but $p \not\equiv_{\hat{M}} q$. Then we can select p, q, and a word w over $U_{\hat{\mathcal{A}}}$ such that $\mathit{Hyp}'(p, q, w)$ holds with $|w|$ being *smallest*:

$$\mathit{Hyp}'(p, q, w)\text{: } p \equiv_M q,\ p \not\equiv_{\hat{M}} q,\ w \in \mathscr{L}^{\mathrm{r}}(q) \setminus \mathscr{L}^{\mathrm{r}}(p).$$

If $|w| = 0$ then $q \in F$, and from $p \equiv_M q$ follows that $p \in F$ which *contradicts* that $w \notin \mathscr{L}^{\mathrm{r}}(p)$.

Assume $|w| > 0$. If w starts with $\langle a, \langle c, ()\rangle\rangle$ then from $w \in \mathscr{L}^{\mathrm{r}}(q)$ follows that q must be q^0 which contradicts that $q \in Q_M$. So $w = \langle a, \langle g, \bar{q}_{i:\mathtt{o}}\rangle\rangle \cdot v$ for some word v and $g \in \Gamma$ with $\natural(g) \geq 1$, and there is a transition $g(\bar{q}) \xrightarrow{\varphi}_M r$ where $q_i = q$ and $a \in [\![\varphi]\!]$ and $v \in \mathscr{L}^{\mathrm{r}}(r)$. Choose $t'(\mathtt{o}) \in \mathcal{L}^{\uparrow}(r)$, $t'(\mathtt{o})$ exists because M is clean. By using $g(\bar{q}) \xrightarrow{\varphi}_M r$, let $t(\mathtt{o}) = t'(g[a](\bar{u}_{i:\mathtt{o}})) \in \mathcal{L}^{\uparrow}(q)$ where, for $j \neq i$, $u_j \in \mathcal{L}^{\downarrow}(q_j)$. From $p \equiv_M q$ follows now that $t(\mathtt{o}) \in \mathcal{L}^{\uparrow}(p)$. Therefore, there is $g(\bar{q}_{i:p}) \xrightarrow{\psi}_M s$ such that $a \in [\![\psi]\!]$ and $r \equiv_M s$. It follows that there is a transition $(p, \boldsymbol{\phi}^i_{\langle\psi,\langle g,\bar{q}_{i:p}\rangle\rangle}, s)$ in $\hat{M}$. So there is a transition instance $(p, \langle a, \langle g, \bar{q}_{i:\mathtt{o}}\rangle\rangle, s)$ in $\hat{M}$. We have two sub-cases:

1. Suppose $v \notin \mathscr{L}^{\mathrm{r}}(s)$. This contradicts the choice of p, q, an w in $\mathit{Hyp}'(p, q, w)$ because $\mathit{Hyp}'(s, r, v)$ holds with $|v| < |w|$, *contradicting* that $|w|$ is smallest.
2. Suppose $v \in \mathscr{L}^{\mathrm{r}}(s)$. Then $\langle a, \langle g, \bar{q}_{i:\mathtt{o}}\rangle\rangle \cdot v \in \mathscr{L}^{\mathrm{r}}(p)$, *contradicting* that $w \notin \mathscr{L}^{\mathrm{r}}(p)$.

The case $p \equiv_M q \Rightarrow p \equiv_{\hat{M}} q$ and thus the theorem follows. □

An interesting observation about Theorem 6 is that completion is never needed in its proof although it is assumed in the minimization algorithm for s-FAs. In particular, $U_{\hat{\mathcal{A}}}$ contains elements such as $\langle a, \langle g, (\mathtt{o}, \mathtt{o})\rangle\rangle$ that are meaningless in M but arise after completion of $\hat{M}$. It is therefore crucial to implement $\hat{\mathcal{A}}$ symbolically in order to avoid explicit enumeration of all the symbols in $\hat{Q}$.

Complexity. Given an s-TA M, let $n = |Q|$ be the number of states, $m = |\Delta|$ be the number of transitions, constructors, r be the maximum rank of any constructor in Γ, and ℓ be the size of the largest guard appearing in any transitions in Δ. Given a predicate φ of size l in the Boolean algebra $\mathcal{A}$, let $f(l)$ be the complexity of deciding satisfiability of φ. We first describe the size of the reduced s-FA $\hat{M}$ and then analyze the complexity of minimizing $\hat{M}$ using SFA minimization. Given $\not\equiv_{\hat{M}}$ and M, the induced minimal s-TA can then be built in time that is linear (up to normalization of predicates).

The s-FA $\hat{M}$ has $\hat{n} = n + 2$ states, $\hat{m} = \mathcal{O}(mr)$ transitions and the largest guard has size $\hat{\ell} = \mathcal{O}(mr + m\ell)$ due to the s-FA completion. Given a predicate φ of size l in the Boolean algebra $\hat{\mathcal{A}}$, let $g_f(l)$ be the complexity of deciding satisfiability of φ (notice that g_f can depend on f). Using the symbolic extension of Moore algorithm [14, Section 3], $\hat{M}$ can be minimized in time $\mathcal{O}(m^2 r^2 g_f(mr + ml))$. Using the symbolic extension of Hopcroft algorithm [14, Section 4], $\hat{M}$ can be minimized in time

$2^{\mathcal{O}(mr)}\mathcal{O}(g_f(m^2r + m^2l) + n\,log\,n)$. Finally, using the new minimization algorithm $Min^{\mathbf{N}}_{\text{SFA}}$ from [14, Section 5], $\hat{M}$ can be minimized in time $\mathcal{O}(n^2 log\, n\, g_f(nmr + nm\ell))$. As shown in [14] the last algorithm outperforms the previous two. An interesting aspect of this last algorithm is that most of the complexity is handled by the solver of the Boolean algebra. We will show that this aspect is crucial in practice.

6. Evaluation

We evaluated the performance of the minimization algorithms with three sets of s-TAs.[9]

1. Bottom-up tree automata over small finite alphabets from the tests appearing in the VATA library [26]. This experiments allowed us to compare against existing tree automata libraries.
2. s-TAs that operate over *large finite alphabets* that can be represented succinctly using the theory of bit-vectors. This experiment showed how s-TAs are beneficial even in the presence of finite alphabets.
3. s-TAs over the alphabet sort $(\mathit{Int} \times \mathit{Int} \times \mathit{String})$. This experiment showed how our algorithms perform on *infinite alphabets*.

We use STAPart and SFARed to refer to the algorithms described in Section 4 and 5 respectively. We also measure the running time of STAPart on a complete automaton (referred to as STAComp).

Implementation details. STAPart allows us to apply further optimizations that take advantage of M being partial. In particular, we use the notion of a *state signature* presented in [8]. The signature of a state q is the set

$$\mathit{sig}_q = \{(g, |\bar{x}|, i) \mid g(\bar{x}) \in \mathit{lhs}(\Delta), x_i = q\} \cup \{\$ \mid q \in F\}$$

One can show that if two states p and q have different signatures, then $p \not\equiv_M q$. Our implementation starts by computing the signatures of all states and then adds $\{p, q\}$ to the initial value of D for all p and q having distinct signatures. This simple optimization is very effective in practice.

In our implementation of SFARed the Boolean algebra $\hat{\mathcal{A}}$ is implemented using the SMT solver Z3 [17], which indeed supports data types [17].[10]

6.1 Small finite alphabets

The VATA library [26] for manipulating *non-deterministic* bottom-up tree automata contains a large collection of automata that may appear when verifying concurrent programs. The automata appearing in the collection have ranks varying between 2 and 11. Here we use the determinized and completed versions of 2414 automata appearing in the VATA library to compare our algorithms, the Lethal library, and the implementation described in [8] (DTAmin from now on). The Lethal library is written in Java and requires the input automaton to be complete. The DTAmin implementation is written in C++, does not require the input automaton to be complete, and is based on the highly optimized implementation of tree automata minimization described in [8]. Fig. 2 shows the running times parametrized by number of states (a), number of transitions (b), number of transitions in the complete automaton (c), and number of transitions in the minimized automaton (d). In (c) we only report the running times on the instances for which the completion algorithm did not time out.

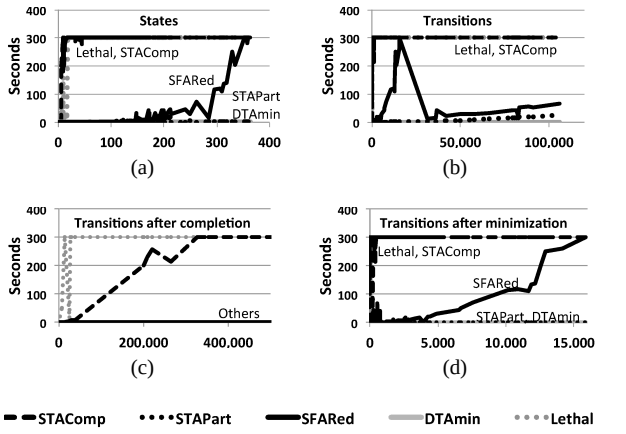


Figure 2. Bottom-up tree automata over small finite alphabets from the VATA library.

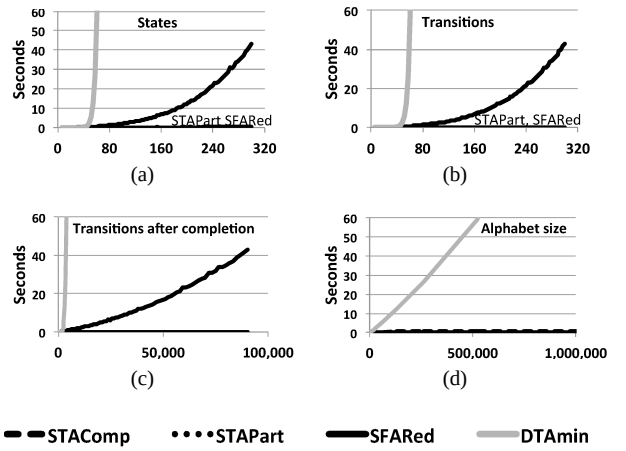


Figure 3. Tree automata over large bit-vector alphabets.

Analysis. STAPart and DTAmin scale to large instances, and are faster than the other algorithms. STAComp and Lethal time out for small instances since the size of the completed automata grows very quickly. SFARed is slower than STAPart and DTAmin, but it still scales to large instances. From (c) and (d) we can see that the bottleneck of SFARed is the number of states rather than the number of transitions. This is due to the fact that the number of states directly affects the number of elements in the the Boolean algebra $\hat{\mathcal{A}}$.

6.2 Large finite alphabets

Here we consider a particular set of s-TAs over the theory of bit-vectors. This theory is widely used in practice, and finite-alphabet techniques do not scale in this setting.[11] Let β_k be the predicate such that $\beta_k(x)$ holds iff the $(k\%32)$'th bit of x is 1. Consider the constructors $\Gamma = \{\texttt{zero}, \texttt{two}\}$ where $\natural(\texttt{zero}) = 0$ and $\natural(\texttt{two}) =$

[9] All the experiments were run on a 4-core Intel i7-2600 CPU 3.40GHz, with 8GB of RAM and we set our timeout to 300 seconds. For the algorithms that require completion we do not measure the completion time.

[10] In the case of large finite label alphabets, such as ASCII, the use of an SMT solver may be an overkill. Another option is to use a finite restriction of the BDD algebra to represent elements of $U_{\hat{A}}$ by reserving certain bit intervals for encoding the constructors, the labels, and the state sequences, respectively.

[11] This phenomenon has also been demonstrated for finite automata over strings in [14, Section 6.3].

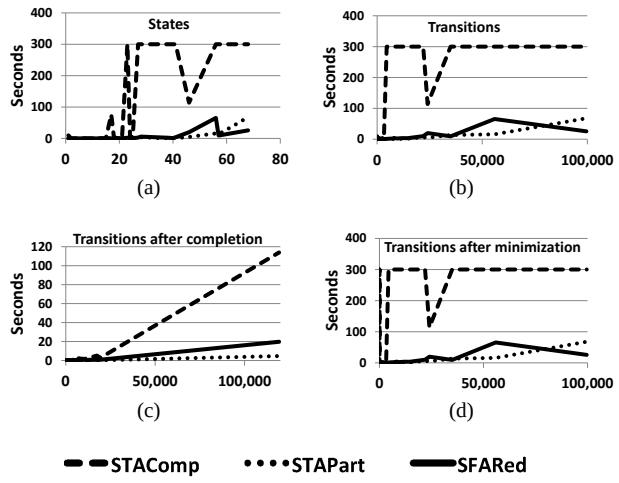


Figure 4. s-TAs over the alphabet $(\mathit{Int} \times \mathit{Int} \times \mathit{String})$.

2 and let B_k for $k \geq 1$ be an s-TA the following transitions:

$$\Delta_{B_k} = \{\texttt{zero} \xrightarrow{x=0} q_0,\ \texttt{zero} \xrightarrow{x=1} q_1,\ \texttt{zero} \xrightarrow{x=2} q_2\}$$
$$\cup\ \{\texttt{two}(q_{3i}, q_{3i+2}) \xrightarrow{\beta_i} q_{3i+3}, \texttt{two}(q_{3i+1}, q_{3i+2}) \xrightarrow{\beta_i} q_{3i+4},$$
$$\texttt{two}(q_{3i+2}, q_{3i+2}) \xrightarrow{\beta_i} q_{3i+5}\}_{i=0}^{k-1}$$

The final states of B_k are q_{3k} and q_{3k+1}. In particular: the states q_{3i} are used at level $k-i$ to denote that the leftmost leaf has label 0 (similarly for 1), and the states q_{3i+2} are used at level $k-i$ to denote that all the underlying leaves have label 2. A tree is accepted by B_k iff it has depth k, is fully balanced, and each node label at level $l < k$ (root having level 0) satisfies β_{k-l}. Moreover, all leaves are labeled with 2, except the leftmost leaf, which is labeled with either 0 or 1. The s-TA B_k is not minimal: in particular, for every i, the state q_{3i} is not distinguishable from the state q_{3i+1}. We consider the set of s-TAs $\{B_k \mid 1 \leq k \leq 100\}$ and we compare our algorithms to DTAmin in Fig. 3. In order to run the algorithm DTAmin we finitize the alphabet by computing the set all the satisfiable Boolean combinations of the predicates appearing in the s-TA.

Analysis. DTAmin is outperformed by all the symbolic algorithms and it times out at $k = 17$. For $k < 32$, the set of minterms (i.e., all satisfiable Boolean combination) computed from B_k contains 2^k satisfiable Boolean combinations of β_i. This causes DTAmin's running time to be exponential in k. In contrast, none of the algorithms described in this paper requires the concrete representation of the alphabet. STAPart and SFARed perform comparably and are faster than STAComp.

6.3 Complex theories

We compare the performance of our algorithms over a sample set of 280 randomly generated s-TAs over the alphabet $(\mathit{Int} \times \mathit{Int} \times \mathit{String})$. We consider trees over the ranked alphabet $\Gamma = \{zero, one, three\}$, such that $\natural(zero) = 0$, $\natural(one) = 1$, and $\natural(three) = 3$. The guards of each s-TA are conjunctions of equality predicates over strings and linear predicates over integers. Each s-TA is non-empty and has at least one rule for each constructor. The results are shown in Fig. 4.

Analysis. STAComp times out for relatively small instances. STAPart and SFARed are quite comparable; interestingly, in some instances SFARed is actually faster. Given that the SFARed is a simple reduction to SFA minimization and uses an off-the-shelf SMT solver, we find this result remarkably surprising.

7. Related Work

Minimization theory. The use of distinguishability refinement for DFA minimization is attributed to Huffman [25] and Moore [28], e.g., the DFA minimization algorithm in [23, Fig. 3.8] uses this idea. A generalization of this idea to symbolic finite automata (s-FAs) is investigated in [14]. This approach is particularly well suited for symbolic alphabets where it is impossible to iterate over the whole alphabet Σ. In contrast, such iteration is in general needed for concrete alphabets [8, 12, 13] and implies a complexity factor of $|\Sigma|$. The definition of $\equiv_M$ (Definition 10) is an extension of [5, Definition 5.3] to infinite alphabets. It is similarly related to the definition of *Equiv* in [12] where the *up-language of state* q of M is a generalization of the *right-language of state* q of a DFA. Similarly, the *left-language* of states in finite word automata is lifted to the *down-language* of states in finite tree automata [12].

Minimization algorithms. The original paper on tree automata minimization [5] as well as the treatment of minimality in [19] are theoretical. The description of minimality in [13] is also brief and it requires completion. An efficient implementation of the the algorithm for minimizing partial *unranked* tree automata is presented in [8]. Although highly optimized, the implementation in [8] only works for finite alphabets. Our implementation of STAPart from Section 4 extends the one described in [8] to symbolic alphabets.

Completion and large alphabets. When working with finite but large alphabets, completion can cause an explosion in the number of transitions of the automaton. This happens already for DFAs and specialized algorithms have been proposed to minimize partial DFAs [30]. As shown in [14], the symbolic representation of transition helps mitigate the blow-up in the number of transitions.

Tree automata libraries. We compared our implementation with the tree automata library Lethal [11] and the algorithm DTAmin from [8]. Lethal only supports minimization of complete tree automata. DTAmin is a stand-alone implementation of a minimization algorithm for partial tree automata and does not support other tree automata operations. Both these libraries only support finite alphabets. To the best of our knowledge Lethal and DTAmin are the only open-source implementations of tree automata minimization and are therefore the only libraries we compared against.

VATA and Timbuk are libraries for manipulating non-deterministic tree automata [20, 26]. Timbuk does not support minimization. VATA supports state reduction of non-deterministic tree automata via upward and downward bisimulation. For deterministic automata, the quotient automaton resulting from upward bisimulation is the minimal one. However, the performance of VATA is slower than that of DTAmin as DTAmin is highly optimized for deterministic automata. Mona [21] supports minimization of deterministic *binary guided* tree automata. This model is orthogonal to ours and we cannot compare our implementation against the one in Mona. Mona also uses symbolic techniques (BDDs) to succinctly represent the automata states and transitions, but it is specialized to the theory of bit-vectors. Applying Mona's transition and state compression techniques to symbolic automata is an interesting research direction. To the best of our knowledge, FAST is the only available library for symbolic tree automata and is the one we build on [16].

Automata with predicates. The concept of automata with predicates instead of concrete symbols was first discussed in [31] in the context of natural language processing. The first paper to formally introduce symbolic automata in combination with the notion of *effective Boolean algebra* was [33] and since then numerous extensions have been proposed [15]. Symbolic tree automata have been studied in [16, 32]. To the best of our knowledge, no algorithms for s-TA minimization have been studied prior to this paper.

Non-deterministic automata. The minimization algorithms presented in this paper require the s-TA to be deterministic. Determinization of s-TAs is expensive and may imply an exponential increase not only in the number of states but also in the number of predicates [32]. Techniques based on *bisimulation* have been proposed to directly minimize nondeterministic tree automata [1–3, 22]. The reduction to s-FAs we propose in Section 5 is a symbolic extension of one of the reduction techniques proposed in [2]. While our reduction handles tree contexts by building an *environment* alphabet, the reduction in [2] does so by creating a more complex state space that enumerates all possible environments. Moreover, while our reduction produces an automaton the one in [2] produces a transitions system. Therefore, our algorithm can directly leverage s-FA minimization and take advantage of the symbolic representation of the alphabet in way that the reduction [2] could not do. Without this symbolic representation the alphabet would be as large as the complete s-TA and the minimization algorithm would be as slow as STAPart.

Applying the bisimulation techniques from [1–3, 22] to nondeterministic symbolic tree automata is an interesting research direction we plan to pursue. Unfortunately, some of the techniques that work well with finite alphabets, need new efficient data structures to work in the purely symbolic setting. For example, multisets based on *multi-terminal binary decision diagrams* [4, 10] are used in [1] to represent partitions and to efficiently split blocks over a finite alphabet. The algorithm in [1] is an adaptation of the *relational coarsest partition* algorithm [29] to automata with *large finite* alphabets. Generalizing such data structures and algorithms to represent finite partitions over *infinite* alphabets is one of the key challenges that arises when transitioning to symbolic automata.

Applications. The development of the theory of symbolic tree automata is motivated by several concrete practical problems [16]. Solvers for Monadic Second-Order Logic (MSO) over trees work well in practice by keeping tree automata minimal at each step and by representing the transitions symbolically [21]. Programs that manipulate trees are ubiquitous and domain-specific languages for tree transformation have been studied in several different contexts [16]. Automata minimization is used to keep the language representations succinct and enable analysis of large programs.

References

[1] P. Abdulla, J. Deneux, L. Kaati, and M. Nilsson. Minimization of nondeterministic automata with large alphabets. In *CIAA*, volume 3845 of *LNCS*, pages 31–42. Springer, 2006.

[2] P. Abdulla, A. Bouajjani, L. Holík, L. Kaati, and T. Vojnar. Composed bisimulation for tree automata. In *CIAA*, volume 5148 of *LNCS*, pages 212–222. Springer, 2008.

[3] R. Almeida, L. Holík, and R. Mayr. Minimization of nondeterministic tree automata. *TACAS*, 2016.

[4] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on*, pages 188–191, Nov 1993.

[5] W. S. Brainerd. The minimalization of tree automata. *Information and Control*, 13(5):484 – 491, 1968.

[6] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C–35(8):677–691, 1986.

[7] J. A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In *Proc. Sympos. Math. Theory of Automata*, pages 529–561, New York, 1963.

[8] R. C. Carrasco, J. Daciuk, and M. L. Forcada. An implementation of deterministic tree automata minimization. In *CIAA'07*, volume 4783 of *LNCS*, pages 122–129. Springer, 2007.

[9] C. C. Chang and H. J. Keisler. *Model Theory*, volume 73 of *Studies in Logic and the Foundation of Mathematics*. North Holland, third edition, 1990.

[10] E. Clarke, M. Fujita, P. McGeer, K. McMillan, and J. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In *IWLS93: International Workshop on Logic Synthesis*, pages 6a:1–15, Lake Tahoe, CA, May 1993.

[11] P. Claves, D. Jansen, S. J. Holtrup, M. Mohr, A. Reis, M. Schatz, and I. Thesing. *Lethal: Little Extensible Tree and Hedge Automaton Library*. http://lethal.sourceforge.net/.

[12] L. Cleophas, D. G. Kourie, T. Strauss, and B. W. Watson. On minimizing deterministic tree automata. In J. Holub and J. Žďárek, editors, *PSC '09*, pages 173–182, 2009.

[13] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications, 2007.

[14] L. D'Antoni and M. Veanes. Minimization of symbolic automata. POPL'14, pages 541–553, New York, NY, USA, 2014. ACM. .

[15] L. D'Antoni and M. Veanes. Extended symbolic finite automata and transducers. *Formal Methods in System Design*, 47(1):93–119, Aug. 2015.

[16] L. D'Antoni, M. Veanes, B. Livshits, and D. Molnar. Fast: A transducer-based language for tree manipulation. *TOPLAS*, 38(1):1–32, Oct. 2015.

[17] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS'08*, LNCS. Springer, 2008.

[18] L. de Moura and N. Bjørner. Satisfiability modulo theories: Introduction and applications. *Comm. ACM*, 54(9):69–77, 2011.

[19] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[20] T. Genet and V. V. T. Tong. *Timbuk: A Tree Automata Library*. http://www.irisa.fr/celtique/genet/timbuk/.

[21] J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *TACAS '95*, volume 1019. Springer, 1995.

[22] J. Högberg, A. Maletti, and J. May. Backward and forward bisimulation minimisation of tree automata. In *CIAA*, volume 4783 of *LNCS*, pages 109–121. Springer, 2007.

[23] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.

[24] H. Hosoya and B. C. Pierce. XDuce: A statically typed XML processing language. *ACM Trans. Internet Technol.*, 3(2):117–148, May 2003. .

[25] D. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3–4):161–190,275–303, 1954.

[26] O. Lengal, J. Šimáček, and T. Vojnar. Vata: A library for efficient manipulation of non-deterministic tree automata. *TACAS'12*, 7214: 79–94, 2012. .

[27] J. May and K. Knight. A primer on tree automata software for natural language processing, 2008.

[28] E. F. Moore. Gedanken-experiments on sequential machines. *Automata studies, Annals of mathematics studies*, (34):129–153, 1956.

[29] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[30] A. Valmari. Fast brief practical DFA minimization. *Information Processing Letters*, 112:213–217, 2012.

[31] G. van Noord and D. Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286, 2001.

[32] M. Veanes and N. Bjørner. Symbolic tree automata. *Information Processing Letters*, 115(3):418–424, 2015.

[33] M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjørner. Symbolic finite state transducers: Algorithms and applications. In *POPL'12*, pages 137–150, 2012.