Minimization of Symbolic Automata*

Margus Veanes

Microsoft Research
margus@microsoft.com

Abstract. Symbolic automata theory lifts classical automata theory to rich alphabet theories. It does so by replacing an explicit alphabet with an alphabet described implicitly by a Boolean algebra. We study here one of the core problems, minimization, of automata. We introduce a new minimization algorithm for symbolic automata that takes advantage of state-of-the-art constraint solving techniques for automata analysis that are both expressive and efficient, even for very large and infinite alphabets.

1 Introduction

Classical automata theory makes two basic assumptions: there is a *finite state space*; and there is a *finite alphabet*. The topic of this paper is along the line of work that challenges the second assumption. One of the major drawbacks of classical finite state automata is that they do not scale well for large (or infinite) alphabets, although there are some recent developments that work around this problem by using partial representations of DFAs [1, 18, 17]. Our interest in this topic originated from regular expression analysis in modern programming languages [21] where the standard alphabet size is 2^{16}. In this context, Symbolic Finite Automata or SFAs were introduced, as an extension of classical finite state automata that addresses the alphabet problem by allowing transitions to be labeled with predicates defined in a separate alphabet theory. This separation of concerns, between on one hand the finite state graph and on the other hand the alphabet theory, raised many fundamental questions about if and how classical algorithms and techniques can be lifted to the symbolic case and what is the price and what is the payoff in doing so [8].

The main contribution of this paper is a new *minimization* algorithm for SFAs. We start by looking at generalizations of the classical minimization algorithms of DFAs by Moore and Hopcroft, called $\mathit{MinSFA}_{\mathrm{Moore}}$ and $\mathit{MinSFA}_{\mathrm{Hopcroft}}$, respectively. We then introduce the new algorithm, called *MinSFA*, and prove its correctness. The complexities of the classical DFA minimization algorithms depend linearly on the size k of the alphabet. A central question is: How is k reflected in the symbolic versions of the algorithms, because there the character domain may be infinite?

In Moore's algorithm k is due to an outer loop over characters that detects so called *distinguishing state* pairs. In $\mathit{MinSFA}_{\mathrm{Moore}}$ that loop corresponds to

* Microsoft Research Technical Report MSR-TR-2013-48

a satisfiability check. In Hopcroft's algorithm, k is due to an outer loop over characters that are used to *split* state equivalence classes in the style of divide-&-conquer. In $\mathit{MinSFA}_{\text{Hopcroft}}$ the same loop translates into a loop iterating over all *minterms* (satisfiable Boolean combinations of predicates) of the SFA. The downside of $\mathit{MinSFA}_{\text{Hopcroft}}$ compared to $\mathit{MinSFA}_{\text{Moore}}$ is that, for some alphabet theories, the number of minterms may be exponential in the number of predicates in the SFA. The new algorithm *MinSFA* does not rely on minterm construction, and thus avoids the potential exponential factor, while maintaining the divide-&-conquer style of $\mathit{MinSFA}_{\text{Hopcroft}}$. A core difference between the symbolic algorithms is *how* the alphabet theory is being used in each case. In particular, $\mathit{MinSFA}_{\text{Hopcroft}}$ needs an efficient implementation of *predicate refinement*, not needed in $\mathit{MinSFA}_{\text{Moore}}$ or *MinSFA*. In $\mathit{MinSFA}_{\text{Hopcroft}}$ we use a predicate refinement technique that relies on BDDs for fast lookup of equivalent predicates. Unlike $\mathit{MinSFA}_{\text{Moore}}$, *MinSFA* relies fundamentally on *complementation* of character predicates. Similar tradeoffs do not exist in the case of DFAs because characters do not have structure in classical automata. We also provide an evaluation of the algorithms using a sample set of regexes.

2 Effective Boolean algebras and SFAs

An *effective Boolean algebra* $\mathcal{A}$ has components $(\mathfrak{D}, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$. $\mathfrak{D}$ is an r.e. (recursively enumerable) set of *domain elements*. Ψ is an r.e. set of *predicates* closed under the Boolean connectives and $\bot, \top \in \Psi$. The *denotation function* $[\![\_]\!] : \Psi \to 2^{\mathfrak{D}}$ is r.e. and is such that, $[\![\bot]\!] = \emptyset$, $[\![\top]\!] = \mathfrak{D}$, for all $\varphi, \psi \in \Psi$, $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$, $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$, and $[\![\neg\varphi]\!] = \mathfrak{D} \setminus [\![\varphi]\!]$. For $\varphi \in \Psi$, we write $\mathit{IsSat}(\varphi)$ when $[\![\varphi]\!] \neq \emptyset$ and say that φ is *satisfiable*. $\mathcal{A}$ is *decidable* if *IsSat* is decidable.

The intuition is that such an algebra is represented programmatically as an API with corresponding methods implementing the Boolean operations and the denotation function. We are primarily going to use two such effective Boolean algebras in the examples, but the techniques in the paper are fully generic.

$\mathbf{2}^{\mathbf{bv}k}$ is the powerset algebra whose domain is the finite set $\text{BV}k$, for some $k > 0$, consisting of all nonnegative integers less than 2^k, or equivalently, all k-bit bit-vectors. A predicate is represented by a BDD of depth k.[1] The Boolean operations correspond directly to the BDD operations, $\bot$ is the BDD representing the empty set. The denotation $[\![\beta]\!]$ of a BDD β is the set of all integers n such that a binary representation of n corresponds to a solution of β.

SMT^{σ} is the decision procedure for a theory over some sort σ, say integers, such as the theory of integer linear arithmetic. This algebra can be implemented through an interface to an SMT solver. Ψ contains in this case the set of all formulas $\varphi(x)$ in that theory with one fixed free integer variable x. Here

[1] The variable order of the BDD is the reverse bit order of the binary representation of a number, in particular, the most significant bit has the lowest ordinal.

$[\![\varphi]\!]$ is the set of all integers n such that $\varphi(n)$ holds. For example, a formula $(x \bmod k) = 0$, say div_k, denotes the set of all numbers divisible by k. Then $div_2 \wedge div_3$ denotes the set of numbers divisible by six.

Definition 1. A *symbolic finite automaton* (*SFA*) M is a tuple $(\mathcal{A}, Q, q^0, F, \Delta)$ where $\mathcal{A}$ is an effective Boolean algebra, called the *alphabet*, 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 *moves* or *transitions*.

Elements of $\mathfrak{D}_{\mathcal{A}}$ are called *characters* and finite sequences of characters, elements of $\mathfrak{D}_{\mathcal{A}}^*$, are called *words*; ϵ denotes the empty word. A move $\rho = (p, \varphi, q) \in \Delta$ is also denoted by $p \xrightarrow{\varphi}_M q$ (or $p \xrightarrow{\varphi} q$ when M is clear) where p is the *source* state, denoted $Src(\rho)$, q is the *target* state, denoted $Tgt(\rho)$, and φ is the *guard* or *predicate* of the move, denoted $Grd(\rho)$. A move is *feasible* if its guard is satisfiable. Given a character $a \in \mathfrak{D}_{\mathcal{A}}$, an *a-move* of M is a move $p \xrightarrow{\varphi} q$ such that $a \in [\![\varphi]\!]$, also denoted $p \xrightarrow{a}_M q$ (or $p \xrightarrow{a} q$ when M is clear). In the following let $M = (\mathcal{A}, Q, q^0, F, \Delta)$ be an SFA.

Definition 2. A word $w = a_1 a_2 \cdots a_k \in \mathfrak{D}_{\mathcal{A}}^*$, is *accepted at state p of M*, denoted $w \in \mathscr{L}_p(M)$, if there exist $p_{i-1} \xrightarrow{a_i}_M p_i$ for $1 \leq i \leq k$ where $p_0 = p$ and $p_k \in F$. The *language accepted by M* is $\mathscr{L}(M) \stackrel{\text{def}}{=} \mathscr{L}_{q^0}(M)$.

For $q \in Q$, we use the definitions

$$\overrightarrow{\Delta}(q) \stackrel{\text{def}}{=} \{\rho \in \Delta \mid Src(\rho) = q\}, \quad \overleftarrow{\Delta}(q) \stackrel{\text{def}}{=} \{\rho \in \Delta \mid Tgt(\rho) = q\}.$$

The definitions are lifted to sets in the usual manner. The following terminology is used to characterize various key properties of M. A state p of M is called *partial* if there exists a character a such that there is no a-move from p.

- M is *deterministic*: for all $p \xrightarrow{\varphi} q, p \xrightarrow{\varphi'} q' \in \Delta$, if $IsSat(\varphi \wedge \varphi')$ then $q = q'$.
- M is *complete*: there are no partial states.
- M is *clean*: for all $p \xrightarrow{\varphi} q \in \Delta$, p is reachable from q^0 and $IsSat(\varphi)$,
- M is *normalized*: for all $p, q \in Q$, there is at most one move from p to q.
- M is *minimal*: M is deterministic, complete, clean, normalized, and for all $p, q \in Q$, $p = q$ if and only if $\mathscr{L}_p(M) = \mathscr{L}_q(M)$.[2]

The special case is when M is *deterministic and complete*. In this case we define, for all $a \in \mathfrak{D}_{\mathcal{A}}$ and $p \in Q$, the *transition function* $\delta_M : \mathfrak{D}_{\mathcal{A}} \times Q \to Q$ as $\delta_M(a, p) \stackrel{\text{def}}{=} q$ where q is the state such that $p \xrightarrow{a}_M q$. Observer that, by using determinism, if $p \xrightarrow{\varphi_1}_M q_1$ and $p \xrightarrow{\varphi_2}_M q_2$ and $a \in [\![\varphi_1 \wedge \varphi_2]\!]$ then $q_1 = q_2$, and by completeness, there exists some q and φ such that $p \xrightarrow{\varphi}_M q$ and $a \in [\![\varphi]\!]$.

Determinization of SFAs is always possible and is studied in [20]. Completion is straightforward: if M is not complete then add a new state $q_\emptyset$ and the self-loop $q_\emptyset \xrightarrow{\top} q_\emptyset$ and for each state q add the move $(q, \bigwedge_{\rho \in \overrightarrow{\Delta}(q)} \neg Grd(\rho), q_\emptyset)$ when feasible. Observe that completion requires complementation of predicates.

[2] It is sometimes convenient to define minimality over incomplete SFAs, in which case the *dead-end* state q ($q \neq q^0$ and $\mathscr{L}_q(M) = \emptyset$) is eliminated if it is present.

Normalization is obvious: if there exist states p and q and two distinct transitions $p \xrightarrow{\varphi} q$ and $p \xrightarrow{\psi} q$ then replace these transitions with the single transition $p \xrightarrow{\varphi \vee \psi} q$. This does clearly not affect $\mathscr{L}_p(M)$ for any p.

We always assume that M is clean. Cleaning amounts to running standard forward reachability that keeps only reachable states, and eliminates infeasible moves. Observe that having infeasible moves $p \xrightarrow{\bot} q$ is semantically useless and may cause unnecessary state space explosion.

It is important to show that minimality of SFAs is in fact well-defined in the sense that minimal SFAs are unique up to renaming of states and equivalence of predicates. To do so we use the following construction.

Assume $M = (\mathcal{A}, Q, q^0, F, \Delta)$ to be deterministic and complete. Let $\Sigma_{\mathcal{A}}$ denote the *first-order language* that contains the *unary relation symbol* $\bar{F}$ and the *unary function symbol* $\bar{a}$ for each $a \in \mathfrak{D}_{\mathcal{A}}$. We define the *$\Sigma_{\mathcal{A}}$-structure of M*, denoted by $\mathbf{M}$, to have the universe Q, and the interpretation function:

$$\bar{F}^{\mathbf{M}} \stackrel{\text{def}}{=} F, \quad \forall \bar{a} \in \Sigma_{\mathcal{A}},\, p \in Q\, (\bar{a}^{\mathbf{M}}(p) \stackrel{\text{def}}{=} \delta_M(a, p)).$$

Also, let

$$\begin{aligned} M(\epsilon) &\stackrel{\text{def}}{=} q^0, \\ M(w \cdot a) &\stackrel{\text{def}}{=} \delta_M(a, M(w)) \qquad \text{for } a \in \mathfrak{D}_{\mathcal{A}} \text{ and } w \in \mathfrak{D}^*_{\mathcal{A}}. \end{aligned}$$

In other words, $M(w)$ is the state reached in M for the word $w \in \mathfrak{D}^*_{\mathcal{A}}$.

Theorem 1. *If M and N are minimal SFAs over the same alphabet $\mathcal{A}$ such that $\mathscr{L}(M) = \mathscr{L}(N)$, then $\mathbf{M}$ and $\mathbf{N}$ are isomorphic $\Sigma_{\mathcal{A}}$-structures.*

Proof. Assume M and N to be minimal SFAs over $\mathcal{A}$ such that $\mathscr{L}(M) = \mathscr{L}(N)$. We define $\imath : \mathbf{M} \cong \mathbf{N}$ as follows:

$$\forall w \in \mathfrak{D}^*_{\mathcal{A}} (\imath(M(w)) \stackrel{\text{def}}{=} N(w)).$$

To show that $\imath$ is well-defined as a function, observe first that all states of M correspond to some w because M is clean. Second, we prove that for all words v and w, if $M(v) = M(w)$ then $N(v) = N(w)$. Fix $v, w \in \mathfrak{D}^*_{\mathcal{A}}$ such that $M(v) = M(w)$. Then, for all $u \in \mathfrak{D}^*_{\mathcal{A}}$,

$$\begin{aligned} u \in \mathscr{L}_{N(v)}(N) &\quad\Leftrightarrow\quad v \cdot u \in \mathscr{L}(N) \\ &\overset{(\text{by } \mathscr{L}(M)=\mathscr{L}(N))}{\Leftrightarrow} v \cdot u \in \mathscr{L}(M) \\ &\Leftrightarrow\quad u \in \mathscr{L}_{M(v)}(M) \\ &\overset{(\text{by } M(w)=M(v))}{\Leftrightarrow} u \in \mathscr{L}_{M(w)}(M) \\ &\Leftrightarrow\quad w \cdot u \in \mathscr{L}(M) \\ &\overset{(\text{by } \mathscr{L}(M)=\mathscr{L}(N))}{\Leftrightarrow} w \cdot u \in \mathscr{L}(N) \\ &\Leftrightarrow\quad u \in \mathscr{L}_{N(w)}(N) \end{aligned}$$

So $\mathscr{L}_{N(v)}(N) = \mathscr{L}_{N(w)}(N)$ and thus $N(v) = N(w)$ by minimality of N. So $\imath$ is well-defined as a function. By switching the roles of M and N we also get the opposite direction. Thus,

$$(*) \qquad \forall v, w \in \mathfrak{D}_{\mathcal{A}}^*(M(v) = M(w) \Leftrightarrow N(v) = N(w))$$

Next, we show that $\imath$ is an isomorphism.

We show that $\imath$ is bijective: $\imath$ is onto because N is clean, i.e., each state of N corresponds to $N(w)$ for some word w; $\imath$ is into because if $M(v) \neq M(w)$ then, by $(*)$, $\imath(M(v)) = N(v) \neq N(w) = \imath(M(w))$.

Finally, we need to show that $\imath$ is an embedding of $\mathbf{M}$ into $\mathbf{N}$, i.e., for all $p \in Q_M$, $p \in \bar{F}^{\mathbf{M}} \Leftrightarrow \imath(p) \in \bar{F}^{\mathbf{N}}$, and for all $p \in Q_M$ and $a \in \mathfrak{D}_{\mathcal{A}}$, $\imath(\bar{a}^{\mathbf{M}}(p)) = \bar{a}^{\mathbf{N}}(\imath(p))$. Let $p \in Q_M$. Let w be any word such that $p = M(w)$. Then

$$\begin{aligned} p \in \bar{F}^{\mathbf{M}} &\Leftrightarrow M(w) \in \bar{F}^{\mathbf{M}} \Leftrightarrow w \in \mathscr{L}(M) \Leftrightarrow w \in \mathscr{L}(N) \\ &\Leftrightarrow N(w) \in \bar{F}^{\mathbf{N}} \Leftrightarrow \imath(M(w)) \in \bar{F}^{\mathbf{N}} \Leftrightarrow \imath(p) \in \bar{F}^{\mathbf{N}}. \end{aligned}$$

and, for any $a \in \mathfrak{D}_{\mathcal{A}}$,

$$\begin{aligned} \imath(\bar{a}^{\mathbf{M}}(p)) &= \imath(\bar{a}^{\mathbf{M}}(M(w))) = \imath(\delta_M(a, M(w))) = \imath(M(w \cdot a)) = N(w \cdot a) \\ &= \delta_N(a, N(w)) = \bar{a}^{\mathbf{N}}(N(w)) = \bar{a}^{\mathbf{N}}(\imath(M(w))) = \bar{a}^{\mathbf{N}}(\imath(p)) \end{aligned}$$

Thus $\mathbf{M}$ and $\mathbf{N}$ are isomorphic. □

The theorem implies that minimal SFAs are unique up to renaming of states and up to equivalence of predicates due to normalization. We use the following definition.

Definition 3. Two states $p, q \in Q$ are *M-equivalent*, $p \equiv_M q$, when $\mathscr{L}_p(M) = \mathscr{L}_q(M)$.

It is clear that $\equiv_M$ is an equivalence relation. If $\equiv$ is an equivalence relation over Q, then for $q \in Q$, $q_{/\equiv}$ denotes the equivalence class containing q, for $X \subseteq Q$, $X_{/\equiv}$ denotes $\{q_{/\equiv} \mid q \in X\}$, and $M_{/\equiv}$ denotes the SFA:

$$M_{/\equiv} \stackrel{\text{def}}{=} (\mathcal{A}, Q_{/\equiv}, q^0_{/\equiv}, F_{/\equiv}, \{(p_{/\equiv}, \bigvee_{(p,\varphi,q)\in\Delta} \varphi, q_{/\equiv}) \mid p, q \in Q, \exists\varphi((p, \varphi, q) \in \Delta)\})$$

Observe that $M_{/\equiv}$ is, by construction, normalized. We need the following theorem that shows that minimization of SFAs preserves their intended semantics.

Theorem 2. *Let M be a clean, complete and deterministic SFA. Then $M_{/\equiv_M}$ is minimal and $\mathscr{L}(M) = \mathscr{L}(M_{/\equiv_M})$.*

Proof. Let $\equiv$ be $\equiv_M$. Clearly, $M_{/\equiv}$ is clean and complete because M is clean and complete. To show determinism, let $\mathbf{p} \xrightarrow{a}_{M_{/\equiv}} \mathbf{q}_1$ and $\mathbf{p} \xrightarrow{a}_{M_{/\equiv}} \mathbf{q}_2$. Take $p_1, p_2 \in \mathbf{p}$, $q_1 \in \mathbf{q}_1$ and $q_2 \in \mathbf{q}_2$ such that $p_1 \xrightarrow{a}_M q_1$ and $p_2 \xrightarrow{a}_M q_2$. Since

$\mathscr{L}_{p_1}(M) = \mathscr{L}_{p_2}(M)$ and M is deterministic it follows that $\mathscr{L}_{q_1}(M) = \mathscr{L}_{q_2}(M)$ i.e., $\mathbf{q_1} = \mathbf{q_2}$. Thus,

$$(*) \qquad \forall a \in \mathfrak{D}_{\mathcal{A}}, p \in Q_M(\delta_{M_{/\equiv}}(a, p_{/\equiv}) = \delta_M(a,p)_{/\equiv}).$$

Minimality of $M_{/\equiv}$ follows by definition. Next we show by induction over the length of w that,

$$(\star) \qquad \forall w \in \mathfrak{D}^*_{\mathcal{A}}(M(w)_{/\equiv} = M_{/\equiv}(w)).$$

For $w = \epsilon$ we have $M(\epsilon) = q^0_M$ and $M_{/\equiv}(\epsilon) = q^0_{M_{/\equiv}}$ and $q^0_{M_{/\equiv}} = (q^0_M)_{/\equiv}$. For $w = v \cdot a$ where $a \in \mathfrak{D}_{\mathcal{A}}$ and $v \in \mathfrak{D}^*_{\mathcal{A}}$, we have that

$$\begin{aligned} M(v \cdot a)_{/\equiv} \quad &= \quad \delta_M(a, M(v))_{/\equiv} \overset{(\text{by } (*))}{=} \delta_{M_{/\equiv}}(a, M(v)_{/\equiv}) \\ &\overset{(\text{by IH})}{=} \delta_{M_{/\equiv}}(a, M_{/\equiv}(v)) = M_{/\equiv}(v \cdot a). \end{aligned}$$

It follows that, for all words $w \in \mathfrak{D}^*_{\mathcal{A}}$, $w \in \mathscr{L}(M)$ iff $M(w) \in F_M$ iff $M(w)_{/\equiv} \in F_{M_{/\equiv}}$ iff (by $(\star)$) $M_{/\equiv}(w) \in F_{M_{/\equiv}}$ iff $w \in \mathscr{L}(M_{/\equiv})$. Thus $\mathscr{L}(M) = \mathscr{L}(M_{/\equiv})$. □

Theorems 1 and 2 are classical theorems lifted to arbitrary (possibly infinite) alphabets. Theorem 2 implies that SFAs have equivalent minimal forms that, by Theorem 1 are unique up to relabeling of states and modulo equivalence of predicates in $\mathcal{A}$, and in particular have minimal number of states because, trivially, $|Q_{/E}| \leq |Q|$ for all Q and all equivalence relations E over Q.

3 Moore's algorithm over symbolic alphabets

Moore's minimization algorithm [15] of DFAs (also due to Huffman [11]) is also commonly known as the *standard* algorithm. The idea can be lifted to SFAs M as follows. Initially, let E be the binary relation $(F \times F) \cup (F^c \times F^c)$ where $F^c \overset{\text{def}}{=} Q \setminus F$. Repeat the following step until E does not change: remove (p, q) from E if there exist moves $(p, \varphi, p'), (q, \psi, q') \in \Delta$ where $(p', q') \notin E$ and $\varphi \wedge \psi$ *is satisfiable*.[3] This process clearly terminates. The resulting relation E is the equivalence relation $\equiv_M$ and the SFA $M_{/E}$ is therefore minimal. We refer to this algorithm as $\mathit{MinSFA}_{\text{Moore}}$. Observe that $\mathit{MinSFA}_{\text{Moore}}$ checks only *satisfiability of conjunctions* of conditions and does not depend on the full power of the alphabet algebra, in particular *complementation* is not used (if the initial completion of M is viewed as a separate preprocessing step). The latter is in contrast to a direct generalization of Hopcroft's algorithm, discussed next.

[3] In a concrete implementation, for representing E, a specific data structure can be used for marking pairs of *inequivalent* states, as in the case of DFAs [10, Section 3.4].

```
1  Minterms_A(ψ̄) ≝
2    tree := new Tree(⊤_A, null, null);
3    foreach (ψ in ψ̄) tree.Refine(ψ);
4    return Leaves(tree);                    //return the set of all the leaf predicates
5
6    class Tree
7      Predicate φ; Tree left; Tree right;
8      Refine(ψ) ≝
9        if (IsSat_A(φ ∧_A ψ) and IsSat_A(φ ∧_A ¬_A ψ))  //refinement wrt ψ is possible
10         if (left = null)                  //if this node is a leaf then split it into two parts
11           left := new Tree(φ ∧_A ψ,null,null);          //leaf for overlap [[φ]] ∩ [[ψ]]
12           right := new Tree(φ ∧_A ¬_A ψ,null,null);  //leaf for difference [[φ]] \ [[ψ]]
13         else left.Refine(ψ); right.Refine(ψ);          //else refine subtrees recursively
```

Fig. 1. Abstract partition refinement algorithm of $\bar{\psi} \subseteq \Psi_{\mathcal{A}}$ modulo $\mathcal{A}$.

4 Hopcroft's algorithm over symbolic alphabets

Hopcroft's algorithm [9] for minimizing DFAs is based on *partition refinement* of states. The initial partition of the state space is into two parts: *final* states and *nonfinal* states, i.e., $\mathcal{P} = \{F, Q \setminus F\}$. Here we assume that the SFA M is complete and nontrivial, so that both F and $Q \setminus F$ are nonempty. The partition $\mathcal{P}$ induces the equivalence relation $\equiv_{\mathcal{P}}$ (or $\equiv$ when $\mathcal{P}$ is clear) over Q such that $Q_{/\equiv} = \mathcal{P}$. At a high level, $\mathcal{P}$ is refined as follows. Suppose there exist parts $P, R \in \mathcal{P}$ and a character $a \in \mathfrak{D}_{\mathcal{A}}$ such that some a-move from P goes into R and some a-move from P goes into R^c $(Q \setminus R)$. Then the (a, R)-*split of* P is $\{P_1, P_2\}$ where P_1 (resp. P_2) is the set of all $p \in P$ from which there is an a-move into R (resp. R^c); P is replaced in $\mathcal{P}$ by P_1 and P_2. Observe that the (a, R)-split is well-defined by determinism and completeness of M. The following invariant is maintained by splitting.

Lemma 1. *For all $p, q \in Q$, if $p \not\equiv q$ then $\mathscr{L}_p(M) \neq \mathscr{L}_q(M)$.*

The splitting is repeated until no further splits are possible, at which point the following property holds.

Lemma 2. *For all $p, q \in Q$, if $p \equiv q$ then $\mathscr{L}_p(M) = \mathscr{L}_q(M)$.*

In addition to the state partition, in the symbolic case, we also use *predicate partition refinement* that is a *symbolic partition refinement of the alphabet domain*. Predicate partition refinement builds a set of *minterms* that are minimal satisfiable Boolean combinations of all guards that occur in the SFA. The algorithm is shown in Figure 1. It uses a binary tree whose leaves define the partition. Initially the tree is the leaf $\top$. Each time a predicate ψ is used to refine the tree it may cause splitting of its leaves into finer predicates.

```
1  MinSFA_Hopcroft(M = (A, Q, q^0, F, Δ)) ≝
2    P := {F, Q \ F};                                          //initial partition
3    W := {if (|F| ≤ |Q \ F|) then F else Q \ F};
4    Ψ := Minterms_A({Grd(ρ) | ρ ∈ Δ});                        //compute the minterms
5    while (W ≠ ∅)                                             //iterate over unvisited parts
6      R := choose(W); W := W \ {R};
7      foreach (ψ in Ψ)                                        //iterate over all minterms
8        S := δ^{-1}(ψ, R);                                    //all states leading into R for given minterm
9        while (exists (P in P) where P ∩ S ≠ ∅ and P \ S ≠ ∅)
10         ⟨P, W⟩ := Split_{P,W}(P, P ∩ S, P \ S);             //split P into two parts
11   return M_{/≡P};
12
13   Split_{P,W}(P, P_1, P_2) ≝
14     ⟨(P \ {P}) ∪ {P_1, P_2},                                //refine P
15     if (P ∈ W) then (W \ {P}) ∪ {P_1, P_2}                  //add both parts to W
16     else W ∪ {if (|P_1| ≤ |P_2|) then P_1 else P_2}⟩ //add only the smaller part to W
```

Fig. 2. Hopcroft's minimization algorithm lifted to deterministic SFAs. M is assumed to be clean, complete, and nontrivial ($F \neq \emptyset$ and $Q \setminus F \neq \emptyset$).

Example 1. Consider the alphabet algebra $\mathbf{2}^{\mathrm{BV7}}$ (ASCII characters). We use standard regex notation for character classes. Suppose that the following two guards occur in the given SFA: `\w` ($[\![\texttt{\textbackslash w}]\!] = [\![\texttt{[a-zA-Z0-9\_]}]\!]$) and `\d` ($[\![\texttt{\textbackslash d}]\!] = [\![\texttt{[0-9]}]\!]$). Then the value of *tree* in $Minterms_{\mathbf{2}^{\mathrm{BV7}}}(\{\texttt{\textbackslash w}, \texttt{\textbackslash d}\})$ in line 4 in Figure 1 is either the first of the two trees below if `\w` is selected first in the loop of line 3, or else the second tree (note that $[\![\texttt{\textbackslash d}]\!] \subsetneq [\![\texttt{\textbackslash w}]\!]$ so $[\![\texttt{\textbackslash d} \wedge \texttt{\textbackslash w}]\!] = [\![\texttt{\textbackslash d}]\!]$):

The minterms are the leaves `\d`, `\w` ∧ ¬`\d` and ¬`\w`. □

Given a predicate $\psi \in \Psi_{\mathcal{A}}$ and a state $p \in Q$, define:

$$\begin{aligned}
\boldsymbol{\delta}(\psi, p) &\stackrel{\text{def}}{=} \{\, Tgt(\rho) \mid \rho \in \overrightarrow{\Delta}(p),\ IsSat(Grd(\rho) \wedge \psi)\} \\
\boldsymbol{\delta}^{-1}(\psi, p) &\stackrel{\text{def}}{=} \{Src(\rho) \mid \rho \in \overleftarrow{\Delta}(p),\ IsSat(Grd(\rho) \wedge \psi)\} \\
\boldsymbol{\delta}^{-1}(\psi, P) &\stackrel{\text{def}}{=} \bigcup_{p \in P} \boldsymbol{\delta}^{-1}(\psi, p) \quad (\text{for } P \subseteq Q)
\end{aligned}$$

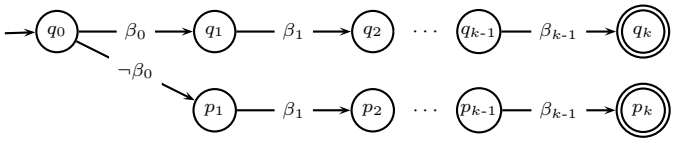
Let $Minterms(M) \stackrel{\text{def}}{=} Minterms_{\mathcal{A}}(\bigcup_{\rho \in \Delta} Grd(\rho))$. The following proposition implies that all characters that occur in one minterm are indistinguishable.

Proposition 1. *For all $\psi \in Minterms(M)$ and $p \in Q$, $|\boldsymbol{\delta}(\psi, p)| = 1$.*

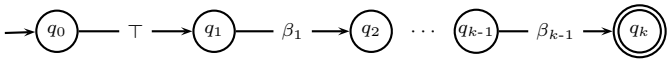
We can therefore treat $\boldsymbol{\delta}$ as a *function* from $\mathit{Minterms} \times Q$ to Q and reduce minimization of the SFA to minimization of the DFA with alphabet *Minterms* and transition function $\boldsymbol{\delta}$. In particular, we may use Hopcroft's algorithm. We refer to the resulting algorithm by $\mathit{MinSFA}_{\mathrm{Hopcroft}}$. Such an algorithm is shown in Figure 2.

One drawback with $\mathit{MinSFA}_{\mathrm{Hopcroft}}$ is that, in the worst case, the number of minterms is exponential in the number of guards occurring the SFA. The following example illustrates a worst case scenario.

Example 2. Let the character domain be nonnegative integers $< 2^k$. Suppose $\beta_i(x)$ is a predicate that is true for x iff the i'th bit of the binary representation of x is 1, e.g. $\beta_3(8)$ is true and $\beta_3(7)$ is false. Predicate β_3 can be defined as $\neg((x\&8) = 0)$, provided that, besides equality, the bitwise-and operator & is a built-in function symbol of $\Psi_{\mathcal{A}}$ (e.g., consider the bitvector theory of an SMT solver). Similarly, we may also use the algebra $\mathbf{2}^{\mathrm{BV}k}$, where the size of the concrete BDD representation for β_i is linear in k, it has one node that is labeled by i and whose left child (case bit is 0) is false and whose right child (case bit is 1) is true. The point is that that predicates are small (essentially constant) in size. Consider the following SFA M_k with such an alphabet $\mathcal{A}$.



Then $\mathit{Minterms}(M_k) = \mathit{Minterms}_{\mathcal{A}}(\{\neg\beta_i, \beta_i\}_{i<k}) = \{\hat{n}\}_{n<2^k}$ has 2^k elements. where $[\![\hat{n}]\!] = \{n\}$. For example, suppose $k = 3$, then $[\![\beta_2 \wedge \neg\beta_1 \wedge \beta_0]\!] = \{5\}$. The minimal automaton is



The dead-end state $q_\emptyset$ added by completion is implicit in both SFAs. □

5 Minimization without predicate refinement

A fundamental question arises here about the necessity of the predicate partition refinement step. Example 2 above indicates that avoiding predicate refinement may provide an *exponential* reduction in complexity. We introduce a new minimization algorithm *MinSFA*, that does not require that step. *MinSFA* is shown in Figure 3. Theorem 3 is also true for $\mathit{MinSFA}_{\mathrm{Moore}}$ and $\mathit{MinSFA}_{\mathrm{Hopcroft}}$.

Theorem 3. *$\mathit{MinSFA}(M)$ is minimal and $\mathscr{L}(\mathit{MinSFA}(M)) = \mathscr{L}(M)$.*

Proof. We show first that the invariant of Lemma 1 holds. The invariant clearly holds initially. We show that it is preserved by each split.

First consider the first splitting loop in Figure 3. Fix $R \in \mathcal{P}$ and let $S = \boldsymbol{\delta}^{-1}(\top, R)$, and choose $P \in \mathcal{P}$ such that $P_1 = P \cap S \neq \emptyset$ and $P_2 = P \setminus S \neq \emptyset$. Fix

```
1  MinSFA(M = (A, Q, q^0, F, Δ)) ≝
2    P := {F, Q \ F};                                          //initial partition
3    W := {if (|F| ≤ |Q \ F|) then F else Q \ F};
4    while (W ≠ ∅)                                             //main loop
5      R := choose(W); W := W \ {R};
6      S := δ^{-1}(⊤, R);                                      //all states leading into R
7      Γ := {p ↦ ⋁_{(p,φ,_)∈←Δ(R)} φ}_{p∈S};                   //maps p to the predicate into R
8      while (exists (P in P) where P ∩ S ≠ ∅ and P \ S ≠ ∅) //first splitting loop
9        ⟨P, W⟩ := Split_{P,W}(P, P ∩ S, P \ S);               //(_, R)-split
10     while (exists (P in P) where P ∩ S ≠ ∅ and            //second splitting loop
11           exists (p1, p2 in P) where IsSat(¬(Γ(p1) ⇔ Γ(p2))))
12       a := choose([[¬(Γ(p1) ⇔ Γ(p2))]]);
13       P1 := {p ∈ P | a ∈ [[Γ(p)]]};
14       ⟨P, W⟩ := Split_{P,W}(P, P1, P \ P1);                 //(a, R)-split
15   return M_{/≡_P};
```

Fig. 3. Minimization of deterministic SFAs. M is assumed to be clean, complete and nontrivial. $Split_{\mathcal{P},W}$ is defined in Figure 2.

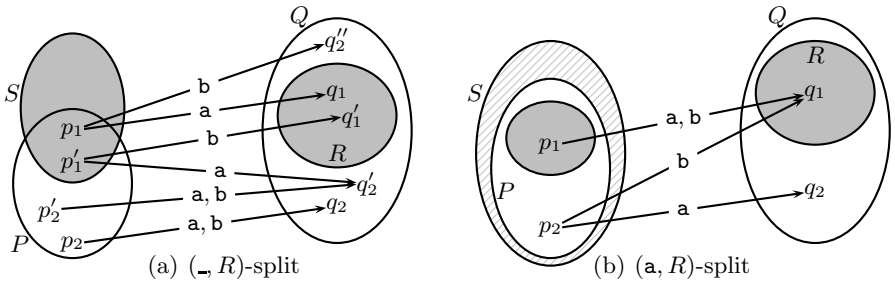


Fig. 4. Split cases of P in *MinSFA*. Suppose for example that $\mathfrak{D}_{\mathcal{A}} = \{\mathtt{a}, \mathtt{b}\}$.

$p_1 \in P_1$ and $p_2 \in P_2$. So there is a move $p_1 \xrightarrow{\varphi_1} q_1$ for some $q_1 \in R$. Let $a \in [\![\varphi_1]\!]$. So there is a move $p_2 \xrightarrow{\varphi_2} q_2$ where $a \in [\![\varphi_2]\!]$ for some $q_2 \in R^c$ because $p_2 \notin S$ and M is complete. The situation is illustrated in Figure 4(a). By using the invariant, $\mathscr{L}_{q_1}(M) \neq \mathscr{L}_{q_2}(M)$, so there is a word w such that $w \in \mathscr{L}_{q_1}(M) \Leftrightarrow w \notin \mathscr{L}_{q_2}(M)$. Thus, by using determinism of M, $a \cdot w \in \mathscr{L}_{p_1}(M) \Leftrightarrow a \cdot w \notin \mathscr{L}_{p_2}(M)$. So $\mathscr{L}_{p_1}(M) \neq \mathscr{L}_{p_2}(M)$.

Second, consider the second splitting loop. Fix P and $p_1, p_2 \in P$ that satisfy the loop condition. All parts in $\mathcal{P}$ that intersect with S must be subsets of S due to the first splitting loop, so $P \subseteq S$ and $IsSat(\Gamma(p_1))$ and $IsSat(\Gamma(p_2))$ because M is clean. The condition $IsSat(\neg(\Gamma(p_1) \Leftrightarrow \Gamma(p_2)))$ means that either $IsSat(\Gamma(p_1) \wedge \neg\Gamma(p_2))$ or $IsSat(\Gamma(p_2) \wedge \neg\Gamma(p_1))$. Assume the former case and choose $a \in [\![\Gamma(p_1) \wedge \neg\Gamma(p_2)]\!]$. By definition of Γ, we know that there is a move

$p_1 \xrightarrow{\varphi_1} q_1$ where $q_1 \in R$ such that $a \in [\![\varphi_1]\!]$. Moreover, since $a \notin [\![\Gamma(p_2)]\!]$, and $\Gamma(p_2)$ covers *all* the characters that lead from p_2 to R, there must (by completeness and determinism of M) be a move $p_2 \xrightarrow{\varphi_2} q_2$ where $q_2 \in R^c$ and $a \in [\![\varphi_2]\!]$. See Figure 4(b). It follows as above, by using $\mathscr{L}_{q_1}(M) \neq \mathscr{L}_{q_2}(M)$, that $\mathscr{L}_{p_1}(M) \neq \mathscr{L}_{p_2}(M)$.

Since each step properly refines the partition, Lemma 1 follows.

We now show that Lemma 2 holds. The proof is by way of contradiction.

(*) Assume there exists x, $P \in \mathcal{P}$, $p_1, p_2 \in P$ s.t. $x \in \mathscr{L}_{p_1}(M) \Leftrightarrow x \notin \mathscr{L}_{p_2}(M)$.

Let w be *shortest* such x. Since w cannot be ϵ, there exist a and v such that $w = a \cdot v$. So there are, by determinism and completeness of M, unique q_1 and q_2 such that $p_1 \xrightarrow{a} q_1$ and $p_2 \xrightarrow{a} q_2$. It follows that $v \in \mathscr{L}_{q_1}(M) \Leftrightarrow v \notin \mathscr{L}_{q_2}(M)$. So $q_1 \not\equiv q_2$ or else v satisfies (*) and v is shorter that w.

Consider any fixed computation of *MinSFA*. It follows from the definition of $Split_{\mathcal{P},W}$, that W is always a subset of $\mathcal{P}$ and (due to the first condition of the update to W) if W ever contains a part containing a state q then W will keep containing a part that contains q, until such a part is chosen and removed from W in line 5.

Next, we show that the following *W-invariant* must hold at all times: for all $R \in W$, $q_1 \in R \Leftrightarrow q_2 \in R$. Let $\{\iota, \hat{\iota}\} = \{1, 2\}$. Suppose, by way of contradiction, that a part R is chosen from W at some point in line 5 such that $q_\iota \in R$ and $q_{\hat{\iota}} \notin R$. So $p_\iota \in S$, with S as in line 6. There are two cases.

1. If $p_{\hat{\iota}} \notin S$ then p_ι and $p_{\hat{\iota}}$ are split apart in the first splitting loop. This contradicts that $p_1 \equiv p_2$.
2. Assume $p_{\hat{\iota}} \in S$ and consider the second splitting loop. By choice of the character a above, we know that there exist moves $p_\iota \xrightarrow{\varphi_\iota} q_\iota$ and $p_{\hat{\iota}} \xrightarrow{\varphi_{\hat{\iota}}} q_{\hat{\iota}}$ where $a \in [\![\varphi_\iota]\!]$ and $a \in [\![\varphi_{\hat{\iota}}]\!]$. It follows that $a \notin [\![\Gamma(p_{\hat{\iota}})]\!]$ (because M is deterministic and $q_{\hat{\iota}} \notin R$) while $a \in [\![\Gamma(p_\iota)]\!]$. So $a \in [\![\Gamma(p_\iota)]\!] \setminus [\![\Gamma(p_{\hat{\iota}})]\!]$ or in other words $a \in [\![\Gamma(p_\iota) \wedge \neg\Gamma(p_{\hat{\iota}})]\!]$ and so $IsSat(\neg(\Gamma(p_\iota) \Leftrightarrow \Gamma(p_{\hat{\iota}})))$ holds. Consequently, p_ι and $p_{\hat{\iota}}$ end up in distinct parts upon termination of the second splitting loop. This contradicts that $p_1 \equiv p_2$.

So initially $q_1 \in F \Leftrightarrow q_2 \in F$, or else the initial part of W violates the invariant. But now consider the point when the part containing both q_1 and q_2 is split into two parts containing q_1 and q_2 respectively. But at this point, at least one of those parts will be added to W by definition of $Split_{\mathcal{P},W}$. Thus, we have reached the desired contradiction, because the W-invariant is violated at that point.

We have shown that, upon termination of *MinSFA*(M), $\equiv_{\mathcal{P}}$ coincides with $\equiv_M$. It follows from Theorem 2 that $M_{/\equiv_{\mathcal{P}}}$ is minimal and accepts $\mathscr{L}(M)$. □

Detailed complexity analysis of *MinSFA* is outside the scope of this paper. It depends on many factors, most importantly on the representation of predicates and the complexity of the decision procedure for the alphabet. For an efficient implementation the complexity also depends on the concrete representation of $\mathcal{P}$ and W that may differ in different programming languages as well as different

```
...
var relevant = new HashSet<Block>();                    //blocks intersecting with S
foreach (var q in S) relevant.Add(Blocks[q]);
foreach (var P in relevant) {
  var P1 = new Block(S, P);                             //P1 is intersection of P with S
  if (P1.Count < P.Count) {                             //if P\S is nonempty
    foreach (var p in P1) {P.Remove(p); Blocks[p] = P1;} //P becomes P\S,
    if (W.Contains(P)) W.Push(P1);                      //if W contains P then push also P1
    else W.Push(P.Count < P1.Count ? P : P1); }}        //else push only the smaller part
...
```

Fig. 5. Concrete C# implementation of the first splitting loop of *MinSFA*.

platforms. Observe also that each splitting loop body may be executed in parallel for all P. In our concrete sequential C# implementation of *MinSFA*, parts are represented by *blocks*, that are objects of type `Block`. Each block contains a `HashSet` of states (states are integers). The work set W is a stack of blocks and the partition $\mathcal{P}$ is an array `Blocks` of blocks indexed by states. The actual C# implementation corresponding to the first splitting loop is shown in Figure 5. The second splitting loop is implemented without computing concrete witnesses a by computing a single (local) minterm, using `Gamma`, during a linear pass over nonsingleton blocks P intersecting with S that splits P into P_1 and P_2. This has semantically the same effect as the computation of P_1 with respect to some a in line 13, but does not require direct use of the denotation function of $\mathcal{A}$.[4]

6 Evaluation

As the first experiment we compared the performance of $\mathit{MinSFA}_{\text{Hopcroft}}$ and *MinSFA* over a sample set of 1700 SFAs with the alphabet $\mathbf{2}^{\text{BV16}}$ (of Unicode characters) constructed from typical regexes (taken from a public website of popular regexes). In all cases, the number of minterms turned out to be *smaller* (by a factor between 2 and 3) than the total number of all predicates, so the exponential blowup of minterms never occurred. The average ratio of $|Q_M|/|Q_{\mathit{MinSFA}(M)}|$ was 1.8, the size of Q_M ranged from a few states to 2000 states with an average of 35 states. The following is a typical regex from the sample set:

```
"[NS] \d{1,}(\:[0-5]\d){2}.{0,1}\d{0,},[EW] \d{1,}(\:[0-5]\d){2}.{0,1}\d{0,}"
```

The generated SFA uses 16 predicates (such as `\d` and `[^NS]`)[5] while there are only 7 minterms (such as `[NS]`, `[0-5]` and `[:]`). (For this regex, the determinized SFA has 47 states and the minimized SFA has 23 states.)

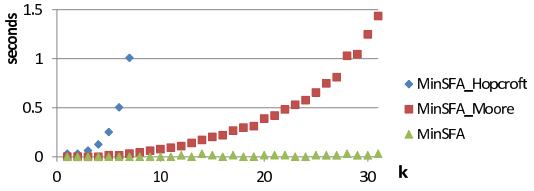
Since there is no blowup of minterms, is there a performance incentive for using *MinSFA* in this context? The total time to minimize all 1700 SFAs in the sample set took 20 seconds with $\mathit{MinSFA}_{\text{Hopcroft}}$ and 0.8 seconds with *MinSFA*, thus showing a 24x speedup and an average minimization time of 0.5 ms.

[4] Although $\mathit{IsSat}_{\mathcal{A}}(\varphi)$ is formally defined as $[\![\varphi]\!]_{\mathcal{A}} \neq \emptyset$, using SMT technology, there is a difference between *satisfiability checking* and *model generation*, the latter is typically more expensive.

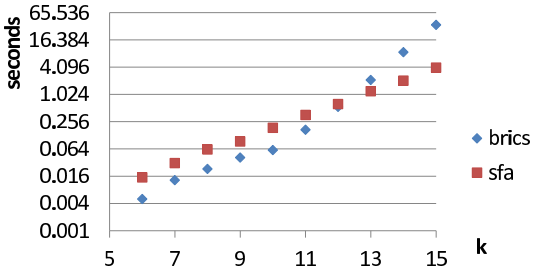
[5] We use standard regex character class notation for character predicates.

We also compared the running time of the sample set against Hopcroft's minimization algorithm in the `brics.automaton` library (version 1.11-8) that uses *symbolic integer ranges* to represent Unicode characters, but does not implement them as a Boolean algebra (since ranges are not closed under complement and union). In order to minimize platform dependencies (`brics.automaton` is written in Java) we serialized the input automata created from the regexes using a platform independent textual format that guaranteed the exact same semantics in both tools and we excluded the process overhead times as well as the times to deserialize the input automata from the serialized representation. With `brics` it took 3.2 seconds to minimize all the automata, which is 4x slower that with *MinSFA*.

In the second experiment, shown in the chart below, we considered SFAs M_k from Example 2 ranging from $k = 1$ to $k = 31$ over the alphabet SMT^{BV32} of 32-bit bitvectors with Z3. As expected, here the performance of $\mathit{MinSFA}_{\text{Hopcroft}}$ degraded exponentially.



Already for $k = 7$ the time to minimize the SFA using $\mathit{MinSFA}_{\text{Hopcroft}}$ was 1 second due to minterm generation, while it was 30ms with $\mathit{MinSFA}_{\text{Moore}}$ and below 1ms with *MinSFA*. With $\mathit{MinSFA}_{\text{Moore}}$ the time increased from 1ms for $k = 1$ to 1.4sec with $k = 31$, while for *MinSFA* the time remained below 20ms for all k.



In order to confirm that our concrete implementation of $\mathit{MinSFA}_{\text{Hopcroft}}$ is comparable to similar state-of-the-art implementations that use symbolic representations of alphabets (as in `brics`) we ran $\mathit{MinSFA}_{\text{Hopcroft}}$ for M_k, for $6 \leq k \leq 15$, using the Unicode alphabet algebra $\mathbf{2}^{\text{BV16}}$ and compared with the `brics` implementation of Hopcroft's algorithm. As above, we excluded the time to construct the SFAs M_k, and only measured the time to run the actual minimization algorithm. The timing behavior is very similar as shown in the chart.

Hopcroft's algorithm is typically implemented by pairing together characters with parts in W (see [3, Figure 3]), rather than using the foreach loop in line 7 in Figure 2. Similar construct is used in the case of partial DFAs (see [18, Figure 1]). However, this is an implementation technique that does not eliminate the foreach loop but implements it indirectly (often more efficiently) and is tailored for a concrete alphabet. Moreover, in the case of partial DFAs it avoids iterating over irrelevant characters that correspond to transitions to the dead-end state. However, the dominating factor in the above samples is the number

of nonoverlapping intervals that are precisely the minterms. Observe that the predicate β_0 requires 2^{k-1} intervals.

7 Related work

The classical minimization algorithms of DFAs have been studied and analyzed extensively from various aspects. In particular, the standard algorithm is studied in [5] where it is shown that, by a clever change of data structures, its complexity can be reduced from $O(kn^2)$ to $O(kn \log n)$. The bound $O(kn \log n)$ has been shown to be tight [4, 2]. Brzozowski [6] observed that a DFA can be minimized by reversing then determinizing then reversing, and finally determinizing again. Linear time minimization algorithms have been studied for acyclic automata [13, 16]. The book chapter [3] provides an in-depth study of the state-of-the-art in automata minimization, including the approaches mentioned above and several other ones. A new approach for minimizing *nondeterministic* (Buchi) automata has recently been studied in [14].

In the case of DFAs it matters whether the DFA to be minimized may be partial (incomplete), because it may be useful to avoid completion of DFAs with sparse transition graphs. Minimization of partial DFAs is studied in [18, 17, 1]. One direct benefit is that the algorithm complexity depends of the number of transitions rather than the alphabet size. However, the number of transitions itself is related to the alphabet size. In contrast, in a normalized SFA the number of transitions is independent of the alphabet size, it is at most n^2 where n is the number of states. One concrete difference between the minimization algorithms of complete DFAs versus partial DFAs is that the initial value of W (in Figure 2) for the partial case must contain both F and F^c [18]. We believe that similar modifications may be applied to *MinSFA*, although in the case of SFAs the benefits of the partial case are less obvious, because, unlike for DFAs, the increase in the number of transitions in the completed version is at most linear in the number of states and independent of the alphabet.

The concept of automata with predicates instead of concrete symbols was first mentioned in [22] and was first discussed in [19] in the context of natural language processing. A symbolic generalization of Moore's algorithm was first discussed in [21]. To the best of our knowledge, no other minimization algorithms have been studied for SFAs. The MONA implementation [7] provides decision procedures for monadic second-order logic. It relies on a highly-optimized BDD-based representation for automata and has seen extensive engineering effort [12]. Therefore the use of BDDs in the context of automata is not new, but is used here merely as an example of a possible Boolean algebra that seems particularly well suited for working with Unicode alphabets.

References

1. M.-P. Béal and M. Crochemore. Minimizing incomplete automata. In *Finite-State Methods and Natural Language Processing, 7th International Workshop*, pages 9–16, 2008.

2. J. Berstel, L. Boasson, and O. Carton. Hopcroft's automaton minimization algorithm and Sturmian words. In *DMTCS'2008*, pages 355–366, 2008.
3. J. Berstel, L. Boasson, O. Carton, and I. Fagnot. Minimization of automata. To appear in Handbook of Automata, 2011.
4. J. Berstel and O. Carton. On the complexity of Hopcroft's state minimization algorithm. In *CIAA'2004*, volume 3317, pages 35–44, 2004.
5. N. Blum. An $0(n \log n)$ implementation of the standard method for minimizing n-state finite automata. *Information Processing Letters*, 57:65–69, 1996.
6. 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.
7. 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 of *LNCS*. Springer, 1995.
8. P. Hooimeijer and M. Veanes. An evaluation of automata algorithms for string analysis. In *VMCAI'11*, volume 6538 of *LNCS*, pages 248–262. Springer, 2011.
9. J. Hopcroft. An $n\log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi, editor, *Theory of machines and computations, Proc. Internat. Sympos., Technion, Haifa, 1971*, pages 189–196, New York, 1971. Academic Press.
10. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.
11. D. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3–4):161–190,275–303, 1954.
12. N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA implementation secrets. *International Journal of Foundations of Computer Science*, 13(4):571–586, 2002.
13. S. L. Krivol. Algorithms for minimization of finite acyclic automata and pattern matching in terms. *Cybernetics*, 27:324–331, 1991.
14. R. Mayr and L. Clemente. Advanced automata minimization. In *POPL'13*, pages 63–74, 2013.
15. E. F. Moore. Gedanken Experiments on Sequential Machines. In *Automata Studies*, pages 129–153. Princeton U., 1956.
16. D. Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoret. Comput. Sci.*, 92:181–189, 1992.
17. A. Valmari. Fast brief practical DFA minimization. *Information Processing Letters*, 112:213–217, 2012.
18. A. Valmari and P. Lehtinen. Efficient minimization of DFAs with partial transition functions. In S. Albers and P. Weil, editors, *25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, pages 645–656, Dagstuhl, 2008.
19. G. van Noord and D. Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286, 2001.
20. M. Veanes, N. Bjørner, and L. de Moura. Symbolic automata constraint solving. In C. Fermüller and A. Voronkov, editors, *LPAR-17*, volume 6397 of *LNCS/ARCoSS*, pages 640–654. Springer, 2010.
21. M. Veanes, P. de Halleux, and N. Tillmann. Rex: Symbolic Regular Expression Explorer. In *ICST'10*, pages 498–507. IEEE, 2010.
22. B. W. Watson. Implementing and using finite automata toolkits. In *Extended finite state models of language*, pages 19–36, New York, NY, USA, 1999. Cambridge University Press.