Incremental Minimization of Symbolic Automata

Jonathan Homburg[1] and Parasara Sridhar Duggirala[2]

[1] Department of Computer Science and Engineering,
University of Connecticut, USA.
jonhom1996@gmail.com
[2] Department of Computer Science,
University of North Carolina Chapel Hill, USA.
psd@cs.unc.edu

Abstract. Symbolic automata are generalizations of finite automata that have symbolic predicates over the alphabet as transitions instead of symbols. Recently, traditional automata minimization techniques have been generalized to symbolic automata. In this paper, we generalize the incremental minimization algorithm to symbolic automata such that the algorithm can be halted at any point for obtaining a partially minimized automaton. Instead of computing the sets of equivalence classes, the incremental algorithm checks for equivalence between pairs of states and if they are equivalent, merges them into a single state. We evaluate our algorithm on SFAs corresponding to Unicode regular expressions and compare them to the state-of-the-art symbolic automata minimization implementations.

1 Introduction

As opposed to classical automata where the alphabet is given as a finite set, symbolic automata have an alphabet given by a Boolean algebra that may have an infinite domain. The transitions between states in symbolic automata are labeled with predicates in a Boolean algebra. Symbolic automata are used in regular expressions over large alphabets such as Unicode, program analysis, and satisfiability modulo theories. In [5], the notion of minimality of a symbolic automaton has been studied and extensions of classical automata minimization algorithms to symbolic setting were presented.

In this paper, we investigate a new class of minimization algorithms called incremental minimization. An incremental minimization algorithm can be interrupted at any point of time to obtain a partially minimized automaton that recognizes the same language as the input automaton. The algorithm can later be resumed with the partially minimized automaton. Upon termination, the algorithm returns the automaton with minimal number of states recognizing the same language. Hence, such an algorithm is suitable for deployment in web-services, such as spam-detection, and pattern matching in DNA sequences where server downtime for minimization is not acceptable.

This paper generalizes the incremental minimization algorithm for DFAs presented in [3] to the symbolic setting. Unlike the traditional minimization

algorithms [12, 13, 11], which compute equivalence classes over states by repeated partitioning, the algorithm in [3] merges equivalent states to create a partially minimized automata. For checking equivalence between pairs of states, the algorithm makes recursive calls and maintains a set of dependencies that need to be resolved. The primary principle in checking equivalence of two states is:

> States p and q are not distinguishable if and only if for all states p' and q' such that $p \xrightarrow{a} p'$ and $q \xrightarrow{a} q'$, p' and q' are not distinguishable.

The corresponding statement for symbolic automata is:

> States p and q are not distinguishable if and only if for all states p' and q' such that such that $p \xrightarrow{\phi} p'$ and $q \xrightarrow{\psi} q'$ where $\phi \wedge \psi$ is satisfiable, p' and q' are not distinguishable.

The primary difference between the above statements and their counterpart in [5] is that the former are useful in iteratively building equivalence classes whereas the latter are useful in partitioning a set of states to arrive at equivalence classes.

In this paper, we present an incremental minimization algorithm for symbolic automata which takes advantage of the symbolic representation of the transitions. Similar to [5], our generalization of algorithm in [3] relies on the observation that a relevant set of predicates can be computed locally rather than computing all the minterms of an automaton's predicate set.

Instead of checking equivalence for every pair of states, we use a heuristic to decrease the number of checks. We observe that two states cannot be equivalent if they have different *minimum distance to an accepting state*. Therefore, we initialize the *non-equivalence* relation between states with all pairs of states that have different minimum distance to accepting states.

This paper is organized as follows. The preliminaries are presented in Section 2. We present the incremental minimization algorithm in Section 3. The evaluation of our algorithm and its comparison with other symbolic automata minimization techniques in presented in Section 4. We discuss related work in Section 5 and present our conclusions and future work in Section 6.

2 Preliminaries

Symbolic automata have the domain of a Boolean algebra as input alphabet. An effective Boolean algebra $\mathcal{A}$ has components $(\mathcal{D}, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$. $\mathcal{D}$ is a recursively enumerable (r.e.) set of domain elements. Ψ is an r.e. set of predicates closed under Boolean operations and $\bot, \top \in \Psi$. $[\![\_]\!]$ is a denotation function (which is recursively enumerable) $[\![\_]\!] : \Psi \to 2^{\mathcal{D}}$ is such that $[\![\bot]\!] = \emptyset$, $[\![\top]\!] = \mathcal{D}$, for all $\phi, \psi \in \Psi$, $[\![\phi \vee \psi]\!] = [\![\phi]\!] \cup [\![\psi]\!]$, $[\![\phi \wedge \psi]\!] = [\![\phi]\!] \cap [\![\psi]\!]$, and $[\![\neg\phi]\!] = \mathcal{D} \setminus [\![\psi]\!]$. For $\phi \in \Psi$, we say $\mathit{IsSat}(\phi)$ if and only if $[\![\phi]\!] \neq \emptyset$ and say that ϕ is satisfiable. $\mathcal{A}$ is said to be decidable if IsSat is decidable.

In our experiments, we only deal with Boolean algebra whose domain is a bit vector of length k ($k = 16$). Each bit vector corresponds to a unique symbol in

the unicode alphabet. Predicates over this domain can be represented as BDDs (or boolean formulas) with the boolean operations corresponding to operations on BDDs (or calls to a SAT solver). Another Boolean algebra discussed in [5] is a theory over some domain σ with the various operations implemented using calls to an SMT solver.

A symbolic automaton, informally, is a finite automaton where the transitions are labeled with predicates over the domain of a Boolean algebra instead of symbols in the domain. We require these predicates originate from a Boolean algebra to ensure closure under set operations such as complement, intersection, union, etc.

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* accepting states *or* final states*, and* $\Delta \subseteq Q \times \Psi_{\mathcal{A}} \times Q$ *is a finite set of transitions.*

Elements of $\mathcal{D}_{\mathcal{A}}$ are called symbols or characters and finite sequence of characters are called *strings* (elements in $\mathcal{D}^*_{\mathcal{A}}$). ϵ denotes the empty string. A transition $\rho = (q, \phi, q') \in \Delta$ is also denoted as $q \xrightarrow{\phi} q'$ (when M is clear from the context). ρ is said to be *feasible* if ϕ is satisfiable. Given a character $a \in \mathcal{D}_{\mathcal{A}}$, an a-transition is $q \xrightarrow{\phi} q'$ such that $a \in [\![\phi]\!]$, also denoted as $q \xrightarrow{a} q'$.

Definition 2. *A string* $w = a_1 a_2 \dots a_k \in \mathcal{D}^*_{\mathcal{A}}$*, is accepted at state* q *of* $M = (\mathcal{A}, Q, q_0, F, \Delta)$*, denoted as* $w \in L_M(q)$*, if there exist states* $q_1, q_2, \dots, q_{k+1}$*, such that* $q_1 = q$*,* $\forall 1 \leq i \leq k, q_i \xrightarrow{a_i} q_{i+1}$*, and* $q_{k+1} \in F$*. The language recognized by* M *is* $L(M) = L_M(q_0)$*.*

We adopt the terminology used in [5] for symbolic automata M and present the definitions for completeness.

- M is *deterministic* if for all $p \xrightarrow{\phi} q$, $p \xrightarrow{\psi} q' \in \Delta$, if $IsSat(\phi \wedge \psi)$ then $q = q'$.
- A state q of M is said to be *partial* if there exists a symbol a such that there is no a-transition for the state q. M is said to be complete if it has no partial states.
- M is clean if for all $p \xrightarrow{\phi} q \in \Delta$, p is reachable from q_0 and $IsSat(\phi)$.
- M is normalized if for all $q, q' \in Q$, there is at most one transition from q to q'.
- Given a deterministic, complete, clean, and normalized M, it is said to be minimal if for all $q, q' \in Q$, $q = q'$ if $L_M(q) = L_M(q')$.

For a deterministic and complete symbolic automaton, we denote the transition function as $\delta_M : Q \times \mathcal{D}_{\mathcal{A}} \to Q$ such that for all $a \in \mathcal{D}_{\mathcal{A}}$ and $q \in Q$, $\delta_M(q, a) \triangleq q'$, where $q \xrightarrow{a} q'$. We drop the subscript when the automaton is clear from the context. In the rest of the paper, we assume that all the automata that we consider are deterministic, complete, clean, and normalized. Steps for obtaining such an automaton have been discussed in [5].

Definition 3. *Given M and $q \in Q$, we define the distance of the state from the accepting set of states as $dist(q) = min\{\ |w| \mid w \in L_M(q)\}$ if $L_M(q) \neq \emptyset$ and $|w|$ represents the length of string w. $dist(q) = \infty$ otherwise.*

We now present an equivalence relation on states of an automaton M and define the minimal automaton using the equivalence relation described in [5].

Definition 4. *Given SFA M, two states q and q' are said to be equivalent, $q \equiv_M q'$ if and only if $L_M(q) = L_M(q')$.*

A couple of trivial observations about $\equiv_M$:

1. $\equiv_M$ is an equivalence relation.
2. q and q' are not equivalent if $dist(q) \neq dist(q')$.

Given any equivalence relation $\equiv$ over the states Q, for any $q \in Q$, $q_\equiv$ is the equivalence class containing q, for $X \subseteq Q$, $X_\equiv = \{\ q_\equiv \mid q \in X\}$, and the corresponding SFA is $M_\equiv \triangleq (\mathcal{A}, Q_\equiv, q_{0\equiv}, F_\equiv, \Delta_\equiv)$ where

$$\Delta_\equiv \triangleq \{(q_\equiv, \bigvee_{(q,\psi,q')\in\Delta} \psi, q'_\equiv) \mid q, q' \in Q, \exists \psi, (q, \psi, q') \in \Delta\}$$

Theorem 1 (Theorem 2 from [5]). *Given a clean, complete, normal, and deterministic SFA M, $M_{\equiv_M}$ is minimal and $L(M) = L(M_{\equiv_M})$.*

3 Incremental Minimization of Symbolic Automata

Typical algorithms for automata minimization attempt to construct the largest possible equivalences classes of states and iteratively refine them. These algorithms initially partition the states into one of two classes, first is the set of accepting states, and second is the set of non-accepting states. Each of these classes are partitioned further if the states in one partition can be *differentiated* from the states in the other. This partitioning continues until no two states in the same class can be differentiated and hence are equivalent. Halting the algorithm abruptly would not yield any partially minimized automaton.

In contrast, incremental minimization attempts to merge states that are provably equivalent and construct new equivalence classes. Checking the equivalence of states p and q would require proving the equivalence of all pairs of states p' and q' that are reached after every a-transition from p and q respectively. If the equivalence of p and q is established, then these two states are merged to form a new equivalent state. Hence, as only equivalent states are merged, halting the algorithm abruptly would yield a partially minimized automaton that accepts the same language as the input automaton.

Informally, the algorithm proceeds as follows. First, a pair of states u and v are chosen from the set of states. Then, for a given input symbol a (chosen from the alphabet), the states reached after a-transition u' and v' are identified. Next, the algorithm performs a recursive call to prove the equivalence of u' and

v'. In addition, it keeps a set *path* of all the pairs of states that are waiting to be proved equivalent. If the recursive calls returns *true*, then a different symbol a' is chosen from the alphabet and the equivalence of next states on a'-transition is checked by another recursive call. In a symbolic automata, the number of symbols can possibly be infinite, and hence, we decrease the number of recursive calls by leveraging the symbolic predicates.

```
1  Function IncrementalMinimize(M = (A, Q, q0, F, Δ)):
2      for q ∈ Q do
3          Make(q)
4      neq = {Normalize(p, q) | p ∈ Q, q ∈ Q, Dist(p) ≠ Dist(q)};
5      for p ∈ Q do
6          for q ∈ {x ∈ Q | x > p} do
7              if (p, q) ∈ neq then
8                  continue;
9              if Find(p) = Find(q) then
10                 continue;
11             equiv, path = ∅
12             if Equiv-p (p,q) then
13                 for ((p', q') ∈ equiv) do
14                     Union(p',q');
15             else
16                 for (p', q') ∈ path do
17                     neq = neq ∪ {(p', q')};
18     return JoinStates(M)
```

Algorithm 1: Algorithm for incremental minimization of symbolic automata.

The incremental minimization algorithm for a complete, deterministic, clean, normalized SFA is provided in Algorithm 1. This algorithm makes a call to a recursive procedure `Equiv-p`() that is given in Algorithm 2.

We first define a few data structures that are used in the minimization algorithm. A disjoint set data structure will be used to represent equivalence classes of states. Specifically, for n disjoint sets, the following operations will be defined:

1. `Make(`i`)`, a set containing only i will be created
2. `Find(`i`)`, returns a (consistent) identifying element for S_i, the set containing i.
3. `Union`(i, j), creates a new set S_k such that $S_k = S_i \cup S_j$ and sets S_i, S_j are destroyed

```
1  Function Equiv-p(p, q):
2      if (p, q) ∈ neq then
3          return False
4      if (p, q) ∈ path then
5          return True
6      path = path ∪ {(p, q)}
7      Out_p = {φ ∈ Ψ_A | ∃p', (p, φ, p') ∈ Δ}
8      Out_q = {ψ ∈ Ψ_A | ∃q', (q, ψ, q') ∈ Δ}
9      while Out_p ∪ Out_q ≠ ∅ do
10         Let a ∈ [[(⋁_{φ∈Out_p} φ) ∧ (⋁_{ψ∈Out_q} ψ)]]
             (p', q') = Normalize(Find(δ(p, a)), Find(δ(q, a)))
11         if p' ≠ q' and (p', q') ∉ equiv then
12             equiv = equiv ∪ {(p', q')}
13             if not Equiv-p(p', q') then
14                 return False
15             else
16                 path = path \ {(p', q')}
17         Let φ ∈ Out_p with a ∈ [[φ]]
18         Let ψ ∈ Out_q with a ∈ [[ψ]]
19         Out_p = Out_p \ {φ} ∪ {φ ∧ ¬ψ}
20         Out_q = Out_q \ {ψ} ∪ {ψ ∧ ¬φ}
21     equiv = equiv ∪ {(p, q)}
22     return True
```

Algorithm 2: Algorithm that checks equivalence of states p and q.

This algorithm for SFA minimization was adapted from [3]. There are two primary modifications. First, the *neq* relation is initialized to contain all pairs of states that have different minimal distance from accepting states, instead of just the pairs of states that contain one accepting and one non-accepting state. Second, the `Equiv-p` function (which returns true on (p, q) if and only if p, q are equivalent) given in Algorithm 2 leverages the symbolic nature of the predicates over the transitions. The usage of data structures *equiv* and *path* – *equiv* tracking the pairs of equivalent states discovered and *path* tracking the path through the sets of pairs of states – is similar to the algorithm presented in [3].

Note that we assume there exists an ordering on Q (i.e. $p < q$ makes sense for all $p, q \in Q$). This can be done easily by labeling each state with a unique positive integer. `Normalize` takes a pair (p, q) as input and reorders it so that the first element is less than the second. `JoinStates` merges the states that share the same equivalence class (i.e. share the same disjoint set). `Dist` measures the minimum distance from a given state to an accepting state.

We will now prove the correctness of the above incremental algorithm for symbolic automata minimization.

Lemma 1. `Equiv-p` *terminates.*

Proof. First, note that there are a finite number of recursive calls to `Equiv-p`. This is because there are a finite number of pairs of states that `Equiv-p` can be called on and `Equiv-p` immediately returns if it recognizes that it has already been called on a given pair of states. So, to prove `Equiv-p` terminates, it needs only be shown that the loop over $Out_p \cup Out_q$ begininning on line 9 is finite.

During each iteration over $Out_p \cup Out_q$, we find some $\varphi \in Out_p$ and $\psi \in Out_q$ such that there exists an $a \in \mathcal{D}_\mathcal{A}$ with $a \in [\![\varphi \wedge \psi]\!]$. Later, during the same iteration, we replace φ in Out_p with $\varphi \wedge \neg\psi$ and ψ in Out_q with $\psi \wedge \neg\varphi$. These new predicates denote strictly smaller subsets of $\mathcal{D}_\mathcal{A}$ (because a does not satisfy either predicate). If $\mathcal{D}_\mathcal{A}$ is finite, this is enough to ensure the loop terminates. Otherwise, if $\mathcal{D}_\mathcal{A}$ is infinite, it needs to be proven that for all $\varphi \in Out_p$, there exist some finite set $S \subseteq Out_q$ such that $[\![\varphi \wedge \neg(\bigvee_{\psi \in S} \psi)]\!] = \varnothing$. Because Out_p is always finite, this is sufficient to prove that the loop terminates.

Fix $\varphi \in Out_p$. Define $S = \{\psi \in Out_q \mid IsSat(\varphi \wedge \psi)\}$. Assume that $\varphi \wedge \neg(\bigvee_{\psi \in S} \psi)$ is satisfiable. Therefore, there exists some $a \in [\![\varphi]\!]$ such that $a \notin \bigvee_{\psi \in S} \psi$. We will inductively prove that this is a contradiction on every iteration of this loop such that $\varphi \in Out_p$.

During the first loop iteration, Out_q is equivalent to the predicates of the outgoing transitions of q. Because M is complete, there exists some $\psi \in Out_q$ such that $a \in [\![\psi]\!]$. So, $\varphi \wedge \psi$ is satisfiable and ψ is an element of S which is a contradiction because $a \notin [\![\bigvee_{\psi \in S} \psi]\!]$. Beyond the first iteration, assume that there exists some $\psi \in Out_q$ at the start of the iteration such that $a \in [\![\psi]\!]$. There are two cases which we must consider:

1. If ψ is not removed from Out_q during this iteration of the loop, then $\psi \wedge \varphi$ is satisfiable and $a \in [\![\psi]\!]$ which is a contradiction.
2. If ψ is removed from Out_q during this iteration, then there exists some $\varphi' \neq \varphi$ in Out_p with $IsSat(\psi \wedge \varphi')$. At the end of this iteration, $\psi \wedge \neg\varphi'$ replaces ψ in Out_q. However, because M is deterministic and $\varphi \neq \varphi'$, $a \notin [\![\varphi']\!]$. Therefore, $a \in [\![\psi \setminus \varphi']\!]$ and $\varphi \wedge (\psi \wedge \varphi')$ is satisfiable. Because $\psi \wedge \varphi' \in Out_q$, this is a contradiction.

So, for any given point in the iteration of this loop, $\varphi \in Out_p$ implies that there exists a finite set $S \subseteq Out_q$ such that $[\![\varphi \wedge \neg(\bigvee_{\psi \in S} \psi)]\!] = \varnothing$. This ensures that the loop over $Out_p \cup Out_q$ is finite. Therefore, `Equiv-p` terminates.

Lemma 2. *A call to* `Equiv-p` *from the body of* `IncrementalMinimize` *returns true if and only if the states p and q of SFA M initially passed to it are equivalent.*

Proof. `Equiv-p` returns false on (p, q) only if the pair (p, q) is contained in neq, which only contains pairs of states known to be distinguishable, or if a recursive call to `Equiv-p` returns false. In the later case, we know that p, q can not be equivalent because we have found a string $w \in \mathcal{D}_A^*$ such that $\delta(p, w)$ and $\delta(q, w)$ are known to be distinguishable.

A recursive call to `Equiv-p` returns true only if (p, q) is contained in $path$, which only occurs if a cycle of indistinguishable states is found, or if all of its recursive calls to `Equiv-p` return true. Therefore, when called from the body of

`IncrementalMinimize`, `Equiv-p` returns true only if for all $w \in \mathcal{D}^*_{\mathcal{A}}$, $\delta(p, w)$ is either known to be equivalent or is indistinguishable from $\delta(p, w)$. Therefore, p, q are equivalent.

Lemma 3. *If a call to* `Equiv-p` *from the body of* `IncrementalMinimize` *returns true, equiv contains only pairs of states (p, q) such that p and q are equivalent. If a call to* `Equiv-p` *from the body of* `IncrementalMinimize` *returns false, then path contains only pairs of states (p, q) such that p and q are distinguishable.*

Proof. *equiv* is a set of pairs of states such that for all $(p', q') \in$ *equiv* there exists some $w \in \mathcal{D}^*_{\mathcal{A}}$ with $p' = \delta(p, w)$ and $q' = \delta(q, w)$. If `Equiv-p` returns true on (p, q) then p and q are equivalent by the previous lemma. So, for all $w \in \mathcal{D}^*_{\mathcal{A}}$, $\delta(p, w)$ is equivalent to $\delta(q, w)$. Therefore, each pair of states in *equiv* contains equivalent states.

path is a set of pairs of states that initially contains (p_0, q_0), the initial arguments passed to `Equiv-p`. From that it tracks the path of `Equiv-p` in the depth first traversal of the automata's set of states. That is, for all $(p_i, q_i) \in$ *path*, either $i = 0$ or there exists $(p_{i-1}, q_{i-1}) \in$ *path* with $p_i = \delta(p_{i-1}, a)$ and $q_i = \delta(q_{i-1}, a)$ for some $a \in \mathcal{D}_{\mathcal{A}}$. If `Equiv-p` returns false on (p_0, q_0), then, every recursive call to $(p_i, q_i) \in$ *path* has returned false. Since the contents of *path* are not changed if `Equiv-p` returns false, every pair of states in *path* is distinguishable. Hence, these are added to *neq*.

Theorem 2. *Running* `IncrementalMinimize` *on M until termination returns an SFA M' such that M' is minimal and $L(M) = L(M')$.*

Proof. Consider the loop starting in line 5 in `IncrementalMinimize`. This loop checks for all normalized pairs of states p, q for equivalence (if the states have same minimum distance to accepting set of states). Each equivalence check is performed by a call to `Equiv-p`. If `Equiv-p` returns true then, from Lemma 3, every pair of states in *equiv* (including the initial arguments) are equivalent.

Hence, when the loop terminates, the equivalence check on all pairs of states is performed and all possible pairs of equivalent states would be identified. Since `IncrementalMinimize` only merges states that are proved to be equivalent (line 14), all equivalent states would be merged into the same equivalence class. Since all the states that are not merged are not-equivalent, `IncrementalMinimize` returns the minimal symbolic automata.

Our algorithm is incremental because each disjoint set only ever contains states that are known to be equivalent. So, the option to halt computation and return a partially minimized SFA comprised of the merged sets is always available.

`IncrementalMinimize` makes several calls to `Equiv-p`. From the proof of Lemma 1, it follows that at most n^2 recursive calls are made to `Equiv-p`. Each of these recursive calls would take at most k iterations where k is the number of local minterms computed in the loop starting at line 9. Further, each call to the theory solver to generate a in line 10 would take worst-case $f(k)$ time. Finally, the operations for performing `Union` and `Find` can be performed in

worst case $\alpha(n)$ where α is the inverse Ackermann's function. Hence, the worst case complexity is $O(n^2kf(k)\alpha(n))$. Notice that this is comparable to the time complexity of $O(n^2 log n \cdot f(nl))$ of the Hopcroft minimization without minterm generation in [5].

4 Evaluation

We have implemented our symbolic incremental algorithm [3] using the *Symbolic Automata Library* [4] in Java. To evaluate the performance of our algorithm and understand its properties, we have used the automata generated by parsing regular expressions acquired from [7] and initially obtained from the regular expressions library RegExLib [1] as the test suite. Our test suite consists of a nearly two thousand symbolic automata with under 400 states.

Our evaluation consists of 4 parts. First, we compare the performance of incremental symbolic minimization with the symbolic adaptations of the Moore's and Hopcroft's minimization algorithms in [5]. Second, we compare the performance of "naïve" incremental minimization with symbolic incremental minimization. Third, we provide a computational budget that is equal to the running time of the most efficient minimization algorithm and observe the minimization achieved by incremental algorithm. Lastly, we observe the fraction of time spent and compare it with the fraction of minimization achieved.

Comparison with other minimization techniques: In [5], the authors extend Moore's and Hopcroft's algorithm to symbolic domain using minterm generation and present a modified Hopcroft algorithm (i.e., without minterm generation). While Hopcroft's algorithm with minterm generation works better than Moore's for larger state spaces, Moore's algorithm outperforms in the case of larger predicates. For this purpose, we choose Moore's minterm generation and modified Hopcroft algorithm (without minterms) to serve as baseline for our comparison.

Figure 1 compares the average running time of symbolic incremental minimization algorithm to Moore's minterm algorithm and modified Hopcroft's algorithm. In comparison to Moore's algorithm, the incremental algorithm is generally quicker for automata of small size (under about 150 states). However, the minimization time for incremental algorithm grows at almost the same rate (if not more, as seen in some cases) as Moore's algorithm. Additionally, the modified Hopcroft algorithm *almost* always outperforms incremental minimization.

However, there are a few instances, where incremental minimization outperforms modified Hopcroft. These are instances where the automata is already minimized (such as automata with less number of states) or are nearly minimal (e.g. automata with 129 states has 122 equivalence classes). We believe that there are two primary reasons for this behavior. First, the number of partitions created by modified Hopcroft algorithm increases when the automata is near minimal.

[3] https://github.uconn.edu/jah12014/symbolic-automata-research
[4] https://github.com/lorisdanto/symbolicautomata

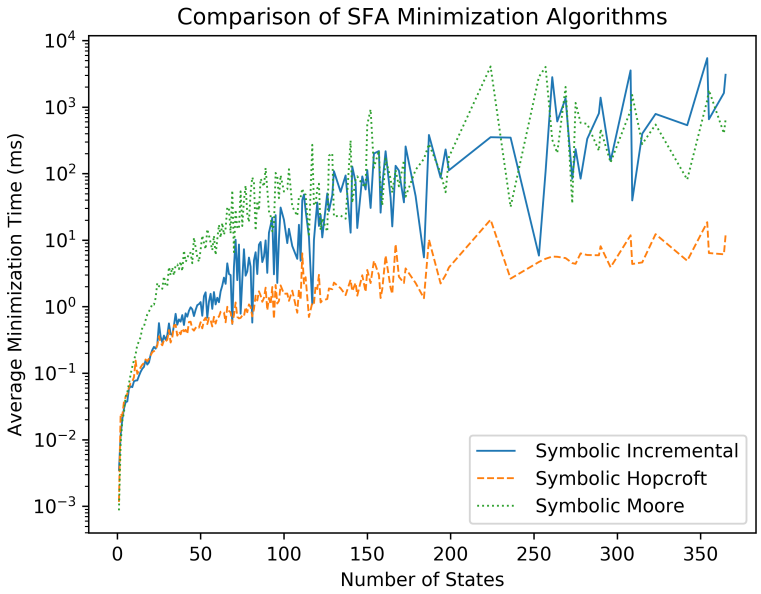

Fig. 1: Plot of the time taken by incremental minimization, Moore's algorithm with minterm generation and modified Hopcroft without minterm generation.

Second, the worst case time complexity of incremental minimization and modified Hopcroft are close.

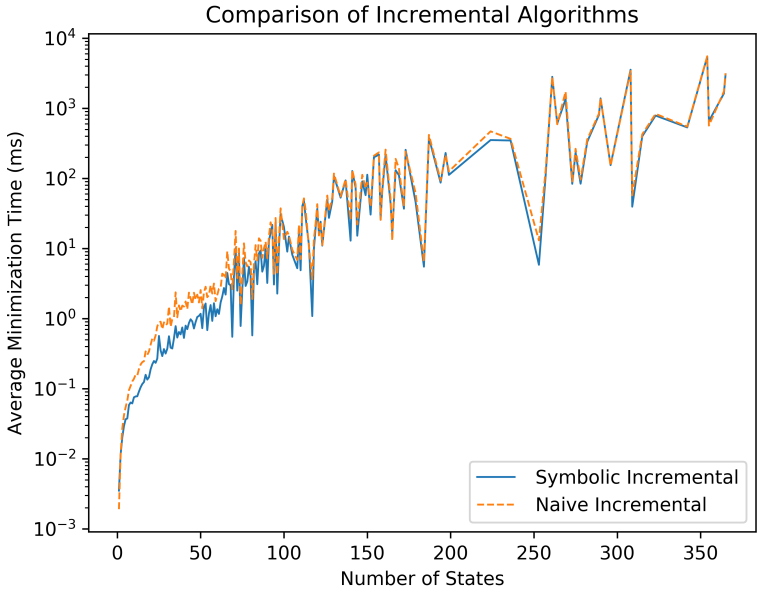

Fig. 2: Time taken by incremental minimization with minterm generation vs without minterm generation.

Comparison between naïve and symbolic incremental minimization: Figure 2 compares our symbolic incremental algorithm to a "naïve" incremental algorithm. This naive algorithm consists of computing the minterms, i.e., the maximal set of satisfiable Boolean combinations of the predicate set in Boolean algebra. It then treats the symbolic automata as if it were a classical DFA with the minterm set as its alphabet, and runs the incremental DFA minimization algorithm on it. In general, any classical algorithm can be modified to run symbolic automata in this way [8]. However, because of this upfront computational cost of minterm computation, the naïve algorithm runs noticeably slower than our symbolic algorithm for small automata. Interestingly though, as the number of states in the automata increases, both algorithms appear to converge to the same running time. This is a surprising but not unreasonable result. Intuitively, our symbolic algorithm computes the local minterms between the predicates of outgoing states while minimization is in progress. On the other hand, the naive algorithm performs all of this computation upfront.

Incremental minimization under time budget: We run the incremental minimization (both with and without minterm generation) under a time budget.

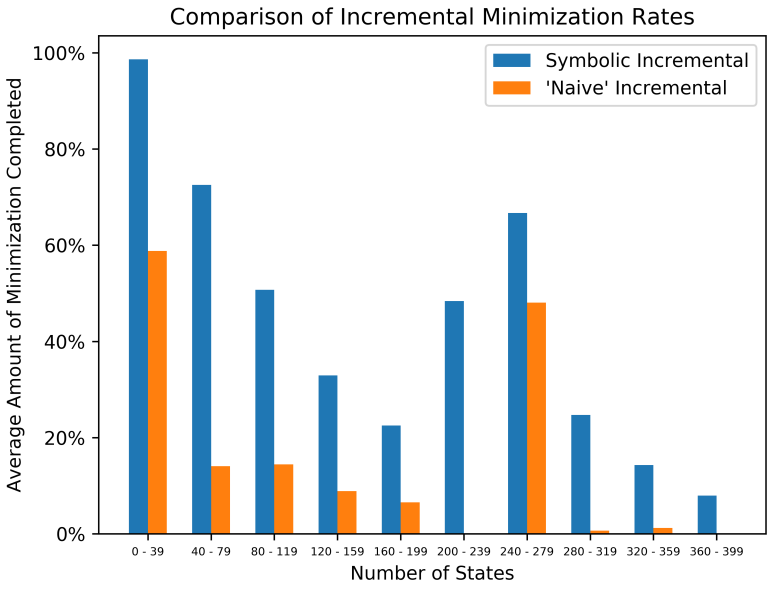

Fig. 3: Incremental minimization achieved under computational budget.

The time budget corresponds to the minimization time taken by the modified Hopcroft algorithm. When the time budget expires, the minimization algorithms are halted and the fraction of minimization achieved is reported. Figure 3 presents the fraction of minimization achieved with the budget.

The symbolic incremental minimization always outperforms the naïve incremental minimization with minterms. Although both incremental algorithms take about the same time to fully complete minimization, Figure 3 reveals that the symbolic incremental algorithm reduces the state size of the given automata significantly quicker than the naive algorithm during its early runtime. Furthermore, the number of minterms might be exponential in the number of predicates. Hence, using an incremental minimization algorithm that takes exponential time pre-processing does not capture the spirit of incremental minimization.

Time taken vs minimization achieved: To understand the nature of incremental minimization, we present a heat map of the average time (as a fraction of the total minimization time) that the algorithm took to reach a certain amount of minization in Figure 4. Blue regions correspond to less amount of time taken (0-50%) and Yellow corresponds to the more time (50-100%). Observe that for automata with less number of states (< 100), the time taken for minimization is fairly proportional to the minimization achieved. However, as the number of states increases (> 200), it takes less than 20% of time to achieve 80% minimization for majority of the automata.

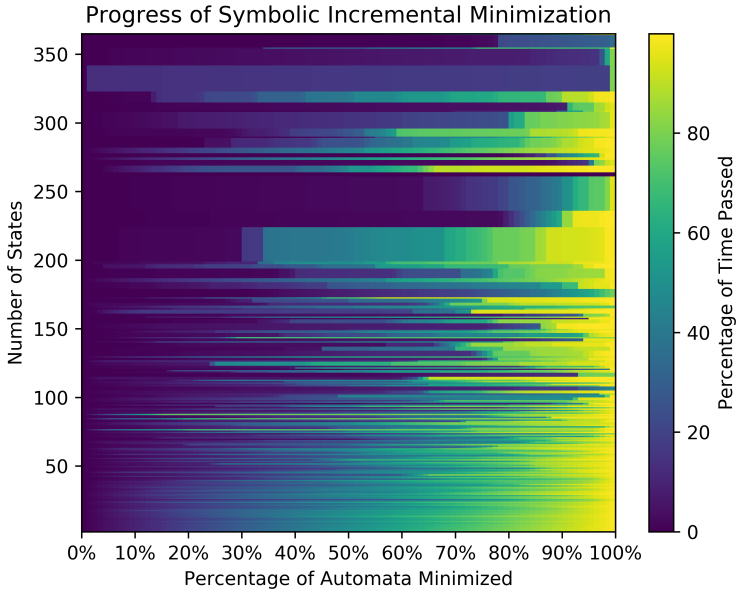

Fig. 4: Heatmap of the time taken over number of states and minimization achieved.

5 Related Work

Minimization of automata has been a very well studied topic with classical results from Huffman [12], Moore [13], Hopcroft [11], and Brzozowski [4]. The concept of automata with transitions labeled by predicates was first conceived in [17] and first studied in [15]. Moore's algorithm for DFA minimization was first extended to symbolic automata in [16]. The non-incremental algorithms that we compare our algorithm against in Figure 1 were adapted from the symbolic minimization algorithms presented in [5]. These algorithms were generalized to the computation of forward bisimulations for nondeterministic symbolic finite automata in [6]. The minimization of symbolic transducers is also studied in [14]. A good overview of the theory and applications of symbolic automata and symbolic transducers is included in [8].

An incremental algorithm for DFA minimization was first proposed by Watson [18]. However, the worst case performance of this algorithm was exponential. An efficient incremental algorithm was presented by Almeida et al. in [3] and tended to outperform Hopcroft's algorithm in empirical evaluation [2]. Our symbolic incremental algorithm was an adaptation of algorithm in [3]. An incremental hybrid of the algorithms by Hopcroft and Almeida et al. was given in [9]. To the best of our knowledge, this paper is the first in presenting an incremental algorithm for minimization of symbolic automata.

In a seminal work on algebraic properties of sequential machines [10], the authors present a notion of partitions (subsets of states) over set of states and prove that they form a lattice. The minimization algorithms are essentially various lattice traversal mechanisms that eventually reach the partition corresponding to automata with minimal number of state. Incremental minimization is a strictly upward lattice traversal mechanism, whereas traditional minimization algorithms are strictly downward lattice traversal mechanisms.

6 Conclusion and Future Work

We have extended an incremental DFA minimization algorithm to symbolic automata. For large automata, this incremental algorithm does not perform as well as the most efficient algorithms for symbolic minimization. However, unlike the other symbolic minimization algorithms, the incremental algorithm can be halted at any time to return a partially minimized automata. Additionally, our algorithm is preferable to the naïve adaptation of incremental minimization via minterm generation. Our experimental results show that, unlike the naive algorithm, the symbolic algorithm achieves the majority of minimization during its early runtime. This makes the symbolic algorithm superior as an incremental algorithm.

As a part of the future work, we would like to improve the efficiency of symbolic incremental minimization by performing memoization. We would also like to extend the incremental minimization to non-deterministic symbolic automata.

Acknowledgements: The authors would like to thank anonymous reviews for their feedback. The work done in this paper is based upon work supported by the National Science Foundation (NSF) under grant numbers CNS 1739936, 1935724. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of NSF.

References

1. Regexlib. `http://www.regexlib.com/`.
2. Marco Almeida, Nelma Moreira, and Rogério Reis. On the performance of automata minimization algorithms. In *Proceedings of the 4th Conference on Computation in Europe: Logic and Theory of Algorithms*, pages 3–14, 2007.
3. Marco Almeida, Nelma Moreira, and Rogério Reis. Incremental DFA minimisation. In *International Conference on Implementation and Application of Automata*, pages 39–48. Springer, 2010.
4. Janusz A Brzozowski. Canonical regular expressions and minimal state graphs for definite events. *Mathematical theory of Automata*, 12(6):529–561, 1962.
5. Loris D'Antoni and Margus Veanes. Minimization of symbolic automata. In *ACM SIGPLAN Notices*, volume 49, pages 541–553. ACM, 2014.
6. Loris D'Antoni and Margus Veanes. Forward bisimulations for nondeterministic symbolic finite automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 518–534. Springer, 2017.

7. Loris DAntoni. symbolicautomata. `https://github.com/lorisdanto/symbolicautomata/`. Accessed 2017-10-30.
8. Loris DAntoni and Margus Veanes. The power of symbolic automata and transducers. In *International Conference on Computer Aided Verification*, pages 47–67. Springer, 2017.
9. Pedro García, Manuel Vázquez de Parga, Jairo A Velasco, and Damián López. A split-based incremental deterministic automata minimization algorithm. *Theory of Computing Systems*, 57(2):319–336, 2015.
10. Juris Hartmanis. Algebraic structure theory of sequential machines (prentice-hall international series in applied mathematics). 1966.
11. John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In *Theory of machines and computations*, pages 189–196. Elsevier, 1971.
12. David A Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3):161–190, 1954.
13. Edward F Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.
14. Olli Saarikivi and Margus Veanes. Minimization of symbolic transducers. In *International Conference on Computer Aided Verification*, pages 176–196. Springer, 2017.
15. Gertjan van Noord and Dale Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286, 2001.
16. Margus Veanes, Peli De Halleux, and Nikolai Tillmann. Rex: Symbolic regular expression explorer. In *International Conference on Software Testing, Verification and Validation*, pages 498–507. IEEE, 2010.
17. Bruce W Watson. Implementing and using finite automata toolkits. *Natural Language Engineering*, 2(4):295–302, 1996.
18. Bruce W Watson. An incremental DFA minimization algorithm. In *International Workshop on Finite-State Methods in Natural Language Processing, Helsinki, Finland*, 2001.