Reducing Nondeterministic Finite Automata with SAT Solvers

Jaco Geldenhuys, Brink van der Merwe, and Lynette van Zijl

Department of Computer Science
Stellenbosch University, Private Bag X1, 7602 Matieland, South Africa
{jaco,abvdm,lynette}@cs.sun.ac.za

Abstract. We consider the problem of reducing the number of states of nondeterministic finite automata, and show how to encode the reduction as a Boolean satisfiability problem. This approach improves on previous work by reducing a more general class of automata. Experimental results show that it produces a minimal automaton in almost all cases and that the running time compares favourably to the Kameda-Weiner algorithm.

1 Introduction

The use of nondeterministic finite automata (NFAs) in natural language processing is well-known [15]. NFAs provide an intuitive model for many natural language problems, and in addition have potentially fewer states than deterministic finite automata (DFAs). However, in large, real-world applications, NFAs can be prohibitively large and unfortunately NFA state minimization is PSPACE-complete [8], and NP-complete for even severely restricted automata [11].

This paper considers NFA state reduction from a new viewpoint. Our goal is to solve the problem for practical settings within a limited time. We encode the NFA reduction problem as a satisfiability problem, and use a SAT solver to determine a solution that can be turned back into a reduced, equivalent NFA.

Furthermore, we are interested in generalized nondeterministic automata. A $\star$-NFAs $M = (S, \Sigma, \Delta, \hat{s}, F, \star)$ is a tuple where S is a finite set of states, Σ is a finite alphabet, $\hat{s} \in S$ is the initial state, $F \subseteq 2^S$ is a set of final state sets, $\Delta : S \times \Sigma \times S$ is the set of transitions, and $\star$ is a binary associative and commutative set operation. Define $d : S \times \Sigma \to 2^S$ by $d(s, a) = \{s' \mid (s, a, s') \in \Delta\}$, define $e : 2^S \times \Sigma \to 2^S$ by $e(X, a) = \star_{s \in X} d(s, a)$, and define $f : 2^S \times \Sigma^* \to 2^S$ by $f(S, \epsilon) = S$ and $f(S, wa) = e(f(S, w), a)$ for $w \in \Sigma^*$ and $a \in \Sigma$. A $\star$-NFA M accepts a word w if and only if $f(\{\hat{s}\}, w) \in F$. In this framework, a traditional NFA is a $\cup$-NFA. The work presented here is the first known reduction solution for this class of automata.

In the rest of the paper, we consider related work in Sect. 2, the details of the reduction algorithm are described in Sect. 3, followed by an example in Sect. 4. We consider the degree of reduction in Sect. 5, experimental results are presented in Sect. 6, and we conclude in Sect. 7.

A. Yli-Jyrä et al. (Eds.): FSMNLP 2009, LNAI 6062, pp. 81–92, 2010.

2 Related Work

We briefly mention previous approaches to NFA minimization/reduction, some of which restrict the NFAs under consideration to, for example, acyclic NFAs or unary NFAs, while others consider reduction instead of minimization, as we do. In the case of the minimization of general NFAs (without any restrictions), there has been some interest since the early 1970s. Indermark [7], Kameda and Weiner [9], Carrez [2], Kim [10], Melnikov [13], and Polák [17] all consider the problem, with Kameda and Weiner's work the classic solution. Almost all of this work is based on a so-called "universal automaton" of which the minimal NFA is a subautomaton. Polák describes some of the relationships between proposed universal automata from a more algebraic perspective.

For large practical applications, it can help to sacrifice accuracy for speed. Algorithms to find reduced but not minimal NFAs within a reasonable time have been investigated, for instance by Matz and Pothoff [12], Champarnaud and Coulon [3] and Ilie and Yu [5,6]. Another approach is to target specific subsets of NFAs that occur in practical applications. For example, the minimization problem for acyclic automata (or, equivalently, finite languages) was considered by Daciuk et al. [4], Mihov [14], Amilhastre, Janssen, and Vilarem [1].

The minimization of unary $\oplus$-NFAs (symmetric difference NFAs) is considered in [19]. Classical solutions to minimization are shown not to apply, as these techniques depend on finding the dual of a given NFA, which is often impossible for a $\oplus$-NFA, as the states do not have well-defined predecessors. Hence, the efficient minimization of $\star$-NFAs in the general case has not yet been solved, which makes the technique presented here particularly interesting.

3 Reduction Algorithm

The new reduction algorithm is shown in Fig. 1. For a given input NFA M, it starts by calculating the minimal equivalent DFA M_+. This determines the smallest size n for any NFA that could possibly be equivalent to M. The algorithm iterates over the sizes $n, n+1, n+2, \ldots$ until it either finds a small equivalent NFA or reaches the size of M (or perhaps M_+ if it is smaller).

The exact $\star$-NFA nature of M (whether it is a $\cup$-, $\cap$-, $\oplus$-NFA, or something else) is used in Step 1 to choose the appropriate determinization, and by changing some details in the construction of P in Step 5, the user can control the $\star$-NFA nature of the output NFA. The rest of this section describes Step 5 in detail.

Suppose that DFA M_+ has $m = |S_+|$ states and $a = |\Sigma|$ alphabet symbols. The final states and transitions of M_+ can be represented in tabular form:

Final	State	α_1	α_2	$\cdots$	α_a
f_1	1	z_{11}	z_{12}	$\cdots$	z_{1a}
f_2	2	z_{21}	z_{22}	$\cdots$	z_{2a}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
f_m	m	z_{m1}	z_{m2}	$\cdots$	z_{ma}

```
Algorithm. SATReduce
Input: NFA $M = (S, \Sigma, \Delta, \hat{s}, F)$
 1. determinize & minimize $M \longrightarrow$ DFA $M_+ = (S_+, \Sigma, \Delta_+, \hat{s}_+, F_+)$
 2. $n := \lceil \log_2 |S_+| \rceil$
 3. while $n < \min(|S|, |S_+|)$ do
 4.    construct symbolic NFA $M_? = (S_?, \Sigma, \Delta_?, \hat{s}_?, F_?)$ with $n = |S_?|$
 5.    construct SAT problem $P$ to describe $M_+ = determinize(M_?)$
 6.    apply a SAT solver to $P$
 7.    if $P$ is satisfiable then extract and return $M_?$
 8.    $n := n + 1$
 9. endwhile
10. return $M$ (if $n = |S|$) or $M_+$ (if $n = |S_+|$)
```

Fig. 1. The reduction algorithm

Each of the variables f_i is Boolean (either 0 or 1), and each of the variables z_{jk} is a state number in the range $0 \dots m$. When $z_{jk} = 0$ it means that there is no α_k-labelled transition leaving state j; otherwise, $(j, \alpha_k, z_{jk}) \in \Delta_+$. The exact values of these variables are known since M is given as input and Step 1 calculates a unique M_+. It now constructs an NFA $M_?$ with n states that is assumed – for the moment – to be equivalent to the input M. If the assumption is false, the SAT problem P will be unsatisfiable. No actual computation is involved in this step.

$$\begin{array}{ccc|cccc}
\text{Final} & \text{Initial} & \text{State} & \alpha_1 & \alpha_2 & \cdots & \alpha_a \\
\hline
g_1 & e_1 & 1 & X_{11} & X_{12} & \cdots & X_{1a} \\
g_2 & e_2 & 2 & X_{21} & X_{22} & \cdots & X_{2a} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
g_n & e_n & n & X_{n1} & X_{n2} & \cdots & X_{na}
\end{array} \tag{1}$$

As before, the g_i and the e_i are Boolean variables, and X_{jk} can be viewed as either a subset of $\{1 \dots n\}$ or equivalently as a bitset $X_{jk} = x_{jk1} x_{jk2} \dots x_{jkn}$, where each of the $x_{jk\ell}$ are once again Boolean variables. Note that the only "guessing" is that the value of n is large enough. The values of f_i and z_{ij} (the knowns) will be used to determine the values of g_i, e_i, and X_{jk} (the unknowns).

The next step is at the heart of the process. The algorithm implicitly determinizes $M_?$ by the subset construction:

$$\begin{array}{cc|cccc}
\text{Final} & \text{State} & \alpha_1 & \alpha_2 & \cdots & \alpha_a \\
\hline
f_1 & T_1 & U_{11} & U_{12} & \cdots & U_{1a} \\
f_2 & T_2 & U_{21} & U_{22} & \cdots & U_{2a} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
f_m & T_m & U_{m1} & U_{m2} & \cdots & U_{ma}
\end{array} \tag{2}$$

(The algorithm does not build this automaton explicitly; it is merely a mental tool for understanding its operation.) The assumption that the subset construction results in a DFA that is isomorphic to M_+, means that $M_?$ must have precisely m states, and that the f_i values are the same as those of M_+. Each T_i is a subset of $\{1 \ldots n\}$, or equivalently as a bitset $T_i = t_{i1}t_{i2} \ldots t_{in}$, and T_i corresponds to state i of M_+; this kind of renumbering of the subsets is usual after determinization. During the subset construction the values of T_i and X_{jk} are used to calculate the value of U_{ik}, but U_{ik} is also constrained by the value of z_{ik}. In particular, if $z_{ik} = p$ and $p \neq 0$, then $U_{ik} = T_p$. This relationship is the starting point of the conversion to a SAT problem.

3.1 Initial State Constraints

Assume, without loss of generality, that $\hat{s}_+ = 1$. State 1 corresponds to T_1, and can be related to the initial states of $M_?$ by $(t_{11} = e_1) \wedge (t_{12} = e_2) \wedge \ldots \wedge (t_{1n} = e_n)$. This is, however, the only role for the e_i variables, and is trivially satisfied by solution to the other constraints. We can safely omit it from the SAT problem.

3.2 Final State Constraints

If $f_i = 1$ (that is, state i of M_+ is final), then T_i must contain at least one final state of the unknown NFA $M_?$. In this case, $(t_{i1} \wedge g_1) \vee (t_{i2} \wedge g_2) \vee \ldots \vee (t_{in} \wedge g_n)$. If $f_i = 0$, then $\neg((t_{i1} \wedge g_1) \vee (t_{i2} \wedge g_2) \vee \ldots \vee (t_{in} \wedge g_n))$, and more generally, $f_i \Leftrightarrow ((t_{i1} \wedge g_1) \vee (t_{i2} \wedge g_2) \vee \ldots \vee (t_{in} \wedge g_n))$ for $i = 1, 2, \ldots, m$. Since the values of the f_i are known, it will be more convenient to not make use of the last equivalence and to write the m constraints separately.

3.3 Transition Constraints

Let $z_{jk} = p$ and assume for now that $p \neq 0$. This means that the result of the subset construction contains the entries

State	α_k
T_j	$U_{jk} = T_p$

or, equivalently,

State	α_k
$t_{j1}t_{j2} \ldots t_{jn}$	$t_{p1}t_{p2} \ldots t_{pn}$

Assume further that the algorithm is looking for a ∪-NFA. During the subset construction, state i of the NFA is included in T_p if and only if at least one of the states in T_j has a transition to state i. This constraint can be formulated as

$$(i \in T_p) \Leftrightarrow (1 \in T_j \wedge i \in X_{1k}) \vee (2 \in T_j \wedge i \in X_{2k}) \vee \ldots \vee (n \in T_j \wedge i \in X_{nk})$$

for $i = 1, 2, \ldots, n$. Written in terms of the Boolean variables, it becomes

$$t_{pi} \Leftrightarrow (t_{j1} \wedge x_{1ki}) \vee (t_{j2} \wedge x_{2ki}) \vee \ldots \vee (t_{jn} \wedge x_{nki}).$$

So each transition $z_{jk} = p$ (and $p \neq 0$) generates the following n constraints:

$$\begin{gathered}
t_{p1} \Leftrightarrow (t_{j1} \wedge x_{1k1}) \vee (t_{j2} \wedge x_{2k1}) \vee \ldots \vee (t_{jn} \wedge x_{nk1}) \\
t_{p2} \Leftrightarrow (t_{j1} \wedge x_{1k2}) \vee (t_{j2} \wedge x_{2k2}) \vee \ldots \vee (t_{jn} \wedge x_{nk2}) \\
\vdots \\
t_{pn} \Leftrightarrow (t_{j1} \wedge x_{1kn}) \vee (t_{j2} \wedge x_{2kn}) \vee \ldots \vee (t_{jn} \wedge x_{nkn}).
\end{gathered} \tag{3}$$

The constraints above apply to non-zero z_{jk} transitions, but what happens when $z_{jk} = 0$? In this case, $U_{jk} = \emptyset$. The corresponding constraints are

$$\begin{gathered}
\neg((t_{j1} \wedge x_{1k1}) \vee (t_{j2} \wedge x_{2k1}) \vee \ldots \vee (t_{jn} \wedge x_{nk1})) \\
\neg((t_{j1} \wedge x_{1k2}) \vee (t_{j2} \wedge x_{2k2}) \vee \ldots \vee (t_{jn} \wedge x_{nk2})) \\
\vdots \\
\neg((t_{j1} \wedge x_{1kn}) \vee (t_{j2} \wedge x_{2kn}) \vee \ldots \vee (t_{jn} \wedge x_{nkn})).
\end{gathered} \tag{4}$$

It is the use of disjunction in the constraints in (3) and (4) that determines the $\star$-NFA nature of the output. If the algorithm is looking for a $\cap$-NFA, all of the constraints would be identical, except that the disjunctions are replaced by conjunctions; for $\oplus$-NFA, they are replaced by exclusive-or operators.

3.4 Additional Constraints

There are two more potential sources of constraints to consider. Firstly, can $T_i = \emptyset$ for some i? The answer is "yes" if M_+ contains a sink state s that is not final and for which $(s, \alpha, s) \in \Delta_+$ for all $\alpha \in \Sigma$. The presence of such as state depends on whether Δ_+ is required to be total or not. This does not affect the algorithm, and the final state constraints will exclude solutions where $T_i = \emptyset$ for all i. Secondly, is it possible $T_i = T_j$ for $i \neq j$? Again the answer is "yes". This indicates that the value of n is too large. In fact, a $(n-1)$-state NFA can be derived directly from the output of the SAT solver by eliminating the duplicated state. However, since the algorithm in Fig. 1 starts with the smallest possible value for n and proceeds in steps of 1 state, this situation will not arise.

3.5 The Size of the SAT Problem

To accommodate SAT solvers, the constraints discussed in the previous sections must be converted to conjunctive normal form (CNF). As the second column of Table 1 shows, the number of clauses (i.e., conjuncts) grows exponentially as n grows. Fortunately, this can be mitigated by introducing auxiliary variables. For example, a constraint such as

$$a \Leftrightarrow (b_1 \wedge c_1) \vee (b_2 \wedge c_2) \vee \ldots \vee (b_l \wedge c_l)$$

can be rewritten by adding a new variable d and the constraint $d \Leftrightarrow (b_2 \wedge c_2) \vee \ldots \vee (b_l \wedge c_l)$, and multiplying out the remaining terms of the original:

$$(\neg a \vee b_1 \vee d) \wedge (\neg a \vee c_1 \vee d) \wedge (a \vee \neg b_1 \vee \neg c_1) \wedge (a \vee \neg d)$$

Table 1. Upper bounds for the number of variables and clauses

	Without new variables	With new variables
$(g_i,\ t_{jk},\ x_{jk\ell})$ vars.	$(n,\ nm,\ n^2a)$	$(n,\ nm,\ n^2a)$
Final states		
New vars.	0	$(n-1)F$
Clauses	$2^nF + 2nG$	$(4n-3)F + 2nG$
Horn	$2nG$	$(2n-1)F + 2nG$
Transitions ($\cup$-NFAs)		
New vars.	0	$n(n-1)R$
Clauses	$n(2^n+n)R - nL + n^2Z$	$n(4n-1)R - 2nL + n^2Z$
Horn	$n^2R - nL + n^2Z$	$n(2n+1)R + n^2Z$
Transitions ($\oplus$-NFAs)		
New vars.	0	$n(n-1)R + n(n-1)Z$
Clauses	$2n3^nR + n(3^n-1)Z/2$	$n(8n-5)R - 3nL + n(8n-10)Z$
Horn	unknown	$n(n+2)R + nL + n(n-4)Z$

The full details of such transformations are simple but tedious.

The values in Table 1 are upper bounds because the exact structure of M_+ may lead to equivalence among some of the constraints. In the table, m is the size of the DFA M_+, and n is the size of the "guessed" NFA $M_?$. The size of the alphabet is a. The number of final states is F, and G is the number of non-final states, so that $F + G = m$. The number of empty (zero) transitions of M_+ is Z, while R is the number of non-empty transitions. $Z + R = ma$. Amongst the R non-empty transitions there are L self-loops.

4 Example

Consider the DFA $M_+ = (S_+, \Sigma, \Delta_+, \hat{s}_+, F_+)$, where $S_+ = \{1, 2, 3, 4, 5\}$, $\Sigma = \{a, b\}$, $\hat{s}_+ = 1$, and Δ_+ and F_+ are shown in the table on the left in Fig. 2. The figure also shows the automaton in a graphical form. The calculation starts with the assumption that $M_?$ contains 3 states:

Final	Initial	State	a	b
g_1	e_1	1	$a_{11}a_{12}a_{13}$	$b_{11}b_{12}b_{13}$
g_2	e_2	2	$a_{21}a_{22}a_{23}$	$b_{21}b_{22}b_{23}$
g_3	e_3	3	$a_{31}a_{32}a_{33}$	$b_{31}b_{32}b_{33}$

For brevity, we use a_{jk} and b_{jk} instead of x_{j1k} and x_{j2k} used in (1).

Let $M_! = \mathit{determinize}(M_?)$. The aim is to construct a SAT problem that expresses the fact that $M_! = M_+$. The number of states of $M_!$ and its final states are taken directly from M_+, as are the values of the transitions (that is, the values of the U_{jk}):

Final	State	a	b
0	1	2	3
1	2	4	5
0	3	4	3
1	4	4	3
0	5	5	3

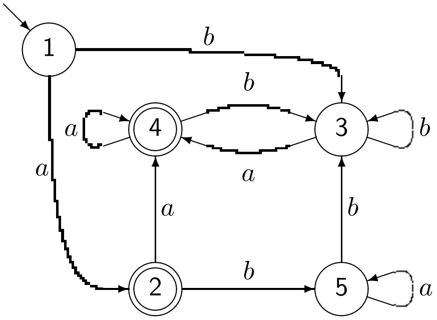

Fig. 2. Tabular and graphical representations of the automaton M_+

Final	State	a	b
0	$t_{11}t_{12}t_{13}$	$t_{21}t_{22}t_{23}$	$t_{31}t_{32}t_{33}$
1	$t_{21}t_{22}t_{23}$	$t_{41}t_{42}t_{43}$	$t_{51}t_{52}t_{53}$
0	$t_{31}t_{32}t_{33}$	$t_{41}t_{42}t_{43}$	$t_{31}t_{32}t_{33}$
1	$t_{41}t_{42}t_{43}$	$t_{41}t_{42}t_{43}$	$t_{31}t_{32}t_{33}$
0	$t_{51}t_{52}t_{53}$	$t_{51}t_{52}t_{53}$	$t_{31}t_{32}t_{33}$

The task is to describe the relation between the values of g_j, a_{jk}, b_{jk}, and t_{jk}, given the available information about M_+. As far as the final states are concerned, state T_2 is final (because $f_2 = 1$) and is marked as such by the subset construction if and only if it contains at least one final state of $M_?$. Either $1 \in T_2$ and $g_1 = 1$, or $2 \in T_2$ and $g_2 = 1$, or $3 \in T_2$ and $g_3 = 1$, or some combination of these statements is true. In other words, $(t_{21} \wedge g_1) \vee (t_{22} \wedge g_2) \vee (t_{33} \wedge g_3)$. The same is true for state T_4: $(t_{41} \wedge g_1) \vee (t_{42} \wedge g_2) \vee (t_{43} \wedge g_3)$, and similar but negated constraints can be derived from the non-final states T_1, T_3, and T_5:

$$\begin{aligned}&\neg((t_{11} \wedge g_1) \vee (t_{12} \wedge g_2) \vee (t_{13} \wedge g_3))\\&\neg((t_{31} \wedge g_1) \vee (t_{32} \wedge g_2) \vee (t_{33} \wedge g_3))\\&\neg((t_{51} \wedge g_1) \vee (t_{52} \wedge g_2) \vee (t_{53} \wedge g_3)).\end{aligned}$$

Constraints for each of the transitions of $M_!$ must now be generated. Consider

State	a
$X = t_{11}t_{12}t_{13}$	$Y = t_{21}t_{22}t_{23}$

The usual formula for the value on the right is $Y = \bigcup_{i \in X} A_i$, where $A_i = a_{i1}a_{i2}a_{i3}$. In other words,

- state $1 \in Y$ iff $(1 \in X \wedge 1 \in A_1)$, or $(2 \in X \wedge 1 \in A_2)$, or $(3 \in X \wedge 1 \in A_3)$;
- state $2 \in Y$ iff $(1 \in X \wedge 2 \in A_1)$, or $(2 \in X \wedge 2 \in A_2)$, or $(3 \in X \wedge 2 \in A_3)$;
- state $3 \in Y$ iff $(1 \in X \wedge 3 \in A_1)$, or $(2 \in X \wedge 3 \in A_2)$, or $(3 \in X \wedge 3 \in A_3)$.

This can be expressed in a propositional form as

$$\begin{aligned} t_{21} &\Leftrightarrow (t_{11} \wedge a_{11}) \vee (t_{12} \wedge a_{21}) \vee (t_{13} \wedge a_{31}) \\ t_{22} &\Leftrightarrow (t_{11} \wedge a_{12}) \vee (t_{12} \wedge a_{23}) \vee (t_{13} \wedge a_{32}) \\ t_{23} &\Leftrightarrow (t_{11} \wedge a_{13}) \vee (t_{12} \wedge a_{23}) \vee (t_{13} \wedge a_{33}) \end{aligned}$$

The other nine transitions yield 27 similar constraints. These constraints can now be given to a SAT solver to determine whether any values of g_i, t_{ij}, a_{ij}, and b_{ij} satisfy the equations. Space does not allow us to explore the results of this step, but we believe that the intention is clear.

5 Reduction v. Minimization

Our reduction algorithm assumes that for a given minimal DFA M_+ there exists an equivalent NFA $M_?$ for which $\mathcal{S}(M_?) \approx M_+$ (where $\approx$ denotes isomorphism, and $\mathcal{S}$ denotes the subset construction). This assumption holds (M_+ itself qualifies as a candidate of $M_?$), but not always for the minimal NFA equivalent to M_+, which explains why the algorithm *reduces* instead of *minimizing*.

But how often *does* the subset construction produce a minimal DFA? To answer this question we investigated randomly generated $\cup$-NFAs. (Small automata with fewer than 2 states were filtered out.) Each NFA was determinized and its size α was recorded, and then minimized and its new minimal size β was recorded. Table 2 displays the results of the "gap" $\alpha - \beta$ for different numbers of states and alphabet symbols. Each cell contains three values: the mean gap size, standard error of the mean (SEM), and number of random samples (in 1000s). For example, for 10-state 4-symbol NFAs the 99% confidence interval for the mean size difference between the determinized and minimized DFA is $5.09 \pm 2.58 \times 0.014 = 5.054 \ldots 5.126$ states (based on 230,000 samples).

Figure 3 shows a typical distribution. Interestingly, for a given alphabet size the gap appears to grow smaller as the number of states increases, although there does seem to be a small initial hump that shifts towards a larger number of states as the alphabet size increases. This is shown more clearly in Fig. 4.

Table 3 contains the mean gap sizes (only) for $\oplus$-NFAs. Surprisingly, in this case the subset construction almost always produces a minimal DFA. The SEM is vanishingly small for all of the samples, and the mean gap is < 1 when $|\Sigma| > 3$.

6 Experimental Results

We implemented both the Kameda-Weiner algorithm and our new reduction algorithm in C. The latter uses the zChaff SAT solver [20]. Preliminary time consumption results are shown in Table 4. The numbers in the table are the mean running times (in milliseconds) for the respective algorithms to minimize/reduce 1000 random NFAs. While the SEM is small for a large number of cases, the sample is relatively small, and further experiments are clearly needed, also to explain the anomalies in some of the results.

Table 2. The mean gap size, SEM, and sample size (1000s) for random ∪-NFAs

						n					
$\vert\Sigma\vert$	5	6	7	8	9	10	11	12	13	14	15
2	2.68	2.80	2.56	2.26	2.00	1.78	1.61	1.47	1.37	1.26	1.23
	0.004	0.004	0.004	0.003	0.003	0.005	0.008	0.009	0.012	0.011	0.011
	650	740	670	770	630	210	70	20	10	10	10
3	4.48	5.04	4.73	4.23	3.77	3.38	3.05	2.80	2.61	2.45	2.31
	0.007	0.007	0.007	0.005	0.006	0.009	0.014	0.022	0.021	0.020	0.019
	610	830	740	980	640	230	80	30	10	10	10
4	5.84	7.18	6.97	6.30	5.63	5.09	4.61	4.26	3.99	3.64	3.47
	0.009	0.010	0.010	0.008	0.009	0.014	0.021	0.032	0.031	0.029	0.027
	630	790	840	940	660	230	80	30	10	10	10
5	6.78	9.13	9.15	8.41	7.57	6.86	6.23	5.74	5.27	4.96	4.70
	0.011	0.011	0.011	0.011	0.012	0.018	0.028	0.043	0.078	0.076	0.075
	580	950	1080	920	650	230	80	30	10	10	10
6	7.31	10.87	11.28	10.50	9.53	8.64	7.88	7.24	6.75	6.38	6.00
	0.011	0.014	0.015	0.015	0.016	0.024	0.036	0.054	0.085	0.083	0.080
	680	860	900	840	610	220	80	30	10	10	10
7	7.53	12.35	13.28	12.57	11.47	10.43	9.58	8.88	8.22	7.66	7.24
	0.012	0.019	0.019	0.015	0.019	0.028	0.044	0.066	0.114	0.089	0.087
	590	620	780	1190	610	230	80	30	10	10	10
8	7.50	13.67	15.24	14.58	13.39	12.27	11.24	10.38	9.56	9.15	8.45
	0.012	0.018	0.018	0.015	0.022	0.034	0.052	0.078	0.121	0.118	0.115
	650	850	1110	1580	610	220	80	30	10	10	10
9	7.28	14.80	17.10	16.57	15.34	14.01	12.87	11.89	11.10	10.35	9.86
	0.012	0.021	0.021	0.021	0.026	0.039	0.097	0.089	0.145	0.125	0.122
	640	800	1120	1080	580	220	30	30	10	10	10
10	6.95	15.72	18.85	18.54	17.21	15.79	14.51	13.48	12.44	12.02	11.00
	0.012	0.024	0.028	0.022	0.031	0.045	0.067	0.101	0.167	0.149	0.144
	640	690	760	1230	530	210	80	30	10	10	10

Table 3. The mean gap size for random ⊕-NFAs

						n					
$\vert\Sigma\vert$	5	6	7	8	9	10	11	12	13	14	15
2	0.50	0.54	0.56	0.57	0.57	0.61	0.66	0.32	0.55	1.64	1.64
3	0.05	0.02	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Nonetheless, we are optimistic about these results, given that the SAT-based reduction has to contend with the overhead of the SAT solver. Moreover, the new algorithm produced minimal NFAs in almost all cases. For $n = 5$ and $|\Sigma| = 4$ it produced 14 (out of 1000) NFAs that were one state larger than the minimal. No larger gap was ever produced.

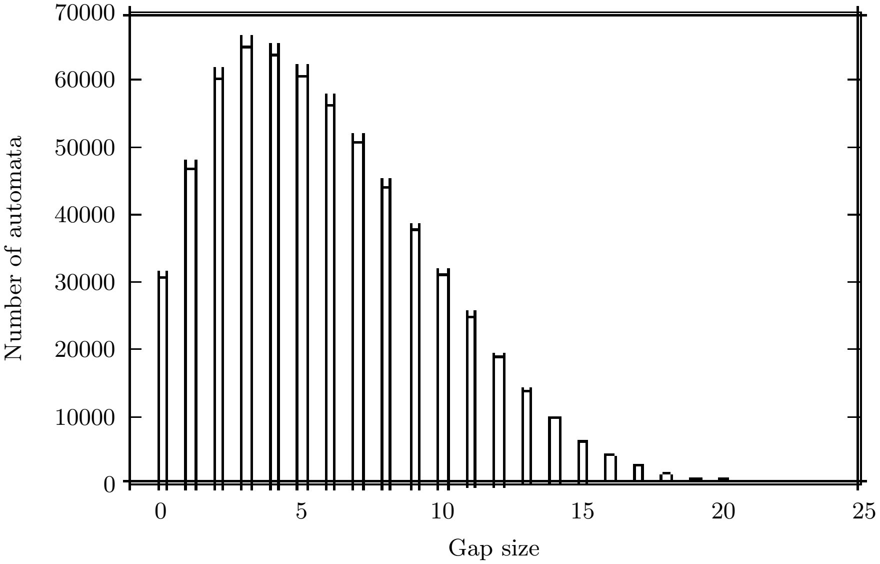

Fig. 3. Histogram of the number of ∪-NFAs per gap size for automata with $n = 5$ (states) and $|\Sigma| = 4$ (alphabet symbols)

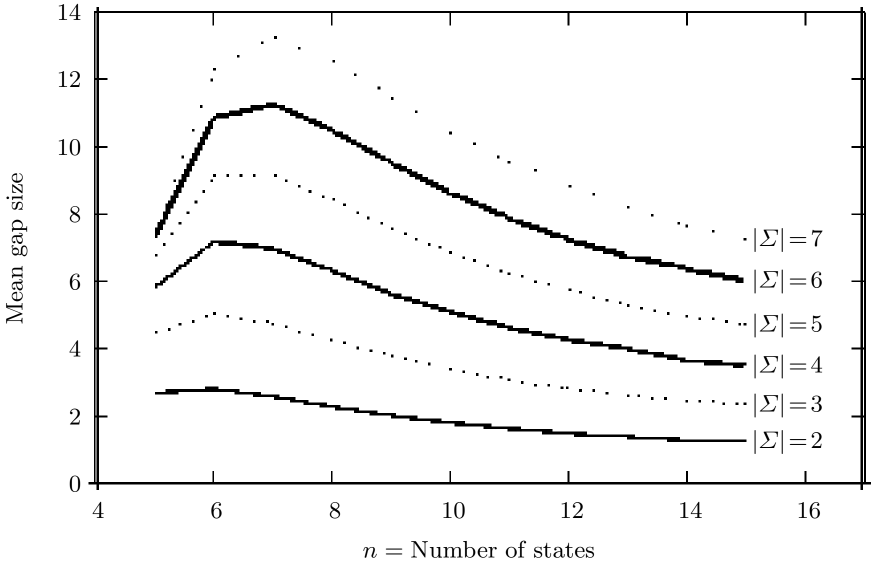

Fig. 4. Mean gap size as a function of the number of states for different alphabet sizes

Table 4. Time consumption for the Kameda-Weiner (KW) and the SAT-based reduction (SR) algorithm based on 1000 samples (in milliseconds)

	$\lvert\Sigma\rvert$							
	2		3		4		5	
n	KW	SR	KW	SR	KW	SR	KW	SR
5	0.143	0.621	16.391	1.870	269.686	3.784	–	7.380
6	0.116	0.483	0.492	1.824	93.133	10.128	759.206	15.043
7	0.144	0.360	0.180	3.329	0.303	4.623	19.925	13.066
8	0.236	0.264	0.232	0.721	0.296	0.887	0.332	1.173
9	0.352	0.244	0.404	0.284	0.452	0.432	0.476	0.452
10	0.616	0.252	0.745	0.344	0.765	0.492	0.729	0.460
11	1.124	0.424	1.305	0.444	1.385	0.553	1.225	0.677
12	2.338	0.733	2.038	0.681	2.378	0.797	2.507	0.881
13	4.505	1.233	3.792	1.293	4.620	1.380	4.713	1.249

7 Conclusion

Based on our results, we conclude that our method has significant potential for use in practical applications. Indeed, our reduction is so successful that it *almost* minimizes the NFA in most cases. Our method is general, in that it can easily handle $\star$-NFAs. It is not clear at this point how much work is needed to convert other reduction algorithms to operate on $\star$-NFA, apart from one investigation conducted by Müller [16].

SAT solvers are widely used and highly optimized, and the SAT problem is actively studied. Recent work on distributed and parallel SAT solvers (see the survey by Singer [18]) opens up the possibility of also improving the running time of our algorithm in this way.

Depending on the SAT solver, it may be possible to specify additional constraints on the reduced NFA. For example, we may be interested in a reduced NFA with the fewest number of transitions, or the fewest number of final states.

Future work can explore many directions, particularly when it comes to the SAT solver used; this includes issues like: (1) determine which SAT solvers work best; (2) determine whether a special SAT solver is required; and (3) investigate an incremental SAT solver that can reuse work from previous iterations.

References

1. Amilhastre, J., Janssen, P., Vilarem, M.C.: FA minimization heuristics for a class of finite languages. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 1–12. Springer, Heidelberg (1999)
2. Carrez, C.: On the minimalization of nondeterministic automata. Technical report, Laboratoire de Calcul de Faculté des Sciences de l'Université de Lille (1970)
3. Champarnaud, J.-M., Coulon, F.: Nfa reduction algorithms by means of regular inequalities. Theoretical Computer Science 327(3), 241–253 (2004)

4. Daciuk, J., Mihov, S., Watson, B.W., Watson, R.E.: Incremental construction of minimal acyclic finite-state automata. Comp. Linguistics 26(1), 3–16 (2000)
5. Ilie, L., Navarro, G., Yu, S.: On NFA reductions. In: Karhumäki, J., Maurer, H., Păun, G., Rozenberg, G. (eds.) Theory Is Forever. LNCS, vol. 3113, pp. 112–124. Springer, Heidelberg (2004)
6. Ilie, L., Yu, S.: Algorithms for computing small NFAs. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 328–340. Springer, Heidelberg (2002)
7. Indermark, K.: Zur Zustandsminimisierung nichdeterministischer erkennender Automaten. GMD Seminarberichte Bd. 33, Gesellschaft für Mathematik und Datenverarbeitung (1970)
8. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. In: Leach Albert, J., Monien, B., Rodríguez-Artalejo, M. (eds.) ICALP 1991. LNCS, vol. 510, pp. 629–640. Springer, Heidelberg (1991)
9. Kameda, T., Weiner, P.: On the state minimization of nondeterministic finite automata. IEEE Transactions on Computers C-19, 617–627 (1970)
10. Kim, J.H.: State minimization of nondeterministic machines. Technical Report RC 4896, IBM Thomas J. Watson Research Center (1974)
11. Malcher, A.: Minimizing finite automata is computationally hard. Theoretical Computer Science 327(3), 375–390 (2004)
12. Matz, O., Miller, A., Potthoff, A., Thomas, W., Valkema, E.: Report on the program AMoRE. Technical Report 9507, Christian-Albrechts-Unversität, Kiel (Oct. 1995)
13. Melnikov, B.F.: A new algorithm of the state-minimization for the nondeterministic finite automata. Korean Jnl. of Comp. and Appl. Mathematics 6(2), 277–290 (1999)
14. Mihov, S.: Direct building of minimal automaton for given list. Annuaire de l'Universite de Sofia "St. Kl. Ohridski" (1998)
15. Mohri, M.: Finite-state transducers in language and speech processing. Comp. Linguistics 23(2), 269–311 (1997)
16. Müller, G.: Minimization of symmetric difference finite automata. Master's thesis, Stellenbosch University (April 2006)
17. Polák, L.: Minimalizations of NFA using the universal automaton. International Jnl. of Foundations of Computer Science 16(5), 999–1010 (2005)
18. Singer, D.: Parallel resolution of the satisfiability problem: A survey. In: Talbi, E.-G. (ed.) Parallel Combinatorial Optimization, ch. 5, pp. 123–148. John Wiley and Sons, Chichester (2006)
19. van Zijl, L., Daciuk, J., Müller, G.: Minimization of unary symmetric difference NFAs. South African Computer Jnl. 34, 69–75 (2005)
20. `http://www.princeton.edu/~chaff/zchaff.html`