Deciding Monadic Second Order Logic over ω-Words by Specialized Finite Automata

Stephan Barth(✉)

Ludwig-Maximilians-Universität München, Munich, Germany
stephan.barth@ifi.lmu.de

Abstract. Several automata models are each capable of describing all ω-regular languages. The most well-known such models are Büchi, parity, Rabin, Streett, and Muller automata. We present deeper insights and further enhancements to a lesser-known model. This model was chosen and the enhancements developed with a specific goal: Decide monadic second order logic (MSO) over infinite words more efficiently.

MSO over various structures is of interest in different applications, mostly in formal verification. Due to its inherent high complexity, most solvers are designed to work only for subsets of MSO. The most notable full implementation of the decision procedure is MONA, which decides MSO formulae over finite words and trees.

To obtain a suitable automaton model, we further studied a representation of ω-regular languages by regular languages, which we call loop automata. We developed an efficient algorithm for homomorphisms in this representation, which is essential for deciding MSO. Aside from the algorithm for homomorphism, all algorithms for deciding MSO with loop automata are simple. Minimization of loop automata is basically the same as minimization of deterministic finite automata. Efficient minimization is an important feature for an efficient decision procedure for MSO. Together this should theoretically make loop automata a well-suited model for efficiently deciding MSO over ω-words.

Our experimental evaluation suggests that loop automata are indeed well suited for deciding MSO over ω-words efficiently.

1 Introduction

Decidability of monadic second order logic (MSO) over ω-words, (alternative names are: full MSO, S1S; MSO alone is also used sometimes for MSO over ω-words) was shown in 1962 [3], using nondeterministic Büchi automata (NBA). MSO has a central position in model checking as it subsumes some important relevant specification languages, e.g. linear temporal logic (LTL) and Presburger arithmetic. Nevertheless MSO is rather used for its rich theory than practically as efficient implementations are missing for most variants of MSO; a gap this paper helps to close.

This research was funded by the DFG (German Research Foundation), within the Research Training Group 1480: Programm- und Modell-Analyse (PUMA).

E. Ábrahám and M. Huisman (Eds.): IFM 2016, LNCS 9681, pp. 245–259, 2016.
DOI: 10.1007/978-3-319-33693-0_16

The most mature implementation of any variant of MSO is MONA [8], an implementation of the decision procedure for weak MSO over words and trees (wMSO, also WS1S and WS2S), a variant of MSO deciable by the use of finite automata. In MONA, minimization in every step is crucial for the efficiency [10], We use this insight to handle MSO over ω-words efficiently. Beside technical insights, the success of MONA also supports the relevance of decision procedures for MSO.

1.1 State of the Art in Minimizing Automata

Within reasonable time, modern computers with state of the art minimization procedures can handle automata of very different sizes. In case of deterministic finite automata (DFA), automata with millions of states can be minimized, due to its $n \log n$-TIME minimization procedure [9]. Widespread models for ω-regular languages have no well scaling complete minimization procedures. Hence, the existing minimization procedures cannot handle more than 20 states.

We studied minimization of NBA in our own work. We failed to minimize automata whose minimal automaton needs more than 10 states in reasonable time [1].

Minimization of Deterministic Büchi automata (DBA) does not work well over 20 states [5].

No minimization procedure is published for deterministic parity automata (DPA). As DPA subsume DBA, it is unlikely, that a minimization procedure based on the same principles will succed for automata with more than 20 states.

Even incomplete heuristics are used, that fail to minimize bigger automata, for example [6], which fails to compute when used for automata with more than 30 states.

These comparable small numbers are not least due to the complexity of these problems, PSPACE-complete for minimizing NBA (corollary from [11], Lemma 3.2.3), NP-complete for DBA and DPA [12].

With more computational power only DFA would allow for minimization of notably bigger automata. The other algorithms do not scale well.

Thus, mostly incomplete heuristics that miss much of opportunities for minimization are used in applications. This situation is comparable to finite automata, where minimization for nondeterministic finite automata (NFA) also lacks efficiency, which is why MONA relies on DFA. Full minimization is not necessary in the decision procedure, but some normalization of automata or other procedures for preventing an unbounded amount of extra states are. There are normalization procedures for Büchi automata [7], but they are not helpful for deciding MSO as normalization is also PSPACE-complete; the normalization in [7] is not successful for automata with more than 20 states.

1.2 Our Approach

In order to obtain a model well suited for deciding MSO, we have chosen a lesser-known automata model for ω-regular languages only by the criterion how

good it can be minimized. Then we developed the missing procedures that are necessary for deciding MSO. The main ingredients of this model are an already known method for representing ω-regular languages by regular ones [4]. Thus, only an efficient minimization procedure for regular languages is needed.

To represent ω-regular languages by regular ones, we take the representation introduced in [4]: for given ω-regular languages L, $L_\$:= \{u\$v \mid uv^\omega \in L\}$ is regular. We call $L_\$$ the loop language of L, an automaton for $L_\$$ loop automaton, and L-X the loop automaton model that results from using the automaton model X for loop languages. Algorithms for converting between NBA and loop deterministic finite automata (L-DFA) were presented in 1994 [4]. For deciding MSO, we investigate further to construct an algorithm to perform various operations directly on loop automata. Most notably, an algorithm for performing homomorphisms is presented here.

First experimental results hint that the decision procedure with loop automata is often considerably more efficient than a more classical NBA/DPA approach, which can be considered state of the art. Though on few formulae the NBA/DPA approach is a bit more efficient than L-DFA, our research hints that they are the most appropriate model for deciding MSO among the known models. This result was to be expected considering the already-mentioned research from the MONA team that minimization in every step is crucial in deciding wMSO [10].

The main advantage of L-DFA is, that the minimization procedure of DFA can be directly be applied, hence automata with millions of states can be minimized, in contrast to the under 20 states for conventional ω-automata.

1.3 Related Work

[3] showed the decidability of MSO over ω-words. NBA were introduced to do so. This procedure was very inefficient. Beside enhancements in complementation of Büchi automata, this is still to consider state of the art, nevertheless.

[13] summarizes the current state of NBA complementation. Using the best known complementation algorithm for NBA greatly enhances the efficiency of the method from [3]. This is used as comparison in the experimental evaluation in Sect. 4.

The MONA tool [8] is a successful implementation of wMSO. An analysis by the MONA team is used as hint for what might contribute to efficiency [10]. For this paper, we focus on their result that minimization after every step is the most important optimization in the decision procedure.

[4] introduces loop automata (albeit not given a name) and transformations between them and NBA. However, they authors did not provide an algorithm for performing homomorphisms, which are essential for deciding MSO.

[7] also uses loop automata, but only as intermediate device for normalizing NBA. No algorithms for working directly on loop automata are presented.

[2] study ω-languages, which only consist of ultimate periodic words. In there, the representation as loop automaton is used and some algorithms are workes out, but only for the special case of these restricted languages.

1.4 Main Contribution

The main contribution of this paper is a more efficient decision procedure for MSO over ω-words. This consists of identifying loop languages as suitable automata model and the homomorphism algorithm for loop languages in Theorem 4. The more detailed characterization of loop languages in Theorem 2, may further help to improve the handling of loop languages. Furthermore, the experimental evaluation in Sect. 4 indicates, that this method is indeed more efficient than existing ones.

2 Notion and Prerequisites

Definition 1 (Alphabet, Word). *An alphabet is a finite set. A word w is a finite or infinite series of elements of the alphabet. w_i denotes the i-th element of the series, also called the i-th letter of the word w.*

Definition 2 (Büchi and Finite Automata, Path). *The first letter in the abbreviation determines whether deterministic (D) or nondeterministic (N) automata are refered to. Büchi (DBA/NBA) and finite automata (DFA/NFA) are tuples $\mathfrak{A} = (Q, \Sigma, \delta, q_0, F)$, where*

- *$Q = \{q_0, \ldots, q_{n-1}\}$ is a finite set of states;*
- *Σ is a finite set, the alphabet;*
- *δ is a function, in case of deterministic automata, $\delta : Q \times \Sigma \to Q$, in case of nondeterministic automata $\delta : Q \times \Sigma \to \mathbb{P}(Q)$;*
- *q_0 is the initial state and the first state of the canonical enumeration;*
- *$F \subseteq Q$ is the set of final states.*

To treat deterministic automata as nondeterministic, $\delta(q, a)$ with $\{\delta(q, a)\}$.

In an automaton $\mathfrak{A}$, there is a path from $q \in Q$ to $q' \in Q$ labeled with word w, written as $q \xrightarrow{w} q'$, when $q = q' \wedge w = \varepsilon$ or w starts with the letter a the remainder of the word is v and there is a state q'' with $q'' \in \delta(q, a)$ and $q'' \xrightarrow{v} q'$.

A finite word $w \in \Sigma^*$ is considered to be accepted by a finite automaton, if there is a state $q \in F$ such that $q_0 \xrightarrow{w} q$. An infinite word $w \in \Sigma^\omega$ is considered to be accepted by a Büchi automaton, if there is a state $q \in F$ such that w can be split in subwords, each of finite length greater than zero $w = uv_0v_1v_2\ldots$, such that $q_0 \xrightarrow{u} q$ and for all $i \in \mathbb{N}_0$ it holds that $q \xrightarrow{v_i} q$.

Definition 3 (Monadic Second Order Logic). *Monadic Second Order Logic (MSO) formulae are of the form: $\varphi, \psi := x < y | x \in X | \exists x.\varphi | \exists X.\varphi | \neg\varphi | \varphi \vee \psi$*

MSO exists in several variants. The specific type of the first order (x, y) and second order variables (X) in this definition depends on the variant of MSO considered. MSO over ω-words means $x, y \in \mathbb{N}$, $X \subseteq \mathbb{N}$; this can be decided with Büchi automata [3]; any other model for ω-regular languages can be used as well, as long as algorithms for conjunction, complementation and homomorphisms are known.

Theorem 1 (MSO over ω-Words is Decidable, Büchi 1962 [3]). *For a MSO formula with no free variables it is decidable, whether it holds. For a formula with free variables it is decidable whether the formula is satisfiable and whether it is falsifiable.*

Proof (sketch).

The values of the variable of the formula ψ are stored in an infinite word. Herein the i-th letter encodes the relation between the number i and each variable. That is, for each variable X we store, whether $i \in X$. First order variables are encoded as second order variables that contain precisely one number. To encode the values of the variables use an alphabet of $2^{\text{number of variables}}$ letters, each letter for one combination of variable values.

An automaton can be constructed by structural induction over the formula. Most notable existential quantification corresponds to homomorphisms, which are examined in Theorem 4. □

3 Representation of ω-Regular Languages by Regular Languages

Definition 4 (Loop Language). *For a given ω-regular language L, $L_\$:= \{u\$v \mid uv^\omega \in L\}$.*

We call $L_\$$ the loop language of L, an automaton for $L_\$$ loop automaton, and L-X the loop automaton model that results from using the automaton model X for loop languages. By M_ω, we denote an ω-regular language for the regular language M with the property that $M_{\omega\$} = M$. Note that not for every regular language M a language M_ω exists.

Transformations between nondeterministic Büchi automata (NBA) accepting L and deterministic finite automata (DFA) accepting $L_\$$ were presented in 1994 [4].

Note that $L_\$$ and thus the minimal DFA are uniquely determined, hence the known efficient minimization procedures for DFA work for L-DFA as well.

It is thus natural to base a decision procedure for MSO on loop automata. However, in doing so, one faces the obstacle that homomorphism has to be implemented on the level of loop automata.

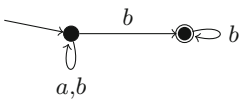
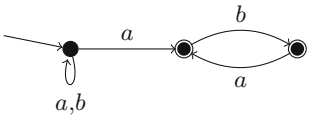
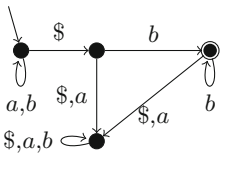
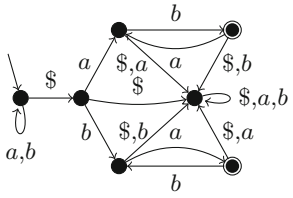
It might be this obstacle has prevented other authors from following this path. This is precisely the gap we are closing in this paper.

We present two example languages in the usual ω-style, as well as in loop style in Table 1.

3.1 Properties of Loop Languages

Definition 5 (Representative). *Given a finite alphabet Σ, a new letter $\$ \notin \Sigma$, and an ultimately periodic ω-word $w \in \Sigma^\omega$, a finite word $u\$v$, with $uv^\omega = w$, is called representative of w.*

Table 1. Two examples of languages for comparing the ω-regular language and its corresponding loop language

ω-regular expression	$(a\|b)^*b^\omega$	$(a\|b)^*(ab)^\omega$
Büchi automaton		
Loop regular expression	$(a\|b)^*\$b^+$	$(a\|b)^*\$((ab)^+\|(ba)^+)$
Loop DFA		

Definition 6 (Duplication and Rotation). *For regular languages $L \subseteq (\Sigma \cup \{\$\})^*$, the following are defined*

- *up-duplication holds for* $L :\Leftrightarrow \forall u, v \in \Sigma^*.(u\$v \in L \Rightarrow \forall i \in \mathbb{N}_1.u\$v^i \in L)$
- *down-duplication holds for* $L :\Leftrightarrow \forall u, v \in \Sigma^*.(u\$v \in L \Leftarrow \exists i \in \mathbb{N}_1.u\$v^i \in L)$
- *up-rotation holds for* $L :\Leftrightarrow \forall u, v \in \Sigma^*, a \in \Sigma.(u\$av \in L \Rightarrow ua\$va \in L)$
- *down-rotation holds for* $L :\Leftrightarrow \forall u, v \in \Sigma^*, a \in \Sigma.(u\$av \in L \Leftarrow ua\$va \in L)$

Theorem 2 (Characterization of Loop Languages). *A language is loop if and only if the following holds*

- *regular: L is regular*
- *wellformed: $L \subseteq \Sigma^*\$\Sigma^+$*
- *representative independence: for all words u, v, x, and y it holds that if $uv^\omega = xy^\omega$, then $u\$v \in L \iff x\$y \in L$*

  Representative independence is further equivalent to the conjunction of up-duplication, down-duplication, up-rotation, down-rotation.

Proof. Regularity of loop languages has already been proven [4].

Wellformedness has to hold as only wellformed words represent ω-words. Representative independence has to hold because $L_\$$ represents L and membership cannot depend on the chosen representative. Furthermore, given representative independence all four (up/down)-(rotation/duplication) properties have to hold, as $\forall i \in \mathbb{N}_1, a \in \Sigma.(uv^\omega = u(v^i)^\omega \wedge u(av)^\omega = ua(va)^\omega)$.

On the other hand, given these properties and any word in L, the representative of minimal length for the same ω-word has to be in the language as well (by down-) and every representative of this word has to be in the language (by up-).

These properties are sufficient for loopness, as there exists an algorithm to construct an ω-regular language L_ω out of every language $L_\$$ with the stated properties. That transformation algorithm was given in [4]. □

A minor remark here is that, except for the empty language, every loop automaton has at least 4 states, resulting from wellformedness:

- q_0: The starting state; must not be final;
- $q_1 = \delta(q_0, \$)$: Different from q_0, as no \$ may ever occur in a word afterwards; may not be final;
- $\delta(q_1, \$)$: Rejection sink; different from all states before; may not be final;
- at least one final state.

3.2 Homomorphism Closure of Loop Automata

Definition 7 (Homomorphism). *Let L be a formal language over the finite alphabet Σ, Γ another finite alphabet and $f : \Sigma \to \Gamma$.*

$f(L) := \{w \mid \exists v \in L.(|v| = |w| \wedge \forall i \in \mathbb{N}_0.w_i = f(v_i))\}$ is called the homomorphism defined by f.

Applying homomorphisms on NBA is simple: replace all letters a by $f(a)$.

The problem with loop automata is that given an homomorphism f it is possible that $f(L_\$) \neq (f(L))_\$$. Hence applying the homomorphisms directly on the regular language does not yield the correct result.

Example 1. Given $L = a(ab)^\omega$, $f = \{a \mapsto a, b \mapsto a\}$ it holds that $L_\$ = a(ab)^*\$(ab)^+|aa(ba)^*\$(ba)^+$, $f(L_\$) = a(aa)^*\$(aa)^+|aa(aa)^*\$(aa)^+ = a^+\$(aa)^+$, $f(L) = a^\omega$, $(f(L))_\$ = a^*\$a^+$.

This results in $f(L_\$) \neq (f(L))_\$$.

Therefore, up till now the only known way to do so far was to convert the L-DFA to an NBA (resulting in $O(n^5)$ states for an n state L-DFA), to perform the homomorphism n the NBA and to convert it back to an L-DFA (resulting in $2^{O(n^2)}$ states for an n state NBA). This leads to a total state blowup of $2^{O(n^{10})}$ states. Our new construction does not need Büchi automata and results in a smaller state blowup of $2^{O(n^2)}$ states.

It is not surprising that this operation is costly, as homomorphism on DFA already results in up to 2^n states for an n state automaton.

Lemma 1. *Given a homomorphism f, it holds, that*
$(f(L))_\$ = \{u\$v \mid \exists i, j \in \mathbb{N}_1.uv^i\$v^j \in f(L_\$)\}$

Proof. Note that \$ $\mapsto$ \$ and no other letter maps to \$, as \$ is not part of the ω-language.

$f(L_\$)$

- is regular: $L_\$$ is regular and homomorphisms map regular languages to regular ones;
- is wellformed, as letters are mapped to letters by f and \$ is kept unmodified;
- is not representative independent, but admits up-(duplication/rotation), as

- up-duplication: $u\$v \in f(L_\$) \Rightarrow \exists u'\$v' \in L_\$.(u\$v = f(u'\$v'))$
  $\overset{\text{up-duplication of } L_\$}{\Longrightarrow} \exists u'\$v' \in L_\$.(u\$v = f(u'\$v') \wedge \forall i \in \mathbb{N}_1.u'\$v'^i \in L_\$) \Rightarrow \forall i \in \mathbb{N}_1.u\$v^i \in f(L_\$)$
- up-rotation: $u\$av \in f(L_\$) \Rightarrow \exists u'\$a'v' \in L_\$.(u\$av = f(u'\$a'v'))$
  $\overset{\text{up-rotation of } L_\$}{\Longrightarrow} \exists u'\$a'v' \in L_\$.(u\$av = f(u'\$a'v') \wedge u'a'\$v'a' \in L_\$) \Rightarrow ua\$va \in f(L_\$)$

– contains at least one representative for every ultimate periodic word in $f(L)$: Given $uv^\omega \in f(L)$, there is some $u'v'^\omega \in L$ with $f(u'v'^\omega) = uv^\omega$. Note that neither necessarily $f(u') = u$, nor $f(v') = v$. With $u'v'^\omega \in L$ it also holds, that $u'\$v' \in L_\$$, as well as $f(u'\$v') \in f(L_\$)$. Consider that $f(u')(f(v'))^\omega = uv^\omega$. $f(u'\$v') \in f(L_\$)$ is hence a representative of uv^ω;
– contains no representatives for words not in $f(L)$, as for every $u\$v \in f(L_\$)$, there exists $u'\$v' \in L_\$$ with $f(u'\$v') = u\$v$. As $u'\$v' \in L_\$$, $u'v'^\omega \in L$ and $f(u'v'^\omega) \in f(L)$. Hence, $u\$v$ is a representative for a word in $f(L)$.

Hence, if all words, which are down-(rotated/duplicated) variants of words in $f(L_\$)$ are added to the language, we obtain $(f(L))_\$$. It is sufficient to ensure down-rotation of the form $uv\$v \in L \Rightarrow u\$v \in L$, as up-rotation holds. Therefore, $(f(L))_\$ = \{u\$v \mid \exists i, j \in \mathbb{N}_1.uv^i\$v^j \in f(L_\$)\}$.

□

Theorem 3. *Given an NFA $\mathfrak{A}$ with n states, $\{u\$v \mid \exists i, j \in \mathbb{N}_1.uv^i\$v^j \in L(\mathfrak{A})\}$ can be accepted by a DFA with $2^n \cdot (2^{n^2}+1) = 2^{n^2+n} + 2^n = 2^{O(n^2)}$ states.*

Proof. Given an NFA $\mathfrak{A} = (Q = \{q_0, \ldots, q_{n-1}\}, \Sigma, \Delta, q_0, F)$, construct a DFA $\mathfrak{B} = (Q', \Sigma, \delta, q_0', F')$ with
let $\vartheta(M, a) = \{q \mid \exists p \in M.(p, a, q) \in \Delta\}$

– $Q' = \mathbb{P}(Q) \times (\mathbb{P}(Q)^n \cup \{()\})$
– $\delta((M, ()), a) = \begin{cases} (\vartheta(M, a), ()) & \text{if } a \neq \$ \\ (M, (\{q_0\}, \ldots, \{q_{n-1}\})) & \text{if } a = \$ \end{cases}$
  $\delta((M, (M_0, \ldots, M_{n-1})), a) = \begin{cases} (M, (\vartheta(M_0, a), \ldots, \vartheta(M_{n-1}, a))) & \text{if } a \neq \$ \\ (\{\}, ()) & \text{if } a = \$ \end{cases}$
– $q_0' = (\{q_0\}, ())$
– $F' = \{(M, (M_0, \ldots, M_{n-1})) \mid$ Let (P, O) be the least fixpoint of the function $f(P, O) = (P \cup M \cup \{q \mid \exists i.q \in M_i \wedge q_i \in P\}, O \cup \vartheta(P, \$) \cup \{q \mid \exists i.q \in M_i \wedge q_i \in O\}), F \cap O \neq \varnothing\}$

Let $w = uv^i\$v^j \in L(\mathfrak{A})$. There are $p_0, \ldots, p_i, o_0, \ldots, o_j \in Q$ such that $q_0 \xrightarrow{u} p_0, \forall 0 \leq k < i.p_k \xrightarrow{v} p_{k+1}, p_i \xrightarrow{\$} o_0, \forall 0 \leq k < j.o_k \xrightarrow{v} o_{k+1}$ and $o_j \in F$, as w is accepted by $\mathfrak{A}$. The run of $\mathfrak{B}$ on $u\$v$ includes one \$-transition, hence the state reached is of the form $(M, (M_0, \ldots, M_{n-1}))$. For every k, M_k contains precisely the states that can be reached from q_k with word v. Let (P, O) be the least fixpoints as in the definition of F'. $p_0 \in P$, as $p_0 \in M$. If $p_k \in P$ then $p_{k+1} \in P$, hence $p_i \in P$. This then leads to $o_0 \in O$. If $o_k \in O$ then $o_{k+1} \in O$, hence $o_j \in O$. As $o_j \in F$, $F \cap O \neq \varnothing$.

Conversely, if $u\$v$ if accepted by $\mathfrak{B}$, let $(M, (M_0, \ldots, M_{n-1})) \in F'$ be the state reached at the end of the run of $\mathfrak{B}$. Let (P, O) be the least fixpoints as in the definition of F'. There is a final state $r \in O$. There is an $l \in \mathbb{N}$ and a $r' \in \vartheta(P, \$)$ such that $r' \xrightarrow{v^l} r$. Furthermore, there is $m \in \mathbb{N}$ and $r'' \in M$ such that $r'' \xrightarrow{v^m\$} r'$. As $q_0 \xrightarrow{u} r''$, $uv^m\$v^l$ is accepted by $\mathfrak{A}$.

□

Theorem 4. *The homomorphism of an L-DFA $\mathfrak{A}$ with language $L_\$$ can be constructed with at most $2^{O(n^2)}$ states.*

Proof. For doing so, perform the following steps

1. compute a nondeterministic finite automaton (NFA) $\mathfrak{B}$ for $f(L_\$)$ (keeps the number of states)
2. transform the NFA into a DFA for the modified language as in Theorem 3. That is the required L-DFA (this transformation needs $2^{O(n^2)}$ states)

By Lemma 1, the language of this DFA is $(f(L))_\$$. All together, homomorphisms on loop automata can be computed in at most $2^{O(n^2)}$ states.

□

Remark 1 (Proposed Decision Procedure for MSO over ω-Words with L-DFA). Concluding we propose as decision procedure to encode the formulae quite like it was done Büchi.

For $x < y$ and $x \in X$ two concrete automata have to be chosen.

Complementation, intersection, union, and minimization of L-DFA is rather trivial given the algorithms for DFA: It is the same, but for the complement, there the result has to be intersected with $\Sigma^*\$\Sigma^+$. This is used to handle $\neg\varphi$ and $\varphi \vee \psi$.

The homomorphism from Theorem 4 is used to compute $\exists x.\varphi$ and $\exists X.\varphi$.

With that, loop automata can be used for deciding MSO.

4 Experimental Evaluation

The base of this experimental evaluation is a set of hand crafted formulae, given in Appendix A, and some random formulae. These are recursively computed by $\text{form}(1, \{x, y\}, \{X\})$, where form is defined by the following expressions:

$$\text{form}(n, f, s) := \text{ randomly select line with probability proportional to its weight}$$

formula	weight
$x < y$	1
$x \in X$	1
$z :=$ freshname; $\exists z.\text{form}(n+1, f \cup \{z\}, s)$	1
$Z :=$ freshname; $\exists Z.\text{form}(n+1, f, s \cup \{Z\})$	1
$\neg\text{form}(n+1, f, s)$	1
$\text{form}(n+1, f, s) \vee \text{form}(n+1, f, s)$	$\frac{5}{n}$
$\text{form}(n+1, f, s) \wedge \text{form}(n+1, f, s)$	$\frac{5}{n}$

The line used for the construction of the formula is randomly selected taking the weight into account. For a call with the weights $p_i, i = 0, \ldots, 6$, each line k is chosen with probability $\frac{p_k}{\sum_{i=0}^{6} p_i}$. Variable names are randomly chosen with equal probability out of the set f for first order and s for second order variables; for $x < y$, x and y are different.

This formula generation was chosen such that it can generate every MSO-formula, the generation terminates with probability 1, and generates reasonable large formulae for experiments.

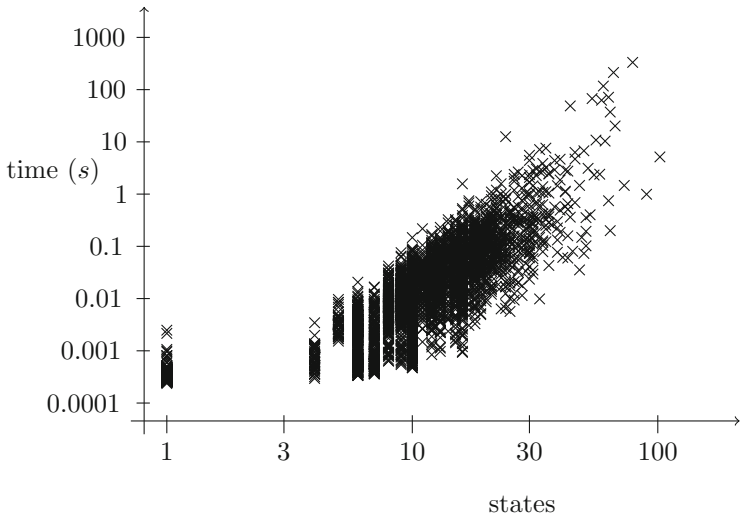


Fig. 1. Runtime and maximal state count of the decision procedures for random formulae

A scatterplot with maximal state count in the decision procedure and runtime (Intel i5-2540M, 2.60 GHz) of our implementation with 5000 of these formulae is given in Fig. 1. No formula lead to timeout or out of memory.

For comparing with state of the art, we use a more classical procedure using NBA and DPA. Just the maximal state count in the run of the decision procedure is compared, as it is more significant than the runtime, as the runtime depends more on the quality of implementation than the count of states and both implementations are not optimized in regards to runtime. While in practice the runtime is more important then the statecount, this comparison aims for comparing how well suited the different automata models are for deciding MSO and not on how well speed optimized the current implementations are; in fact both are more optimized for simplicity and debugging the automata themselves than for runtime. Especially the time efficiency of our NBA/DPA implementation is suboptimal so using our implementation would not result in a meaningful comparison.

A further advantage is that comparison can be done with a particularly strong minimization heuristic in the NBA/DPA approach, which increases the confidence in L-DFA to be better suited for deciding MSO as further advantages might be able to reduce the state count for the NBA/DPA approach, while L-DFA are already at their global minimum.

Complementation of NBA is done via transformation to DPA, as advised by [13]. There are some widely used minimization heuristics for NBA and DPA. The two heuristic in this experiment are: (1) In the DPA, for every pair of states is checked whether merging these keeps the language the same. This heuristic subsumes quite some widely used heuristics, but it is not frequently used as such, as it is too slow for most applications.

(2) Additionally, on the NBA it is checked whether the automaton gets smaller after complemented two times. Given the high complexity of the complementation procedure, this sounds quite time intensive. Nevertheless, it even speeds up the decision process in many cases.

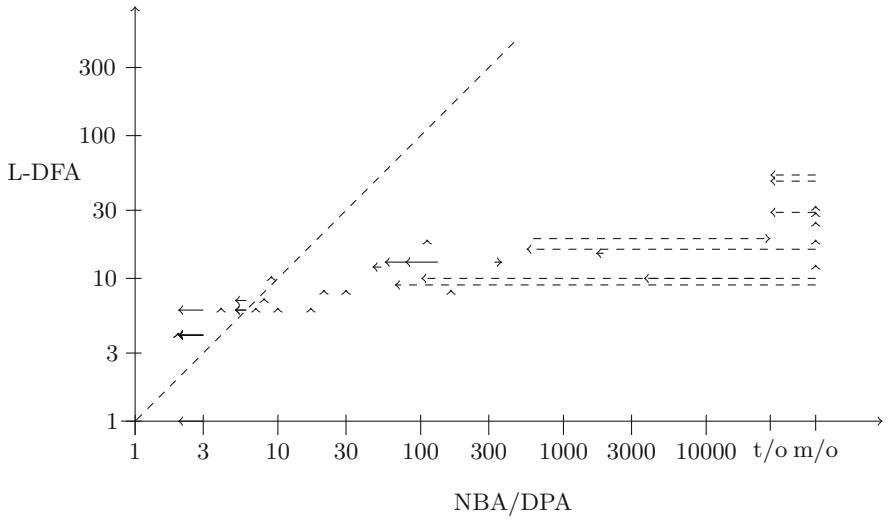


Fig. 2. Comparison of maximal state count in the run of the decision procedures for random formulae; arrowheads compare L-DFA against NBA/DPA with strong minimization heuristic; arrowtails compare L-DFA against NBA/DPA with weak minimization heuristic

On the other hand, there are formulae, for which the advantage from the heuristical minimization (1) is not that big. In fact, there are even very few counterintuitive examples, for which the state count is smaller when the state merging heuristic is not used; Appendix A, second table entry gives an example of such a formula.

The L-DFA approach indeed turns out to be a lot more efficient than the NBA/DPA-approach for many formulae.

Figure 2 contains the maximal automata sizes in the course of the decision procedure of 40 random formulae. Each formula is given as arrow. The size of the L-DFA is given in y-direction. The size of the NBA/DPA with only minimization heuristic (1) is given as arrow tail, the size with both heuristics is given as arrowhead. If the sizes coincide, this is denoted as arrow of length zero, directing upwards. Dashed arrows denote that at least one end is timeout (t/o) or memoryout (m/o). All these sizes are the maximal state count in the run. L-DFA have a strong advantage here. It becomes even bigger, the larger the resulting automata are.

5 Conclusion

We now have all necessary algorithms to decide MSO with loop automata. Hence, we now have an automaton model for ω-regular languages that is suitable for deciding MSO and allows for efficient minimization at the same time. Along with its applicability for MSO, these deeper insights might offer further theoretical and practical enhancements in the field of ω-languages.

On the experimental side, the first benchmarks hint that loop automata are indeed superior to classical automata for ω-regular languages in efficiently deciding MSO. The automata computed out of formulae are often smaller than the corresponding NBA and DPA. Additionally, the word test for L-DFA is simpler than for conventional ω-automata, hence even for automata of the same size L-DFA are preferable. Furthermore, the implementation with L-DFA for automata of the same size is more performant than the implementation with NBA/DPA, which is mostly because of the more complex minimization heuristic, but also because the transformation from NBA to DPA needs to compute more per state than the homomorphism on L-DFA.

Furthermore, we collected further evidence that frequent minimization or at least a minimization heuristic is indeed helpful for deciding MSO which supports the observations in [10].

A full implementation of MSO over ω-words utilizing L-DFA with a stronger focus on runtime and integration of other optimizations is under development.

A Automata Sizes for Various Formulae

In Table 2 formulae are enlisted together with the efficiency with the classical NBA/DPA approach as well as with L-DFA.

The formulae are on the one hand formulae that appear often in model checking, such as fairness. The formulae are collected in a random like manner, with the goal to cover a broad part of the behaviour of the MSO solvers.

Efficiency is measured mainly in state count of the automata here. The resulting state count and the maximum state count for any subformula in the course of the decision procedure are recorded. The two lines in each cell denote these.

Timeout (t/o) after more than at least an hour, and memory out (m/o) when using more than 1 GB of RAM are stated in these cases. In some cases fine tuning in the solving procedure allowed for a partial handguided solving. Some

Table 2. State count of automata from MSO formulae for biggest intermediate result and end result

Formula	NBA/DPA	NBA/DPA	L-DFA
	Strong min	Weak min	
$(x \in X \wedge \neg x \in Y) \vee (x \in Y \wedge \neg x \in X)$	14/11	17/16	9
	14/11	17/16	9
$\neg\exists x.((x \in X \wedge \neg x \in Y) \vee$	33/4	30/79	9
$(x \in Y \wedge \neg x \in X))$	5/4	5/79	4
$\text{after}(X, Y) := \forall x.(x \in X \Rightarrow$	18/5	33/11	9
$\exists y.(y > x \wedge y \in Y))$	6/3	27/11	7
$\text{fair}(X, Y) := \text{after}(X, Y) \wedge \text{after}(Y, X)$	42/27	288/313	9
	42/27	288/313	9
$\forall X.(\text{fair}(X, Y) \Rightarrow \text{fair}(Y, Z))$	(14377)/	m/o	14
	(6131)[a]		12
$\text{suc}(x, y) := x < y \wedge$	20/32	26/41	10
$\forall z.(\neg x < z \vee \neg z < y)$	6/5	14/17	6
$\text{suc2}(x, y) := \exists z.(\text{suc}(x, z) \wedge \text{suc}(z, y))$	780/32	783/41	10
	8/6	19/15	7
$\text{suc4}(x, y) := \exists z.(\text{suc2}(x, z) \wedge \text{suc2}(z, y))$	780/32	783/41	10
	12/8	29/20	9
$\text{suc8}(x, y) := \exists z.(\text{suc4}(x, z) \wedge \text{suc4}(z, y))$	780/32	783/41	13
	20/12	43/27	13
$\text{inf}(X) := \forall u \exists v.(u < v \wedge v \in X)$	8/5	17/11	9
	5/3	13/6	4
$\text{inf}(X) \vee \text{inf}(Y)$	9/5	25/290	9
	7/5	25/290	4
$(\text{inf}(U) \vee \text{inf}(V)) \Rightarrow (\text{inf}(X) \vee \text{inf}(Y))$	15/6	m/o	9
	14/6		6
$\exists U.((\text{inf}(U) \vee \text{inf}(V)) \Rightarrow$	15/8	m/o	9
$(\text{inf}(X) \vee \text{inf}(Y)))$	15/8		6
$\text{infsuc}(X, Y) := \forall u \exists x, y.(u < x \wedge$	394/31	397/41	18
$\text{suc}(x, y) \wedge x \in X \wedge y \in Y)$	6/3	19/10	8
$\exists Y.(\text{infsuc}(X, Y) \wedge \text{infsuc}(Z, Y))$	394/508	m/o	20
	73/508		6
$\text{zeroin}(X) := \exists u.(u \in X \wedge \neg\exists v.(v < u))$	19/5	27/9	6
	9/8	9/9	6
$\text{alter}(X) := \text{zeroin}(X) \wedge \forall x, y.$	53/90	61/719	12
$(\text{suc}(x, y) \Rightarrow x \in X \iff \neg y \in X)$	4/3	11/11	9
$\text{offset}(X, Y) := \forall i \forall j.(suc(i, j) \wedge$	28/31	31/169	11
$i \in X \Rightarrow j \in Y)$	4/3	9/163	9

Table 2. (*Continued*)

Formula	NBA/DPA Strong min	NBA/DPA Weak min	L-DFA
$\text{offset}(X,Y) \wedge \text{offset}(Y,Z) \wedge \text{offset}(Z,X)$	49/40	81/163	107
	49/40	81/163	107
$\text{offset}(V,W) \wedge \text{offset}(W,X) \wedge$	97/(444)[b]	161/444	2331
$\text{offset}(X,Y) \wedge \text{offset}(Y,Z) \wedge \text{offset}(Z,V)$		161/444	2331
$\exists Y.(\text{offset}(X,Y) \wedge \text{offset}(Y,Z))$	28/31	41/163	29
	21/14	29/26	29
$\text{insm}(i,j,U,V,W) := (j \in U \Rightarrow$	7/13	8/23	15
$i \in V \vee i \in W)$	7/13	8/23	15
$\forall i \forall j.(\text{suc}(i,j) \Rightarrow \text{insm}(i,j,U,V,Z) \wedge$	t/o[c]	m/o	198
$\text{insm}(i,j,V,X,V) \wedge \text{insm}(i,j,X,Y,V) \wedge$			16
$\text{insm}(i,j,Y,Z,X) \wedge \text{insm}(i,j,Z,U,Y)$			
$\forall x \exists y.(x < y \wedge y \in X \wedge y \in Y)$	12/5	21/13	9
	5/3	16/7	4
$\forall x \exists y.(x < y \wedge y \in X \wedge y \in Y) \wedge$	40/6	165/118	9
$\forall x \exists y.(x < y \wedge y \in X \wedge y \notin Y)$	6/3	153/118	6
$\forall x \exists y.(x < y \wedge y \in X \wedge y \in Y) \wedge$	641/44	m/o	18
$\forall x \exists y.(x < y \wedge y \in X \wedge y \notin Y) \wedge$	85/44		18
$\forall x \exists y.(x < y \wedge y \notin X \wedge y \in Y) \wedge$			
$\forall x \exists y.(x < y \wedge y \notin X \wedge y \notin Y)$			

[a]Timeout in minimization of DPA. With weaker minimization, it ended up in this result.
[b]Minimization of parity automaton did not finish in a day.
[c]Computation did not finish in over a day. When it was stopped, it was in the process of computing a DPA out of an NBA and had already over 190000 states.

formulae needed too long the minimizing heuristics but got far more states with general weaker minimization. In that case, the states are given in braces.

References

1. Barth, S., Hofmann, M.: Learn with SAT to minimize Büchi automata. In: Faella, M., Murano, A. (eds.) GandALF. EPTCS, vol. 96, pp. 71–84 (2012)
2. Bresolin, D., Montanari, A., Puppis, G.: A theory of ultimately periodic languages and automata with an application to time granularity. Acta Informatica **46**(5), 331–360 (2009)
3. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Nagel, E., Suppes, P., Tarski, A. (eds.) Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science (LMPS 1960), pp. 1–11. Stanford University Press, June 1962
4. Calbrix, H., Nivat, M., Podelski, A.: Ultimately periodic words of rational omega-languages. In: Brookes, S., Main, M., Melton, A., Mislove, M., Schmidt, D. (eds.) MFPS 1993. LNCS, vol. 802, pp. 554–566. Springer, Heidelberg (1994). doi:10.1007/3-540-58027-1_27
5. Ehlers, R.: Minimising deterministic Büchi automata precisely using SAT solving. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 326–332. Springer, Heidelberg (2010)
6. Ehlers, R., Finkbeiner, B.: On the virtue of patience: minimizing Büchi automata. In: Pol, J., Weber, M. (eds.) SPIN 2010. LNCS, vol. 6349, pp. 129–145. Springer, Heidelberg (2010)
7. Farzan, A., Chen, Y.-F., Clarke, E.M., Tsay, Y.-K., Wang, B.-Y.: Extending automated compositional verification to the full class of omega-regular languages. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 2–17. Springer, Heidelberg (2008)
8. Henriksen, J.G., Jensen, J., Jørgensen, M., Klarlund, N., Paige, B., Rauhe, T., Sandholm, A.: Mona: monadic second-order logic in practice. In: Brinksma, E., Cleaveland, W.R., Larsen, K.G., Margaria, T., Steffen, B. (eds.) TACAS 1995. LNCS, vol. 1019, pp. 89–110. Springer, Heidelberg (1995)
9. Hopcroft, J.E.: An $n \log n$ algorithm for minimizing states in a finite automaton. Technical report CS-TR-71-190, Stanford University (1971)
10. Klarlund, N., Møller, A.: MONA implementation secrets. In: Yu, S., Păun, A. (eds.) CIAA 2000. LNCS, vol. 2088, pp. 571–586. Springer, Heidelberg (2001)
11. Kozen, D.: Lower bounds for natural proof systems. In: FOCS, pp. 254–266. IEEE Computer Society (1977)
12. Schewe, S.: Minimisation of deterministic parity and buchi automata and relative minimisation of deterministic finite automata. CoRR, abs/1007.1333 (2010)
13. Tsai, M.-H., Fogarty, S., Vardi, M.Y., Tsay, Y.-K.: State of Büchi complementation. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 261–271. Springer, Heidelberg (2011)