Lazy Automata Techniques for WS1S

FIT BUT Technical Report Series

Tomáš Fiedor, Lukáš Holík, Petr Janků, Ondřej Lengál, Tomáš Vojnar

Technical Report No. FIT-TR-2016-01
Faculty of Information Technology, Brno University of Technology

Last modified: April 29, 2016

Lazy Automata Techniques for WS1S

Tomáš Fiedor[1], Lukáš Holík[1], Petr Janků[1], Ondřej Lengál[1,2], and Tomáš Vojnar[1]

[1] FIT, Brno University of Technology, IT4Innovations Centre of Excellence, Czech Republic
[2] Institute of Information Science, Academia Sinica, Taiwan

Abstract. We present a new decision procedure for the logic WS1S. It originates from the classical approach, which first builds an automaton accepting all models of a formula and then tests whether its language is empty. The main novelty is to test the emptiness on the fly, while constructing the automaton, and prune the constructed state space from parts irrelevant for the test. The pruning is done by a generalization of two techniques used in antichain-based language inclusion and universality checking of finite automata: subsumption and early termination. The richer structure of the WS1S decision problem allows us, however, to elaborate on these techniques in novel ways. Our experiments show that the proposed approach can indeed significantly outperform the classical decision procedure (implemented in the MONA tool) as well as its recently proposed alternative based on using nondeterministic automata.

1 Introduction

Weak monadic second-order logic of one successor (WS1S) is a powerful logic for reasoning about regular properties of finite words. It has found numerous uses, from software and hardware verification through controller synthesis to computational linguistics, and further on. Some more recent applications of WS1S include verification of pointer programs and deciding related logics [1,2,3,4,5] as well as synthesis from regular specifications [6]. Most of the successful applications were due to the tool MONA [7], which implements automata-based decision procedures for WS1S and WS2S (a generalization of WS1S to finite binary trees). The worst case complexity of WS1S is nonelementary [8] and, despite many optimizations implemented in MONA and other tools, the complexity sometimes strikes back. Authors of methods translating their problems to WS1S/WS2S are then forced to either find workarounds to circumvent the complexity blowup, such as in [2], or, often restricting the input of their approach, give up translating to WS1S/WS2S altogether [9].

The classical WS1S decision procedure, used in MONA, builds an automaton A_φ accepting all models of the given formula φ, in a form of finite words, and then tests A_φ for language emptiness. The bottleneck of the procedure is the size of A_φ, which can be huge due to the fact that the derivation of A_φ involves many nested automata product constructions and complementation steps, preceded by determinization. The main point of this paper is to avoid the state-space explosion involved in this construction by testing the emptiness *on the fly*, while constructing A_φ, and by omitting the state space irrelevant for the emptiness test. This is done using two main principles: *lazy evaluation* and *subsumption-based pruning*. These principles have already appeared in the so-called antichain-based testing of language universality and inclusion of finite automata [10]. The richer structure of the WS1S decision problem allows us, however, to elaborate on these principles in novel ways and utilise their power even more.

Basic idea. We start by rewriting the input formula φ into a *language term* t_φ describing the language L_φ of all models of φ. As in MONA, the models are encoded by finite words over a multi-track alphabet where each track corresponds to a variable of φ (with the symbol $\bar{0}$ specifying that there is zero in each track). In t_φ, the atomic formulae of φ are replaced by predefined automata accepting languages of their models. Boolean operators ($\wedge$, $\vee$, and $\neg$) are turned into the corresponding set operators ($\cup$, $\cap$, and complement) over the languages of models. An existential quantification $\exists X$ becomes a sequence of two operations. First, a projection π_X removes information about valuations of the quantified variable X from symbols of the alphabet. After the projection, the resulting language may, however, encode some but not necessarily *all* encodings of the models. In particular, encodings with some specific numbers of trailing $\bar{0}$'s, which are used as a padding, may be missing. To obtain a language containing *all* encodings of the models, the language must be extended to include encodings with any number of trailing $\bar{0}$'s. This corresponds to taking the (right) $\bar{0}^*$-quotient of the language.[3]

The standard procedure, used also by MONA, constructs the automaton $\mathcal{A}_\varphi$ accepting L_φ, inductively to the structure of the term t_φ, starting from the leaves and combining the automata of sub-terms by standard automata constructions, implementing the term operators. The nonelementary complexity of this procedure is caused by the exponential worst-case complexity of complementing a nondeterministic automaton. Since complementation is hidden in every alternation of quantifiers, the worst-case complexity of the procedure is the tower of exponentials of the height equal to the alternation depth of the formula.

The main novelty of our work is that we test emptiness of L_φ *on the fly* while constructing a symbolic representation of automata corresponding to sub-formulae of φ and to φ itself. We do not need to first build automata for the sub-formulae, then for the formula itself, and only then to test emptiness—we do all these steps simultaneously.

The states of the automata are represented symbolically by language terms and accept right quotients of the languages of models of the corresponding formulae. Intuitively, for a (sub-)formula ψ, the term t_ψ represents the final state of A_ψ, encoding the entire language L_ψ (which can be seen as the trivial quotient $L_\psi - \epsilon$). Other states of A_ψ are then constructed backwards as predecessors of the final state in the transition relation of A_ψ. They are symbolically represented by terms encoding quotients $L_\psi - w$ for $|w| > 0$. Such terms are obtained when solving the above mentioned problem of trailing zeros by computing the $\bar{0}^*$-quotients. The computation is done iteratively by quotienting wrt higher and higher powers of $\bar{0}$ (the symbols $\bar{0}$ then become sets of non-zero symbols due to projection).

Optimizations. Our procedure can often avoid building large parts of the automata when they are not necessary for answering the emptiness query. For instance, when testing emptiness of the language of a term $t_1 \cup t_2$, we adopt the *lazy approach* (in this particular case the so-called *short-circuit evaluation*) and first test emptiness of the language of t_1; if it is non-empty, we do not need to process t_2. Terms built using intersection and complement can be handled similarly. Dealing with quantifiers is more problematic. The main difficulties stem from constructing the $-\bar{0}^*$ quotients. To handle

[3] The (right) L'-quotient of a language L, written $L - L'$, is the set of all prefixes of words of L with the remaining suffix in L'.

them, we proceed by considering quotients $t - \bar{0}^i$ for increasing powers of $\bar{0}$ until either we reach a fixpoint, meaning that the union of the so-far constructed quotients *subsumes* quotients for higher powers, or we can deduce that the language of $t - \bar{0}^*$ is non-empty based on some of the computed quotients $t - \bar{0}^i$. For testing the subsumption of terms (implying inclusion of their languages), we propose an efficient, safely under-approximating procedure.

Moreover, our approach can benefit from multiple further possible optimizations. For example, it can be *combined* with the *classical WS1S decision procedure*, which can be used to transform selected non-atomic sub-terms of t_φ to automata. The combination can be advantageous when no state explosion happens when determinizing the automata obtained within the classical decision procedure. The size of such automata can then be rather small due to minimization, which cannot be applied in the on-the-fly approach. Thus, we can get the best out of both worlds. Next, we can use functional-programming-style *continuations* to represent a partially evaluated term, so that we can avoid expensive recomputing of the term in case it is needed later in a different context. Finally, our method can exploit various methods of *logic-based pre-processing*, such as *anti-prenexing*, which, in our experience, can often significantly reduce the search space to be explored within fixpoint computations.

Experiments. We have implemented our decision procedure in a prototype tool called GASTON and compared its performance with two WS1S solvers, MONA and DWINA, on benchmarks from various sources. Our implementation of the algorithm performs in the majority of cases better than both MONA and DWINA. It is often better by orders of magnitude, and works well on number of formulae where MONA runs out of time or memory. We believe that efficiency of our approach can be pushed much further, making WS1S scale enough for a new classes of applications.

Related work. As already mentioned above, MONA [7] is the usual tool of choice for deciding WS1S formulae. The efficiency of MONA stems from many optimizations, both higher-level (such as automata minimization, the encoding of first-order variables used in models, or the use of BDDs to encode the transition relation of the automaton) as well as lower-level (e.g. optimizations of hash tables etc.) [11,12]. Apart from MONA, there are other related tools based on the classical automata procedure, such as JMOSEL [13] for a related logic M2L(Str), which implements several optimizations (such as second-order value numbering [14]) that allow it to outperform MONA on some benchmarks (MONA also provides an M2L(Str) interface on top of the WS1S decision procedure).

Recent works on efficient methods of testing language inclusion and universality of automata, based on the so-called antichains [15,10,16], suggest a way of alleviating the problem of automata state space explosion within the classical WS1S decision procedure. This direction has been studied for the first time in the work of Fiedor *et al.* [17] The work presents an approach addressing the issue of repeated automata determinization, which, as noted previously, is performed at every quantifier alternation when complementing a nondeterministic automaton. The discussed method does not construct the determinized automaton explicitly, but uses only its final states, which are represented symbolically using antichains. Because the determinization needs to be performed for every quantifier alternation, the constructed antichains are nested, with one nesting level for each alternation. Although the authors avoid the explicit determiniza-

tion of automata, their approach suffers from several other issues. First, it is restricted to formulae translated to the prenex normal form. For larger formulae, especially those with many conjunctions, this can mean that constructing the automaton for the matrix of the formula may be too difficult, because minimization usually does not help too much (the minimization in MONA exploits the fact that projection, done during the construction, decreases the size of the alphabet). Moreover, because the construction is bottom-up, the approach first constructs a representation of the set of final states of the automaton, and only then tests whether the set is empty or not. This may perform too much redundant work in the case there is an easy proof of (non-)emptiness. Our work builds on their approach, adapting it in several ways, most importantly by *generalizing* the structure of terms we are reasoning on and using a *lazy* algorithm, which allows early termination of nested fixpoints when testing language (non-)emptiness.

Recently, a couple of logic-based approaches for deciding WS1S appeared. Ganzow and Kaiser [18] developed a new decision procedure for the weak monadic second-order logic on inductive structures, which is even more general than WSkS. Their approach completely avoids automata; instead, it is based on Shelah's composition method. Treytel [19], on the other hand, uses the classical decision procedure, recast in the framework of coalgebras. He focuses on testing equivalence of a pair of formulae, which is performed by finding a bisimulation between derivatives of the formulae. Although the two approaches outperform MONA on some simple artificial examples, they are currently too far from being practically usable.

2 Preliminaries on Languages and Automata

A *word* over a finite alphabet Σ is a finite sequence $w = a_1 \cdots a_n$, for $n \geq 0$, of symbols from Σ. We use $w[i]$ to denote a_i. For $n = 0$, the word is the empty word ϵ. A language L is a set of words over Σ. We use the standard language operators of concatenation $L.L'$ and iteration L^*. The (right) quotient of a language L wrt the language L' is the language $L - L' = \{u \mid \exists v \in L' : uv \in L\}$. We abuse notation and write $L - w$ to denote $L - \{w\}$, for a word $w \in \Sigma^*$.

A *finite automaton* (FA) over an alphabet Σ is a quadruple $\mathcal{A} = (Q, \delta, I, F)$ where Q is a finite set of states, $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions, $I \subseteq Q$ is a set of *initial* states, and $F \subseteq Q$ is a set of *final* states. The *pre-image* of a state $q \in Q$ over $a \in \Sigma$ is the set of states $pre_{[a]}(q) = \{q' \mid (q', a, q) \in \delta\}$, and it is the set $pre_{[a]}(S) = \bigcup_{q \in S} pre_{[a]}(q)$ for a set of states S.

The language $\mathcal{L}(q)$ of a state $q \in Q$ is the set of words that can be read along a run ending in q, i.e. all words $a_1 \cdots a_n$, for $n \geq 0$, such that δ contains transitions $(q_0, a_1, q_1), \ldots, (q_{n-1}, a_n, q_n)$ with $q_0 \in I$ and $q_n = q$.[4] The language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is then the union $\bigcup_{q \in F} \mathcal{L}(q)$ of languages of its final states.

[4] Note that we speak about *left* languages of states here and throughout the entire paper, i.e. about sets of words accepted *at* a state, not accepted *from* a state.

3 WS1S

In this section, we give a minimalistic introduction to the *weak monadic second-order logic of one successor* (WS1S) and outline its classical decision procedure based on representing sets of models as regular languages and finite automata. See, for instance, Comon *et al.* [20] for a more thorough introduction.

3.1 Syntax and Semantics of WS1S

WS1S allows quantification over second-order *variables*, which we denote by upper-case letters $X, Y, \ldots$, that range over finite subsets of $\mathbb{N}_0$. Atomic formulae are of the form (i) $X \subseteq Y$, (ii) $\mathrm{Sing}(X)$, (iii) $X = \{0\}$, and (iv) $X = Y + 1$. Formulae are built from the atomic ones using the logical connectives $\wedge, \vee, \neg$, and the quantifier $\exists X$. (The $\forall X$ quantifier can be, as usual, obtained using $\neg\exists X\neg$.) A *model* of a WS1S formula $\varphi(\mathbb{X})$ is an assignment $\rho : \mathbb{X} \to 2^{\mathbb{N}_0}$ of the free variables $\mathbb{X}$ of φ to finite subsets of $\mathbb{N}_0$ for which the formula is *satisfied*, written $\rho \models \varphi$. Satisfaction of atomic formulae is defined as follows: (i) $\rho \models X \subseteq Y$ iff $\rho(X) \subseteq \rho(Y)$, (ii) $\rho \models \mathrm{Sing}(X)$ iff $\rho(X)$ is a singleton set, (iii) $\rho \models X = \{0\}$ iff $\rho(X) = \{0\}$, and (iv) $\rho \models X = Y + 1$ iff $\rho(X) = \{y + 1 \mid y \in \rho(Y)\}$. Satisfaction for formulae obtained using Boolean connectives is defined as usual. A formula φ is *valid*, written $\models \varphi$, iff all assignments of its free variables to finite subsets of $\mathbb{N}_0$ are models, and *satisfiable* if it has a model. W.l.o.g. we assume that, in a formula, each variable is quantified at most once.

3.2 Models as Words

Let $\mathbb{X}$ be a finite set of variables. A *symbol* τ over $\mathbb{X}$ is a mapping of all variables in $\mathbb{X}$ to either 0 or 1, e.g. $\tau = \{X_1 \mapsto 0, X_2 \mapsto 1\}$ for $\mathbb{X} = \{X_1, X_2\}$, which we will write as $\tau = \begin{smallmatrix} X_1 : 0 \\ X_2 : 1 \end{smallmatrix}$ below. The set of all symbols over $\mathbb{X}$ is denoted as $\Sigma_{\mathbb{X}}$. For any $\mathbb{X}$ (even empty), we use $\bar{0}$ to denote the symbol in $\Sigma_{\mathbb{X}}$ that maps all variables from $\mathbb{X}$ to 0, i.e. $\bar{0} = \{X \mapsto 0 \mid X \in \mathbb{X}\}$. For any set of symbols $\mathbb{Y}$ and any two symbols τ_1, τ_2 over $\mathbb{X}$, we write $\tau_1 \sim_{\mathbb{Y}} \tau_2$ iff $\forall X \in \mathbb{X} \setminus \mathbb{Y} : \tau_1(X) = \tau_2(X)$, i.e. the two symbols differ (at most) in the value of variables in $\mathbb{Y}$. The relation $\sim_{\mathbb{Y}}$ is generalized to words such that $w_1 \sim_{\mathbb{Y}} w_2$ iff $|w_1| = |w_2|$ and $\forall 1 \leq i \leq |w_1| : w_1[i] \sim_{\mathbb{Y}} w_2[i]$.

An assignment $\rho : \mathbb{X} \to 2^{\mathbb{N}_0}$ may be encoded as a word w_ρ of symbols over $\mathbb{X}$ in the following way: w_ρ contains 1 in the i-th position of the row for X iff $i \in X$ in ρ. Notice that there exists an infinite number of encodings of ρ: the shortest encoding is w^s_ρ of the length $n + 1$, where n is the largest number appearing in any of the sets that is assigned to a variable of $\mathbb{X}$ in ρ, or -1 when all these sets are empty. The rest of the encodings are all those corresponding to w^s_ρ extended with an arbitrary number of $\bar{0}$'s appended to its end. For example, $\begin{smallmatrix} X_1 : 0 \\ X_2 : 1 \end{smallmatrix}$, $\begin{smallmatrix} X_1 : 00 \\ X_2 : 10 \end{smallmatrix}$, $\begin{smallmatrix} X_1 : 000 \\ X_2 : 100 \end{smallmatrix}$, $\begin{smallmatrix} X_1 : 000\ldots0 \\ X_2 : 100\ldots0 \end{smallmatrix}$ are all encodings of the assignment $\rho = \{X_1 \mapsto \emptyset, X_2 \mapsto \{0\}\}$. We use $\mathcal{L}(\varphi) \subseteq \Sigma^*_{\mathbb{X}}$ to denote the language of all encodings of a formula φ's models, where $\mathbb{X}$ are the free variables of φ.

Let π be an operator that, for a symbol $\tau \in \Sigma_{\mathbb{X}}$ and a set $\mathbb{Y}$ of variables, returns the set $\pi_{\mathbb{Y}}(\tau) = \{\tau' \in \Sigma_{\mathbb{X}\cup\mathbb{Y}} \mid \tau' \sim_{\mathbb{Y}} \tau\}$ of symbols that are the same as τ up to that they

may differ in the membership of variables in $\mathbb{Y}$. For a language $L \subseteq \Sigma_{\mathbb{X}}^*$, we define

$$\pi_{\mathbb{Y}}(L) = \{w \mid \exists \tau_1 \cdots \tau_n \in L : w \in \pi_{\mathbb{Y}}(\tau_1) \,.\, \cdots \,.\, \pi_{\mathbb{Y}}(\tau_n)\} \tag{1}$$

to be the language over $\Sigma_{\mathbb{X}\cup\mathbb{Y}}$ containing the words that are the same as the words of L up to the occurrence of variables from $\mathbb{Y}$ in their symbols. Seen from the point of view of encodings of sets of assignments, $\pi_{\mathbb{Y}}(L)$ encodes all assignments that may differ from those encoded by L (only) in the values of variables from $\mathbb{Y}$. If $\mathbb{Y}$ is disjoint with the free variables of φ, then $\pi_{\mathbb{Y}}(\mathcal{L}(\varphi))$ corresponds to the so-called cylindrification of $\mathcal{L}(\varphi)$, and if it is their subset, then $\pi_{\mathbb{Y}}(\mathcal{L}(\varphi))$ corresponds to the so-called projection [20]. We use π_Y to denote $\pi_{\{Y\}}$ for a variable Y.

Let $\mathit{free}(\varphi)$ be the set of free variables of φ, and let $\mathcal{L}^{\mathbb{X}}(\varphi) = \pi_{\mathbb{X}\setminus \mathit{free}(\varphi)}(\mathcal{L}(\varphi))$ be the language $\mathcal{L}(\varphi)$ cylindrified wrt those variables of $\mathbb{X}$ that are not free in φ. Let φ and ψ be formulae whose both free and non-free variables are from $\mathbb{X}$, and assume that $\mathcal{L}^{\mathbb{X}}(\varphi)$ and $\mathcal{L}^{\mathbb{X}}(\psi)$ are languages of encodings of their models cylindrified wrt $\mathbb{X}$. Languages of formulae obtained from φ and ψ using logical connectives are the following:

$$\mathcal{L}^{\mathbb{X}}(\varphi \vee \psi) = \mathcal{L}^{\mathbb{X}}(\varphi) \cup \mathcal{L}^{\mathbb{X}}(\psi) \tag{2}$$

$$\mathcal{L}^{\mathbb{X}}(\varphi \wedge \psi) = \mathcal{L}^{\mathbb{X}}(\varphi) \cap \mathcal{L}^{\mathbb{X}}(\psi) \tag{3}$$

$$\mathcal{L}^{\mathbb{X}}(\neg\varphi) = \Sigma_{\mathbb{X}}^* \setminus \mathcal{L}^{\mathbb{X}}(\varphi) \tag{4}$$

$$\mathcal{L}^{\mathbb{X}}(\exists Y : \varphi) = \pi_Y(\mathcal{L}^{\mathbb{X}}(\varphi)) - \bar{0}^* \tag{5}$$

Equations (2)-(4) above are straightforward: Boolean connectives translate to the corresponding set operators over the universe of encodings of assignments of variables in $\mathbb{X}$. Existential quantification $\exists Y : \varphi$ translates into a composition of two language transformations. First, π_Y makes the valuation of $Y \in \mathbb{X}$ in φ arbitrary, which intuitively corresponds to forgetting everything about Y's value (notice that this is a different use of π_Y than the cylindrification since here Y *is* a free variable of φ). The second step, removing suffixes of $\bar{0}$'s from the model encodings, is necessary since $\pi_Y(\mathcal{L}^{\mathbb{X}}(\varphi))$ might be missing some encodings of models of $\exists Y : \varphi$. For example, suppose that $\mathbb{X} = \{X, Y\}$ and the only model of φ is $\{X \mapsto \{0\}, Y \mapsto \{1\}\}$, yielding $\mathcal{L}^{\mathbb{X}}(\varphi) = \begin{smallmatrix} X:10 \\ Y:01 \end{smallmatrix}\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]^*$. Then $\pi_Y(\mathcal{L}^{\mathbb{X}}(\varphi)) = \begin{smallmatrix} X:10 \\ Y:?? \end{smallmatrix}\left[\begin{smallmatrix}0\\?\end{smallmatrix}\right]^*$ does not contain the shortest encoding $\begin{smallmatrix} X:1 \\ Y:? \end{smallmatrix}$ (where each '?' denotes an arbitrary value) of the only model $\{X \mapsto \{0\}\}$ of $\exists Y : \varphi$. It only contains its variants with at least one $\bar{0}$ appended to it. This generally happens for models of φ where the largest element of the value of a quantified variable Y is a number larger than any element of the value of a free variable of $\exists Y : \varphi$. The role of $-\bar{0}^*$ is to include the missing encodings of models with a smaller number of trailing $\bar{0}$'s into the language.

The standard approach to decide satisfiability of a WS1S formula φ with the set of variables $\mathbb{X}$ is to construct an automaton $\mathcal{A}_\varphi$ accepting $\mathcal{L}^{\mathbb{X}}(\varphi)$ and check emptiness of its language. The construction starts with simple pre-defined automata $\mathcal{A}_\psi$ for φ's atomic formulae ψ (see e.g. [20] for more details) accepting cylindrified languages $\mathcal{L}^{\mathbb{X}}(\psi)$ of models of ψ. These are simple regular languages. The construction then continues by inductively constructing automata $\mathcal{A}_{\varphi'}$ accepting languages $\mathcal{L}^{\mathbb{X}}(\varphi')$ of models for all other sub-formulae φ' of φ, using equations (2)–(5) above. The language operators used in the rules are implemented using standard automata-theoretic constructions (see [20]).

4 Satisfiability via Language Term Evaluation

In this section, we describe the basic principles of our algorithm for deciding satisfiability of a WS1S formula φ with a set of variables $\mathbb{X}$. To simplify the presentation, we will consider the particular case of *ground* formulae (i.e. formulae without free variables), for which satisfiability corresponds to validity. Satisfiability of a formula with free variables can be reduced to this case by prefixing it with existential quantification over the free variables. If φ is ground, the language $\mathcal{L}^{\mathbb{X}}(\varphi)$ is either $\Sigma_{\mathbb{X}}^*$ in the case φ is valid, or empty if φ is invalid. Then, to decide the validity of φ, it suffices to test if $\epsilon \in \mathcal{L}^{\mathbb{X}}(\varphi)$. Our algorithm is designed to allow on-the-fly emptiness checking, lazy evaluation, and a use of subsumption to prune the search space. These techniques are described in the next section.

We reason over the so-called *language term* t_φ, a symbolic representation of the language $\mathcal{L}^{\mathbb{X}}(\varphi)$ that encodes the construction of $\mathcal{A}_\varphi$. A language term is a (finite) term generated by the following grammar:

$$t ::= \mathcal{A} \mid t \cup t \mid t \cap t \mid \overline{t} \mid \pi_X(t) \mid t - \bar{0}^* \mid t - \tau \tag{6}$$

where $\mathcal{A}$ is a finite automaton over the alphabet $\Sigma_{\mathbb{X}}$, $X \in \mathbb{X}$, and $\tau \in \Sigma_{\mathbb{X}}$. We call a term of the form $t - \bar{0}^*$ a *star-quotient*, a term $t - \tau$ a *symbol-quotient* (if no confusion may arise, both of them can be denoted as quotient terms), and we call it a *simple* term otherwise. A term is *quotient-free* if it does not contain any quotients, i.e. all its subterms are simple. The *language* $\mathcal{L}(t)$ *represented by a term* t is obtained by taking the languages of the automata in its leaves and combining them using the term operators. We call *language equivalent* terms with the same language. We will often abuse the notation and write t to denote its language $\mathcal{L}(t)$.

The formula φ is transformed into the term t_φ by replacing every atomic subformula ψ in φ by the automaton $\mathcal{A}_\psi$ accepting $\mathcal{L}^{\mathbb{X}}(\psi)$, and by replacing the logical connectives with language operators according to equations (2)–(5) of Section 3.2. Notice that symbol-quotients are absent from t_φ—they will be used later to describe internal states of our algorithm.

Language terms and automata. We will later present our algorithm for deciding validity of φ as an evaluation of the query $\epsilon \in t_\varphi$, using term rewriting rules that preserve the term's language. As we have already noted in the introduction, the algorithm can, however, also be interpreted as an on-the-fly test of ϵ-membership in the language of an automaton $\mathcal{A}_\varphi$ accepting $\mathcal{L}(\varphi)$. For every term t, it is possible to construct the automaton $\mathcal{A}_t$ accepting $\mathcal{L}(t)$, s.t. $\mathcal{A}_t$ is in the form of a right quotient automaton, i.e. an automaton where every state corresponds to a right quotient of $\mathcal{L}(t)$, and *vice versa*. Each state q of $\mathcal{A}_t$ is represented using a term t_q with the property that $\mathcal{L}(q) = \mathcal{L}(t_q)$. The transition relation of $\mathcal{A}_t$ can be constructed by starting with the state corresponding to t and iterating the predecessor construction: the predecessor of a term t' in $\mathcal{A}_t$ via a symbol a is a term t_a such that $\mathcal{L}(t_a) = \mathcal{L}(t') - a$.

4.1 Towards Membership by Term Rewriting

The ϵ-membership query on a quotient-free term is evaluated using the following equivalences. They allow us to reduce tests on terms to Boolean combinations of tests on their

sub-terms, and to eventually evaluate the tests on automata in the leaves.

$$\epsilon \in t_1 \cup t_2 \quad \text{iff} \quad \epsilon \in t_1 \text{ or } \epsilon \in t_2 \tag{7}$$
$$\epsilon \in t_1 \cap t_2 \quad \text{iff} \quad \epsilon \in t_1 \text{ and } \epsilon \in t_2 \tag{8}$$
$$\epsilon \in \overline{t_1} \quad \text{iff} \quad \text{not } \epsilon \in t_1 \tag{9}$$
$$\epsilon \in \pi_X(t_1) \quad \text{iff} \quad \epsilon \in t_1 \tag{10}$$
$$\epsilon \in \mathcal{A} = (Q, \delta, I, F) \quad \text{iff} \quad I \cap F \neq \emptyset \tag{11}$$

The ϵ-membership in this case can thus be evaluated by pushing the test to its leaves, using equivalences (7) to (10), where it is decided by testing intersection of the initial and final states of an automaton, using equivalence (11).

The term t_φ is, however, not necessarily quotient-free. Every quantified sub-formula of φ translates to a star-quotient sub-term $t - \bar{0}^*$ (cf. rule (5) in Section 3). When the ϵ-membership test reaches such a sub-term, we use a set of rewriting rules of the form $t_1 \to t_2$ to rewrite $t - \bar{0}^*$ to a language equivalent simple term t' on which some of the equivalences above apply.

In particular, the language of $t - \bar{0}^*$ equals $\bigcup_{i \geq 0} \mathcal{L}(t - \bar{0}^i)$ where $t - \bar{0}^i$ stands for the *i-th unfolding of* $t - \bar{0}^*$, which we define such that $t - \bar{0}^0 = t$ and $t - \bar{0}^i = (t - \bar{0}^{i-1}) - \bar{0}$ for $i > 0$. We will eliminate the star-quotient $-\bar{0}^*$ by iteratively extending the sequence $t \cup (t - \bar{0}) \cup (t - \bar{0}^2) \cup \ldots$ of symbol-quotients until the language of the whole union stabilizes (at which point it is equal to $\mathcal{L}(t - \bar{0}^*)$). The following rewriting rule implements one unfolding of the fixpoint computation:

$$t - \bar{0}^* \;\to\; \begin{cases} t & \text{if } \mathcal{L}(t - \bar{0}) \subseteq \mathcal{L}(t) \\ (t \cup (t - \bar{0})) - \bar{0}^* & \text{otherwise} \end{cases} \tag{12}$$

Rule (12) consists of two cases. The first case, applied when $\mathcal{L}(t - \bar{0}) \subseteq \mathcal{L}(t)$, is used to terminate the fixpoint computation. Because testing language inclusion of the terms precisely is infeasible (we would need to build the finite automata from the terms first and then test inclusion of their languages, which is a PSPACE-complete problem), we use a safe under-approximation of the test, as described later.

The other case of the rule introduces symbol-quotients, which also need to be eliminated in order for the equivalences (7)–(11) to apply. This is done by the following rules:

$$\pi_X(t) - \tau \quad \to \quad \bigcup_{\tau' \in \pi_X(\tau)} \pi_X(t - \tau') \tag{13}$$
$$(t_1 \cup t_2) - \tau \quad \to \quad (t_1 - \tau) \cup (t_2 - \tau) \tag{14}$$
$$(t_1 \cap t_2) - \tau \quad \to \quad (t_1 - \tau) \cap (t_2 - \tau) \tag{15}$$
$$\bar{t} - \tau \quad \to \quad \overline{t - \tau} \tag{16}$$
$$\mathcal{A} = (Q, \delta, I, F) - \tau \quad \to \quad \mathcal{A}' = (Q, \delta, I, \mathit{pre}_{[\tau]}(F)) \tag{17}$$

If t is quotient-free, then rules (13)–(16) applied to $t - \tau$ push the symbol-quotient down the structure of t towards automata in the leaves, where it is eliminated by rule (17), which replaces the final states of an automaton by their predecessors. The result is a quotient-free term that is language-equivalent to $t - \tau$; we denote such a term as $[t - \tau]$.

We call a *quotient of the term* t every term that is either t or equals $[t' - \tau]$ for some quotient t' of t and symbol τ.

To detect that $\mathcal{L}(t - \bar{0}) \subseteq \mathcal{L}(t)$, we use the following method. We say that two terms are *Boolean equivalent* if they can be obtained from one another by using standard commutativity and associativity laws for $\cup$ and $\cap$ and by removing duplicities from unions and intersections. Clearly, if t and t' are Boolean equivalent, then $\mathcal{L}(t) = \mathcal{L}(t')$. Therefore, we under-approximate the test of $\mathcal{L}(t - \bar{0}) \subseteq \mathcal{L}(t)$ by testing whether the terms $[t - \bar{0}]$ and $[t]$ are Boolean equivalent. Note that this is sound but, of course, incomplete. Based on the following lemma, however, it can be argued that this under-approximation is sufficient for the fixpoint unfolding to terminate.

Lemma 1. *A quotient-free term has only finitely many quotients that are not Boolean equivalent.*

Proof (sketch). By induction on the structure of t. The lemma clearly holds if t is an automaton. In this case, the quotient is evaluated by rule (17) that only changes final states. Since the automaton is finite-state, it has only finitely many different quotients.

Let now $t = t_1 \cup t_2$. Then, by rule (14), $[t - \tau]$ is obtained as $[t_1 - \tau] \cup [t_2 - \tau]$. The quotients of t therefore arise by combining a quotient of t_1 and a quotient of t_2. Because, by the induction hypothesis, t_1 and t_2 have only finitely many quotients, t has finitely many quotients too. The reasoning is analogous for t of the form $t_1 \cap t_2$ or $\neg t'$.

Finally, let $t = \pi_X(t')$. Then $[t - \tau]$ is evaluated as $\bigcup_{\tau' \in \pi_X(\tau)} \pi_X([t' - \tau'])$ by rule (13). Because, by the induction hypothesis, t' has finitely many quotients, the term t also has finitely many quotients. □

4.2 Basic Algorithm

This section gives a basic algorithm that uses the rules given above to decide the validity of a ground formula φ. In this algorithm, we do not use subsumption, lazy evaluation, or any other of the optimizations described in further sections. The algorithm performs the following three basic steps: (1) It translates the formula φ into the term t_φ as described at the beginning of Section 4. (2) It rewrites the term t_φ into the quotient-free form $[t_\varphi]$ using the procedure described below. (3) It tests whether $\epsilon \in [t_\varphi]$ using equivalences (7)–(10) and returns the result of the test.

The rewriting of a term t into the quotient-free form $[t]$ proceeds using the following recursive procedure. For terms $t_\cup = t_1 \cup t_2$, $t_\cap = t_1 \cap t_2$, $t_\neg = \bar{t}$, and $t_\pi = \pi_X(t)$, the procedure returns $[t_\cup] = [t_1] \cup [t_2]$, $[t_\cap] = [t_1] \cap [t_2]$, $[t_\neg] = \overline{[t]}$, and $[t_\pi] = \pi_X([t])$ respectively. For a star-quotient term $t_0 - \bar{0}^*$, the procedure is more complex:

1. If t_0 is quotient-free, then rewriting $t_0 - \bar{0}^*$ into $[t_0 - \bar{0}^*]$ is an iterative fixpoint computation. At the beginning of the $(k+1)$-st iteration, for $k \geq 0$, the term is in the form $(t_0 \cup \cdots \cup t_k) - \bar{0}^*$ where each t_{i+1} is the quotient-free term $[t_i - \bar{0}]$, language equivalent to $t_0 - \bar{0}^{i+1}$. The $(k+1)$-st iteration applies rule (12) in the following way. It first uses rules (13)–(17) to rewrite the term $t_k - \bar{0}$ into the quotient-free form $t_{k+1} = [t_k - \bar{0}]$. The language inclusion test $\mathcal{L}(t_1 \cup \cdots \cup t_{k+1}) \subseteq \mathcal{L}(t_0 \cup \cdots \cup t_k)$ of rule (12) is then underapproximated by

testing Boolean equivalence of the terms, for which it is enough to test whether t_{k+1} is Boolean equivalent to some t_i, for $i \leq k$. If the test returns *false*, the computation continues with $(t_0 \cup \cdots \cup t_{k+1}) - \bar{0}^*$ into the $(k+2)$-nd iteration. Otherwise the fixpoint computation ends with the final quotient-free term $[t_0 - \bar{0}^*] = (t_0 \cup \cdots \cup t_k)$, which is language equivalent to the original $t_0 - \bar{0}^*$. Lemma 1 implies that the used approximation of the inclusion test of rule (12) by Boolean equivalence is enough to guarantee termination of the fixpoint computation.
2. If t_0 is not quotient-free, the quotient-free term $[t_0 - \bar{0}^*]$ is computed by recursively calling the star-quotient elimination procedure on all star-quotient sub-terms $t' - \bar{0}^*$ and replacing them by the resulting quotient-free language equivalent terms $[t' - \bar{0}^*]$. This yields a quotient-free term $[t_0]$ language equivalent to t_0, and $[t_0 - \bar{0}^*]$ is then computed as $[[t_0] - \bar{0}^*]$ by the fixpoint computation from point 1.

5 Efficient Algorithm

A key advantage of the approach of Section 4 is that it can be easily modified such that it does *not* construct a symbolic representation of the entire automaton $\mathcal{A}_\varphi$ in order to test ϵ-membership in its language afterwards. The test can be done *on the fly*, while constructing a symbolic encoding of $\mathcal{A}_\varphi$'s state space. Moreover, it allows one to use the following heuristics that prune out the state space irrelevant wrt the test:

1. *Lazy evaluation*: the construction of the symbolic, term-based representation of $\mathcal{A}_\varphi$ is done on-demand; as soon as the test of ϵ-membership can be answered, we stop.
2. *Subsumption*: we can discard parts of the search space represented by some terms if the information they carry wrt the ϵ-membership is represented by other terms too.

We will now discuss how these optimizations can be implemented as a modification of the basic algorithm of Section 4.2.

5.1 Lazy Evaluation

The basic algorithm first computes the quotient-free term $[t]$ and then follows by testing whether $\epsilon \in [t]$. As we show now, it can, however, be easily optimized to perform the ϵ-membership test *on the fly*. This allows us to use the following techniques of *lazy evaluation*. First, ϵ-membership tests for unions (equation (7)) can be *short-circuited* in the obvious way: When testing whether $\epsilon \in t_1 \cup t_2$, we first evaluate, e.g., the test $\epsilon \in t_1$, and when it holds, we can completely avoid exploring t_2. When testing $\epsilon \in t_1 \cap t_2$, we can proceed analogously.

Second, we can also reduce the number of evaluated elements of a star-quotient term by computing them *on demand*, i.e. only when required by the test. The lazy evaluation of ϵ-membership in a star-quotient term $t - \bar{0}^*$ proceeds as follows. We keep the intermediate states of unfolding of the star-quotient term in the form $(t_0 \cup \cdots \cup t_k) - \bar{0}^*$ as described in Section 4.2 with $t = t_0$. The difference is that we let the ϵ-membership test to be evaluated on the terms $t_0, ..., t_k$, and the rewriting rules are allowed to rewrite them.

In particular, with every $(k+1)$-st unfolding of the fixpoint computation, for $k \geq 0$, we start by evaluating whether $\epsilon \in t_k$. During the evaluation, when pushing quotients

into t_k, the term t_k may be rewritten into a term t'_k. Since rewriting of t_k again uses lazy evaluation and stops as soon as the answer to the ϵ-membership query is obtained, t'_k is, in general, not equal to $[t_k]$ (unlike in the basic algorithm of Section 4.2). It may still contain unhandled quotients. If the ϵ-membership test on t_k fails, the term t'_k with partially eliminated quotients is tested for Boolean equivalence with the previously existing elements in the fixpoint's state. If the test succeeds, the fixpoint computation ends, otherwise the next disjunct t_{k+1} is added in the form $t'_k - \bar{0}$, and the computation continues by the $(k+2)$-nd iteration. Conversely, if the ϵ-membership test on t_k succeeds, the fixpoint computation stops and the top-level membership query returns *true*.

Moreover, the intermediate state of the fixpoint computation $(t_0 \cup \cdots \cup t_k) - \bar{0}^*$ is maintained in the form of a *continuation*, which is invoked when more unfoldings of the fixpoint are needed. This happens when we are dealing with nested fixpoint computations where discharging a higher iteration of the outer fixpoint may require more iterations of the inner fixpoint (and it would be wasteful to recompute all the already computer iterations of the inner fixpoint).

5.2 Subsumption

Our next technique for reducing the explored state space is based on the notion of *subsumption* between terms, which is similar to the subsumption used in antichain-based universality and inclusion checking over finite automata [10]. We define subsumption as the relation $\sqsubseteq$ on terms in the following way:

$$t_1 \cup \cdots \cup t_n \sqsubseteq t'_1 \cup \cdots \cup t'_m \iff \forall 1 \leq i \leq n \; \exists 1 \leq j \leq m \,.\, t_i \sqsubseteq t'_j \quad (18)$$

$$t_1 \cap \cdots \cap t_n \sqsubseteq t'_1 \cap \cdots \cap t'_m \iff \forall 1 \leq j \leq m \; \exists 1 \leq i \leq n \,.\, t_i \sqsubseteq t'_j \quad (19)$$

$$\overline{t} \sqsubseteq \overline{t'} \iff t \sqsupseteq t' \quad (20)$$

$$\pi_X(t) \sqsubseteq \pi_X(t') \iff t \sqsubseteq t' \quad (21)$$

$$t - \bar{0}^* \sqsubseteq t' - \bar{0}^* \iff t \sqsubseteq t' \quad (22)$$

$$\mathcal{A} = (Q, \delta, I, F) \sqsubseteq \mathcal{A}' = (Q, \delta, I, F') \iff F \subseteq F' \quad (23)$$

The subsumption relation $\sqsubseteq$ under-approximates language inclusion and can therefore be used as an implementation of the $\mathcal{L}(t - \bar{0}) \subseteq \mathcal{L}(t)$ test of rule (12), which is more precise than the Boolean equivalence introduced in Section 4.1. Moreover, we use it to prune star-quotient terms $(t_1 \cup \cdots \cup t_k) - \bar{0}^*$ that are intermediate states of fixpoint computations. In particular, we remove disjuncts that are subsumed by newly added ones, keeping the intermediate states in the form of *antichains* of $\sqsubseteq$-incomparable sub-terms. The pruning essentially corresponds to using the following rewriting rule:

$$t \cup t' \quad \to \quad t \qquad \text{if } t' \sqsubseteq t. \quad (24)$$

Obviously, the rule preserves the language of the term, i.e. $\mathcal{L}(t \cup t') = \mathcal{L}(t)$.

Notice that a subsumption test cannot propagate across quotient terms, apart from rule (22). To make subsumption more powerful, we allow it to use rewriting rules (13)–(17) on both sides of $\sqsubseteq$. This comes with a trade-off since subsumption can now trigger evaluation of nested fixpoints that could possibly stay folded. Nevertheless, according to our empirical experience, letting subsumption unfold star-quotients still pays off.

Further, note that the $\sqsubseteq$ relation can only hold between terms that are built over the same atomic automata (up to different sets of final states) that have almost the same

Boolean structure (apart from the number of elements in unions that were introduced for unfolding of a star-quotient). We therefore annotate terms with the automata at their leaves, and skip evaluating subsumption for terms with inconsistent annotations. This allows us to check subsumption much faster.

5.3 Correctness

We now state the correctness of our algorithm, including lazy evaluation and subsumption from Sections 5.1 and 5.2.

Theorem 1. *For a given closed formula φ, the algorithm described in Section 5 terminates and returns $true$ iff $\models \varphi$.*

Proof (sketch). The soundness of our algorithm can be seen from the soundness of the equivalences used to evaluate ϵ-membership (which are trivial equivalences from set theory and automata theory), and from the fact that our rewriting rules preserve the language of the original term t_φ (basic automata theory). Correctness of the automata of the atomic formulae comes from the classical decision procedure [20]. Since $\epsilon \in t_\varphi$ iff $\models \varphi$ (for a ground formula φ), when the ϵ-membership test concludes, we have the correct answer to the validity of φ.

Termination of the basic algorithm was argued in Section 4.2. It follows as a consequence of Lemma 1, which implies that every term can be rewritten into a quotient-free form in a finite number of steps.. The test of ϵ-membership on a quotient-free term also terminates in a finite number of steps (in each recursive call, the test decreases the depth of the term it is evaluated on). Termination of the basic algorithm implies termination of the efficient algorithm that uses lazy evaluation and subsumption as described in Sections 5.1 and 5.2. The reason is that using subsumption only allows to detect sooner that an unfolding of a star-quotient term is at a fixpoint. Using lazy evaluation only enables skipping parts of the computation. Therefore, if the basic algorithm terminates, the efficient algorithm must terminate too. □

6 Further Optimizations

In this section, we discuss two techniques to optimize our approach. First, we utilise usual optimizations by transformation of the formula, which translate directly to the structure of the language terms generated by the algorithm. The technique that we found most helpful is anti-prenexing. The second technique is a combination of our algorithm with the classical decision procedure, which is used to replace language sub-terms by language-equivalent minimal automata. Both optimizations have a significant positive impact on the performance of our implementation.

6.1 Anti-prenexing

Anti-prenexing transforms a formula φ into an equivalent formula φ' using the following logical identities:

$$\exists X : \varphi_1 \vee \varphi_2 \quad \equiv \quad (\exists X : \varphi_1) \vee (\exists X : \varphi_2) \tag{25}$$

$$\exists X : \varphi_1 \wedge \varphi_2 \quad \equiv \quad (\exists X : \varphi_1) \wedge \varphi_2 \qquad \text{if } X \notin \mathit{free}(\varphi_2) \tag{26}$$

$$\exists X : \varphi_1 \quad \equiv \quad \varphi_1 \qquad \text{if } X \notin \mathit{free}(\varphi_1) \tag{27}$$

By moving a quantifier down in the abstract syntax tree of a formula, we are effectively speeding up the fixpoint computation induced by the moved quantifier.

Intuitively, moving quantifier down the structure of the formula into sub-formulae causes that one costlier fixpoint computation is replaced by several fixpoint computations in the sub-formulae. This is usually helpful for two reasons: (1) If the smaller fixpoint computations unfolds a union of terms of some structure, then the original larger fixpoint computation would unfold a union of terms with a combined structure. The number of terms that the larger fixpoint computation generates can therefore be (in the worst case) combinatorially larger than the sum of terms generated by all the smaller fixpoint computations. (2) When using the lazy evaluation strategy, it often happens that some of the smaller fixpoint computations will never be unfolded.

We also preprocess the formula by moving negations down the structure towards the leaves, using De Morgan's laws $\neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2$ and $\neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$, and double negation elimination $\neg\neg\varphi \equiv \varphi$. For the automata in the leaves, the negation is eliminated by complementing the automata, which is cheap due to their small size (they can also be precomputed). Eliminating negation from the terms, or only moving it down the structure of the terms, helps anti-prenexing push quantifiers deeper.

6.2 Translating a Sub-term into an Automaton

Our method allows to replace a sub-term of a language term by a language-equivalent automaton and test ϵ-membership on the resulting term. The advantage of this is that explicit automata have a simpler structure and can be efficiently minimized. The minimized automaton can have significantly smaller state space than the state space of quotients of the original sub-term. It requires, however, to construct the whole automaton first. Replacing a sub-term with its minimized version thus represents a trade-off between the lazy evaluation and subsumption on one side, and minimization of the classical procedure on the other side. A positive aspect of this technique is that it allows to leverage the extremely efficient implementation of formula to automaton translation implemented in MONA. The effect of this technique depends on the choice of sub-terms to be translated. In our experiments, we are mostly using a simple heuristic that translates maximal quotient-free sub-terms. Studying the effect of different strategies of choosing the sub-terms could, however, be fruitful.

7 Experiments

We implemented a prototype of the approach presented in this paper in a tool GASTON,[5] written in C++, and evaluated it on several sets of benchmarks found in the literature. We compared its performance with the tools MONA [7] and DWINA [17]. Our tool uses the frontend of MONA to parse input formulae and construct their abstract syntax trees, and also to construct automata for sub-formulae (as mentioned in Section 6.2).

[5] The name was chosen to pay homage to Gaston, an Africa-born brown fur seal who escaped the Prague Zoo during the floods in 2002 and made a heroic journey for freedom of over 300 km to Dresden. There he was caught and subsequently died due to exhaustion and infection.

We chose three sets of benchmarks that were used by the authors of DWiNA. The first benchmark, `Strand1`, is from [1], which uses WS1S within an algorithm for shape analysis, and the second benchmark, `Strand2`, is from the follow-up work [2], where the algorithm is tuned to produce formulae that are more suitable for MONA. Both previous benchmarks are available at [21]. We use `sl` to denote benchmarks with sorted lists and `bs` to denote benchmarks with bubble sort. Unfortunatelly, DWiNA could not run on the `Strand1` benchmark due to its lack of support for some features of the input language. The third benchmark, `Antichains`, contains artificially generated WS1S formulae from the evaluation of DWiNA [17].

We compared the running times (given in seconds) that the three tools needed to decide the validity of the formulae, and also compared the sizes of the generated state spaces using suitable metrics. For MONA, we were counting the overall number of states of the generated automata, for DWiNA the number of its generated symbolic states, and for GASTON the overall number of generated sub-terms. Our experiments were performed on an Intel Core i7-4770@3.4 GHz processor with 32 GiB RAM. Table 1 is a digest of interesting results (see Appendix A for the results of all benchmarks). We use ∞ to represent that the running time exceeded 10 minutes or that the tool ran out of available memory.

As mentioned above, GASTON uses MONA to build automata for sub-formulae of the tested formula. In the automata representation of MONA, MTBDDs are used to encode transition relations of automata by representing *post*-images of states. Because in our approach, we compute *pre*-images of states, we had to deal with the issue of inverting the transition relation. This could be done by generating the inverted relation directly from MONA. Unfortunately, we were not able to hack into MONA and implement this functionality, and we therefore invert the automata only after they are created. The overhead of this conversion can be quite substantial. Because the conversion is only an implementation feature of our prototype and not inherent to the procedure, to better show the capabilities of our approach, we subtracted the conversion time from the total run-time of our tool (we still include the time it took MONA to build the automata).

The results show that the performance of various tools differs according to the benchmark set. GASTON is a clear winner over MONA on benchmarks from `Strand1`. On the benchmarks from `Strand2`, which are benchmarks made easier to process by MONA, GASTON still in the majority of cases beats both MONA and DWiNA. In a few cases, it does, however, run out of the available time or memory. For the generated benchmarks from `Antichains`, GASTON and DWiNA still clearly win over MONA, each of them being better on different classes of formulae.

8 Conclusion

We have presented a new decision procedure for WS1S that is based on rewriting of the so-called language terms representing the language of all models of the formula. It can also be seen as a construction of the right quotient automaton accepting the same language, while on-the-fly testing membership of ϵ in its language. The on-the-fly nature of the algorithm allows us to reduce the state space of the automaton/term by using lazy evaluation and subsumption. Several other heuristics that significantly improve the performance of the method have also been proposed. On our benchmark formulae, our implementation performs in the majority of cases significantly better than the well-known

Table 1. A digest of the results of our experiments

Benchmark	MONA		DWINA		GASTON	
	Time	**Space**	**Time**	**Space**	**Time**	**Space**
Strand1						
bs-loop-else	94.73	168909			0.69	1056
bs-loop-if-else	358.81	390593			1.63	1059
bs-loop-if-if	398.63	442138			1.97	2084
sl-insert-after-loop	∞	∞			1.41	1108
sl-insert-before-head	61.56	106858			8.89	259
sl-insert-before-loop	13.38	36620			0.33	258
sl-insert-error-error	11.81	32238			0.21	330
sl-insert-in-loop	159.31	233264			0.72	543
sl-reverse-after-loop	8.11	22327			0.05	175
sl-search-after-loop	5.49	17656			0.06	143
Strand2						
bs-loop-else	1.13	15170	0.01	19	0.19	454
bs-loop-if-else	9.49	63444	0.23	234	0.68	531
bs-loop-if-if	31.17	130555	1.14	28	2.64	604
sl-insert-after-loop	0.12	2584	0.01	36	0.01	183
sl-insert-before-head	0.02	645	0.01	45	0.01	88
sl-insert-before-loop	0.06	1415	0.01	47	∞	∞
sl-insert-error-error	0.05	1415	0.01	47	∞	∞
sl-insert-in-loop	0.32	5963	0.01	59	0.01	273
sl-reverse-after-loop	0.09	1886	0.01	110	0.01	138
sl-search-after-loop	0.03	1012	0.01	274	0.01	95
Antichains						
horn-trans12	189.14	2766	0.01	1347	0.09	10
horn-trans13	409.56	3564	0.02	1745	0.24	10
horn-trans14	∞	∞	0.04	2215	0.88	10
horn-trans15	∞	∞	0.06	2763	3.39	10
horn-trans16	∞	∞	0.10	3395	13.37	10
horn-trans17	∞	∞	0.15	4117	53.08	10
horn-trans18	∞	∞	0.23	4935	212.68	10
horn-trans19	∞	∞	0.32	5855	∞	∞
set-closed01	0.01	104	0.01	26	0.01	67
set-closed02	0.01	195	0.01	118	0.01	478
set-closed03	0.08	1376	0.01	1488	0.03	7127
set-closed04	153.09	52697	∞	∞	4.00	278696
set-closed05	∞	∞	∞	∞	∞	∞
set-singletons04	268.19	66024	0.03	167	0.01	3
set-singletons05	∞	∞	0.15	350	0.01	3
set-singletons06	∞	∞	0.82	857	0.01	3
set-singletons07	∞	∞	∞	∞	0.01	3
set-singletons08	∞	∞	∞	∞	0.01	3
set-singletons09	∞	∞	∞	∞	0.01	3
set-singletons10	∞	∞	∞	∞	0.05	3
set-singletons11	∞	∞	∞	∞	0.17	3
set-singletons12	∞	∞	∞	∞	∞	∞

tool MONA and also than the tool DWINA. As an immediate future work, we intend to turn our prototype into a mature tool that would be widely useful. We also wish to explore other heuristics that might be helpful in reducing the explored state space even more. Based on our experimental results, we believe that this will significantly extend applicability of WS1S in practice. Further, we also plan to explore applicability of our approach to WSkS and other related logics.

References

1. Madhusudan, P., Parlato, G., Qiu, X.: Decidable logics combining heap structures and data. In: POPL 2011, ACM (2011) 611–622
2. Madhusudan, P., Qiu, X.: Efficient decision procedures for heaps using STRAND. In: SAS 2011. Volume 6887 of Lecture Notes in Computer Science., Springer (2011) 43–59
3. Iosif, R., Rogalewicz, A., Šimáček, J.: The tree width of separation logic with recursive definitions. In: CADE 2013. Volume 7898 of Lecture Notes in Computer Science., Springer (2013) 21–38
4. Chin, W., David, C., Nguyen, H.H., Qin, S.: Automated verification of shape, size and bag properties via user-defined predicates in separation logic. Sci. Comput. Program. **77**(9) (2012) 1006–1036
5. Zee, K., Kuncak, V., Rinard, M.C.: Full functional verification of linked data structures. In: POPL 2008, ACM (2008) 349–361
6. Hamza, J., Jobstmann, B., Kuncak, V.: Synthesis for regular specifications over unbounded domains. In: FMCAD 2010, IEEE (2010) 101–109
7. Elgaard, J., Klarlund, N., Møller, A.: MONA 1.x: new techniques for WS1S and WS2S. In: CAV 1998. Volume 1427 of Lecture Notes in Computer Science., BRICS, Department of Computer Science, Aarhus University, Springer (1998) 516–520
8. Meyer, A.R.: Weak monadic second order theory of successor is not elementary-recursive. In Parikh, R., ed.: Logic Colloquium—Symposium on Logic Held at Boston, 1972–73. Volume 453 of Lecture Notes in Mathematics., Springer (1972) 132–154
9. Wies, T., Muñiz, M., Kuncak, V.: An efficient decision procedure for imperative tree data structures. In Bjørner, N., Sofronie-Stokkermans, V., eds.: CADE 2011. Volume 6803 of Lecture Notes in Computer Science., Springer (2011) 476–491
10. Wulf, M.D., Doyen, L., Henzinger, T.A., Raskin, J.F.: Antichains: A new algorithm for checking universality of finite automata. In: CAV'06. Volume 4144 of LNCS., Springer (2006) 17–30
11. Klarlund, N., Møller, A., Schwartzbach, M.I.: MONA implementation secrets. International Journal of Foundations of Computer Science **13**(4) (2002) 571–586
12. Klarlund, N.: A theory of restrictions for logics and automata. In: Proc. of CAV'99. Volume 1633 of LNCS., Springer (1999) 406–417
13. Topnik, C., Wilhelm, E., Margaria, T., Steffen, B.: jMosel: A stand-alone tool and jABC plugin for M2L(Str). In Valmari, A., ed.: 13th International SPIN Workshop. Volume 3925 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2006) 293–298
14. Margaria, T., Steffen, B., Topnik, C.: Second-order value numbering. In: Proc. of GraMoT 2010. Volume 30 of ECEASST., EASST (2010) 1–15
15. Doyen, L., Raskin, J.F.: Antichain algorithms for finite automata. In: Proc. of TACAS'10. Volume 6015 of LNCS., Springer (2010) 2–22
16. Abdulla, P.A., Chen, Y.F., Holík, L., Mayr, R., Vojnar, T.: When simulation meets antichains (on checking language inclusion of NFAs). In: Proc. of TACAS'10. Volume 6015 of LNCS., Springer (2010) 158–174

17. Fiedor, T., Holík, L., Lengál, O., Vojnar, T.: Nested antichains for WS1S. In: Proc. of TACAS'15. Volume 9035 of LNCS., Springer (2015)
18. Ganzow, T., Kaiser, L.: New algorithm for weak monadic second-order logic on inductive structures. In: CSL 2010. Volume 6247 of Lecture Notes in Computer Science., Springer (2010) 366–380
19. Traytel, D.: A coalgebraic decision procedure for WS1S. In Kreutzer, S., ed.: 24th EACSL Annual Conference on Computer Science Logic (CSL 2015). Volume 41 of Leibniz International Proceedings in Informatics (LIPIcs)., Dagstuhl, Germany, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2015) 487–503
20. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications. (2008)
21. Madhusudan, P., Parlato, G., Qiu, X.: Strand benchmark. `http://web.engr.illinois.edu/~qiu2/strand/` Accessed: 2014-01-29.

A Results of Experiments

The times in the tables are given in seconds. We use ∞ to denote that the running time of the tool exceeded 10 minutes and `oom` to denote that the tool ran out of memory.

Table 2. Results of the experiments for the `Strand1` and `Strand2` benchmark sets

Benchmark	MONA		DWINA		GASTON	
	Time	**Space**	**Time**	**Space**	**Time**	**Space**
`Strand1`						
`bs-loop-else`	94.73	168909			0.69	1056
`bs-loop-if-else`	358.81	390593			1.63	1059
`bs-loop-if-if`	398.63	442138			1.97	2084
`sl-insert-after-loop`	∞	∞			1.41	1108
`sl-insert-before-head`	61.56	106858			8.89	259
`sl-insert-before-loop`	13.38	36620			0.33	258
`sl-insert-error-error`	11.81	32238			0.21	330
`sl-insert-in-loop`	159.31	233264			0.72	543
`sl-reverse-after-loop`	8.11	22327			0.05	175
`sl-reverse-before-loop`	5.11	13564			0.05	79
`sl-reverse-in-loop`	193.04	205011			1.33	724
`sl-search-after-loop`	5.49	17656			0.06	143
`sl-search-before-loop`	10.73	31778			0.29	737
`sl-search-in-loop`	21.70	55886			0.74	155
`Strand2`						
`bs-loop-else`	1.13	15170	0.01	19	0.19	454
`bs-loop-if-else`	9.49	63444	0.23	234	0.68	531
`bs-loop-if-if`	31.17	130555	1.14	28	2.64	604
`sl-insert-after-loop`	0.12	2584	0.01	36	0.01	183
`sl-insert-before-head`	0.02	645	0.01	45	0.01	88
`sl-insert-before-loop`	0.06	1415	0.01	47	∞	∞
`sl-insert-error-error`	0.05	1415	0.01	47	∞	∞
`sl-insert-in-loop`	0.32	5963	0.01	59	0.01	273
`sl-reverse-after-loop`	0.09	1886	0.01	110	0.01	138
`sl-reverse-before-loop`	0.09	1886	0.01	1782	0.01	138
`sl-reverse-in-loop`	2.28	24210	0.02	271	∞	∞
`sl-search-after-loop`	0.03	1012	0.01	274	0.01	95
`sl-search-before-loop`	0.04	1119	0.01	274	∞	∞
`sl-search-in-loop`	2.28	24210	0.02	84	∞	∞

Table 3. Results of the experiments for the `Antichains` benchmark set

Benchmark	MONA		DWINA		GASTON	
	Time	**Space**	**Time**	**Space**	**Time**	**Space**
`Antichains`						
`horn-trans05`	0.04	204	0.01	73	0.01	10
`horn-trans06`	0.22	330	0.01	135	0.01	10
`horn-trans07`	0.89	516	0.01	227	0.01	10
`horn-trans08`	3.11	774	0.01	355	0.01	10
`horn-trans09`	10.57	1116	0.01	525	0.01	10
`horn-trans10`	31.60	1554	0.01	743	0.01	10
`horn-trans11`	81.81	2100	0.01	1015	0.01	10
`horn-trans12`	189.14	2766	0.01	1347	0.09	10
`horn-trans13`	409.56	3564	0.02	1745	0.24	10
`horn-trans14`	∞	∞	0.04	2215	0.88	10
`horn-trans15`	∞	∞	0.06	2763	3.39	10
`horn-trans16`	∞	∞	0.10	3395	13.37	10
`horn-trans17`	∞	∞	0.15	4117	53.08	10
`horn-trans18`	∞	∞	0.23	4935	212.68	10
`horn-trans19`	∞	∞	0.32	5855	∞	∞
`set-closed01`	0.01	104	0.01	26	0.01	67
`set-closed02`	0.01	195	0.01	118	0.01	478
`set-closed03`	0.08	1376	0.01	1488	0.03	7127
`set-closed04`	153.09	52697	∞	∞	4.00	278696
`set-closed05`	∞	∞	∞	∞	∞	∞
`set-singletons01`	0.01	57	0.01	26	0.01	3
`set-singletons02`	0.01	108	0.01	53	0.01	3
`set-singletons03`	0.01	443	0.01	92	0.01	3
`set-singletons04`	268.19	66024	0.03	167	0.01	3
`set-singletons05`	∞	∞	0.15	350	0.01	3
`set-singletons06`	∞	∞	0.82	857	0.01	3
`set-singletons07`	∞	∞	∞	∞	0.01	3
`set-singletons08`	∞	∞	∞	∞	0.01	3
`set-singletons09`	∞	∞	∞	∞	0.01	3
`set-singletons10`	∞	∞	∞	∞	0.05	3
`set-singletons11`	∞	∞	∞	∞	0.17	3
`set-singletons12`	∞	∞	∞	∞	∞	∞