# 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 a symbolic, term-based representation of the automaton, and prune the constructed state space from parts irrelevant to 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 in many cases significantly outperform the classical decision procedure (implemented in the MONA tool) as well as recently proposed alternatives.

## 1 Introduction

Weak monadic second-order logic of one successor (WS1S) is a powerful language 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 classical 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 builds an automaton $A_\varphi$ accepting all models of the given formula $\varphi$ in a form of finite words, and then tests $A_\varphi$ for language emptiness. The bottleneck of the procedure is the size of $A_\varphi$, which can be huge due to the fact that the derivation of $A_\varphi$ 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 the classical *explicit* construction by representing automata *symbolically* and testing the emptiness *on the fly*, while constructing $A_\varphi$, and by omitting the state space irrelevant to the emptiness test. This is done using two main principles: *lazy evaluation* and *subsumption-based pruning*. These principles have, to some degree, 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 utilize their power even more.

*Overview of our algorithm.* Our algorithm originates in the classical WS1S decision procedure as implemented in MONA, in which models of formulae are encoded by finite words over a multi-track binary alphabet where each track corresponds to a variable of $\varphi$. In order to come closer to this view of formula models as words, we replace the input formula $\varphi$ by a *language term* $t_\varphi$ describing the language $L_\varphi$ of all word encodings of its models.

In $t_\varphi$, the atomic formulae of $\varphi$ 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 $\pi_X$ removes information about valuations of the quantified variable $X$ from symbols of the alphabet. After the projection, the resulting language $L$ may, however, encode some but not necessarily *all* encodings of the models. In particular, encodings with some specific numbers of trailing $\bar{0}$'s, used as a padding, may be missing. $\bar{0}$ here denotes the symbol with 0 in each track. To obtain a language containing *all* encodings of the models, $L$ must be extended to include encodings with any number of trailing $\bar{0}$'s. This corresponds to taking the (right) $\bar{0}^*$-quotient of $L$, written $L - \bar{0}^*$, which is the set of all prefixes of words of $L$ with the remaining suffix in $\bar{0}^*$. We give an example WS1S formula $\varphi$ in (1) and its language term $t_\varphi$ in (2). The dotted operators represent operators over language terms. See Fig. 2 for the automata $\mathcal{A}_{\mathrm{Sing}(X)}$ and $\mathcal{A}_{Y=X+1}$.

$$\varphi \equiv \exists X\colon \mathrm{Sing}(X) \wedge (\exists Y\colon Y = X + 1) \tag{1}$$

$$t_\varphi \equiv \pi_X(\{\mathcal{A}_{\mathrm{Sing}(X)} \mathbin{\dot{\cap}} (\pi_Y(\mathcal{A}_{Y=X+1}) \mathbin{\dot{-}} \bar{0}^*)\}) \mathbin{\dot{-}} \bar{0}^* \tag{2}$$

The main novelty of our work is that we test emptiness of $L_\varphi$ directly over $t_\varphi$. The term is used as a symbolic representation of the automata that would be explicitly constructed in the classical procedure: inductively to the terms structure, starting from the leaves and combining the automata of sub-terms by standard automata constructions that implement the term operators. Instead of first building automata and only then testing emptiness, we test it on the fly during the construction. This offers opportunities to prune out large portions of the state space that turn out not to be relevant for the test.

A sub-term $t_\psi$ of $t_\varphi$, corresponding to a sub-formula $\psi$, represents final states of the automaton $\mathcal{A}_\psi$ accepting the language encoding models of $\psi$. Predecessors of the final states represented by $t_\psi$ correspond to quotients of $t_\psi$. All states of $\mathcal{A}_\psi$ could hence be constructed by quotienting $t_\psi$ until fixpoint. By working with terms, 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 nonempty, we do not need to process $t_2$. Testing language emptiness of terms arising from quantified sub-formulae is more complicated since they translate to $-\bar{0}^*$ quotients. We evaluate the test on $t - \bar{0}^*$ by iterating the $-\bar{0}$ quotient from $t$. We either conclude with the positive result as soon as one of the iteration computes a term with a nonempty language, or with the negative one if the fixpoint of the quotient construction is reached. The fixpoint condition is that the so-far computed quotients *subsume* the newly constructed ones, where subsumption is a relation under-approximating inclusion of languages represented by terms. Subsumption is also used to prune the set of computed terms so that only an *antichain* of the terms maximal wrt subsumption is kept.

Besides lazy evaluation and subsumption, our approach can benefit from multiple further optimizations. For example, it can be *combined* with the *explicit WS1S decision procedure*, which can be used to transform arbitrary sub-terms of $t_\varphi$ to automata. These automata can then be rather small due to minimization, which cannot be applied in the on-the-fly approach (the automata can, however, also explode due to determinisation and product construction, hence this technique comes with a trade-off). We also propose a novel way of *utilising BDD-based encoding* of automata transition functions in the MONA style for computing quotients of terms. 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 of fixpoint computations.

*Experiments.* We have implemented our decision procedure in a prototype tool called GASTON and compared its performance with other publicly available WS1S solvers on benchmarks from various sources. In the experiments, GASTON managed to win over all other solvers on various parametric families of WS1S formulae that were designed—mostly by authors of other tools—to stress-test WS1S solvers. Moreover, GASTON was able to significantly outperform MONA and other solvers on a number of formulae obtained from various formal verification tasks. This shows that our approach is applicable in practice and has a great potential to handle more complex formulae than those so far obtained in WS1S applications. We believe that the efficiency of our approach can be pushed much further, making WS1S scale enough for 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 explicit 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), or the procedure using symbolic finite automata of D'Antoni *et al.* in [15].

Our work was originally inspired by antichain techniques for checking universality and inclusion of finite automata [16,10,17], which use symbolic computation and subsumption to prune large state spaces arising from subset construction. In [18], which is a starting point for the current paper, we discussed a basic idea of generalizing these techniques to a WS1S decision procedure. In the current paper we have turned the idea of [18] to an algorithm efficient in practice by roughly the following steps: (1) reformulating the symbolic representation of automata from nested upward and downward closed sets of automata states to more intuitive language terms, (2) generalizing the procedure originally restricted to formulae in the prenex normal form to arbitrary formulae, (3) introduction of lazy evaluation, and (4) many other important optimizations.

Recently, a couple of logic-based approaches for deciding WS1S appeared. Ganzow and Kaiser [19] developed a new decision procedure for the weak monadic second-order logic on inductive structures, within their tool TOSS, which is even more general than WS$k$S. Their approach completely avoids automata; instead, it is based on Shelah's composition method. The TOSS tool is quite promising as it outperforms MONA on some of the benchmarks. It, however, lacks some features in order to perform meaningful comparison on benchmarks used in practice. Traytel [20], on the other hand, uses the classical decision procedure, recast in the framework of coalgebras. The work focuses on testing equivalence of a pair of formulae, which is performed by finding a bisimulation between derivatives of the formulae. While it is shown that it can outperform MONA on some simple artificial examples, the implementation is not optimized enough and is easily outperformed by the rest of the tools on other benchmarks.

## 2 Preliminaries on Languages and Automata

A *word* over a finite alphabet $\Sigma$ is a finite sequence $w = a_1 \cdots a_n$, for $n \geq 0$, of symbols from $\Sigma$. Its $i$-th symbol $a_i$ is denoted by $w[i]$. For $n = 0$, the word is the empty word $\epsilon$. A language $L$ is a set of words over $\Sigma$. 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 $\Sigma$ 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 $\mathcal{L}(q)$ accepted *at* 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 $\delta$ contains transitions $(q_0, a_1, q_1), \ldots, (q_{n-1}, a_n, q_n)$ with $q_0 \in I$ and $q_n = q$. 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.

## 3 WS1S

In this section, we give a minimalistic introduction to the *weak monadic second-order logic of one successor* (WS1S) and outline its explicit decision procedure based on representing sets of models as regular languages and finite automata. See, for instance, Comon *et al.* [21] for a more thorough introduction.

### 3.1 Syntax and Semantics of WS1S

WS1S allows quantification over second-order *variables*, which we denote by uppercase 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 $\exists \mathcal{X}$ where $\mathcal{X}$ is a finite set of variables (we write $\exists X$ if $\mathcal{X}$ is a singleton $\{X\}$). A *model* of a WS1S formula $\varphi(\mathcal{X})$ with the set of free variables $\mathcal{X}$ is an assignment $\rho : \mathcal{X} \to 2^{\mathbb{N}_0}$ of the free variables $\mathcal{X}$ of $\varphi$ 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 $\rho(X) = \{x\}, \rho(Y) = \{y\}$, and $x = y + 1$. Satisfaction for formulae obtained using Boolean connectives is defined as usual. A formula $\varphi$ is *valid*, written $\models \varphi$, iff all assignments of its free variables to finite subsets of $\mathbb{N}_0$ are its models, and *satisfiable* if it has a model. Wlog we assume that each variable in a formula is quantified at most once.

### 3.2 Models as Words

Let $\mathcal{X}$ be a finite set of variables. A *symbol* $\tau$ over $\mathcal{X}$ is a mapping of all variables in $\mathcal{X}$ to the set $\{0, 1\}$, e.g. $\tau = \{X_1 \mapsto 0, X_2 \mapsto 1\}$ for $\mathcal{X} = \{X_1, X_2\}$, which we will write as $\tau = \begin{smallmatrix} X_1 : 0 \\ X_2 : 1 \end{smallmatrix}$ below. The set of all symbols over $\mathcal{X}$ is denoted as $\Sigma_{\mathcal{X}}$. We use $\bar{0}$ to denote the symbol in $\Sigma_{\mathcal{X}}$ that maps all variables to 0, i.e. $\bar{0} = \{X \mapsto 0 \mid X \in \mathcal{X}\}$.

An assignment $\rho : \mathcal{X} \to 2^{\mathbb{N}_0}$ may be encoded as a word $w_\rho$ of symbols over $\mathcal{X}$ in the following way: $w_\rho$ contains 1 in the $(i+1)$-st position of the row for $X$ iff $i \in X$ in $\rho$. Notice that there exists an infinite number of encodings of $\rho$: the shortest encoding is $w^s_\rho$ of the length $n+1$, where $n$ is the largest number appearing in any of the sets that is assigned to a variable of $\mathcal{X}$ in $\rho$, or $-1$ when all these sets are empty. The rest of the encodings are all those corresponding to $w^s_\rho$ 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 $\mathcal{L}(\varphi) \subseteq \Sigma^*_{\mathcal{X}}$ to denote the language of all encodings of a formula $\varphi$'s models, where $\mathcal{X}$ are the free variables of $\varphi$.

For two sets $\mathcal{X}$ and $\mathcal{Y}$ of variables and any two symbols $\tau_1, \tau_2 \in \Sigma_{\mathcal{X}}$, we write $\tau_1 \sim_{\mathcal{Y}} \tau_2$ iff $\forall X \in \mathcal{X} \setminus \mathcal{Y} : \tau_1(X) = \tau_2(X)$, i.e. the two symbols differ (at most) in the values of variables in $\mathcal{Y}$. The relation $\sim_{\mathcal{Y}}$ is generalized to words such that $w_1 \sim_{\mathcal{Y}} w_2$ iff $|w_1| = |w_2|$ and $\forall 1 \leq i \leq |w_1| : w_1[i] \sim_{\mathcal{Y}} w_2[i]$. For a language $L \subseteq \Sigma^*_{\mathcal{X}}$, we define $\pi_{\mathcal{Y}}(L)$ as the language of words $w$ that are $\sim_{\mathcal{Y}}$-equivalent with some word $w' \in L$. Seen from the point of view of encodings of sets of assignments, $\pi_{\mathcal{Y}}(L)$ encodes all assignments that may differ from those encoded by $L$ (only) in the values of variables from $\mathcal{Y}$. If $\mathcal{Y}$ is disjoint with the free variables of $\varphi$, then $\pi_{\mathcal{Y}}(\mathcal{L}(\varphi))$ corresponds to the so-called *cylindrification* of $\mathcal{L}(\varphi)$, and if it is their subset, then $\pi_{\mathcal{Y}}(\mathcal{L}(\varphi))$ corresponds to the so-called *projection* [21]. We use $\pi_Y$ to denote $\pi_{\{Y\}}$ for a variable $Y$.

$$\mathcal{L}^{\mathbb{V}}(\varphi \vee \psi) = \mathcal{L}^{\mathbb{V}}(\varphi) \cup \mathcal{L}^{\mathbb{V}}(\psi) \quad (3)$$

$$\mathcal{L}^{\mathbb{V}}(\varphi \wedge \psi) = \mathcal{L}^{\mathbb{V}}(\varphi) \cap \mathcal{L}^{\mathbb{V}}(\psi) \quad (4)$$

$$\mathcal{L}^{\mathbb{V}}(\neg\varphi) = \Sigma^*_{\mathbb{V}} \setminus \mathcal{L}^{\mathbb{V}}(\varphi) \quad (5)$$

$$\mathcal{L}^{\mathbb{V}}(\exists \mathcal{X} : \varphi) = \pi_{\mathcal{X}}(\mathcal{L}^{\mathbb{V}}(\varphi)) - \bar{0}^* \quad (6)$$

Consider formulae over the set $\mathbb{V}$ of variables. Let *free*$(\varphi)$ be the set of free variables of $\varphi$, and let $\mathcal{L}^{\mathbb{V}}(\varphi) = \pi_{\mathbb{V} \setminus \mathit{free}(\varphi)}(\mathcal{L}(\varphi))$ be the language $\mathcal{L}(\varphi)$ cylindrified wrt those variables of $\mathbb{V}$ that are not free in $\varphi$. Let $\varphi$ and $\psi$ be formulae and assume that $\mathcal{L}^{\mathbb{V}}(\varphi)$ and $\mathcal{L}^{\mathbb{V}}(\psi)$ are languages of encodings of their models cylindrified wrt $\mathbb{V}$. Languages of formulae obtained from $\varphi$ and $\psi$ using logical connectives are defined by equations (3) to (6). Equations (3)-(5) above are straightforward: Boolean connectives translate to the corresponding set operators over the universe of encodings of assignments of variables in $\mathbb{V}$. Existential quantification $\exists \mathcal{X} : \varphi$ translates into a composition of two language transformations. First, $\pi_{\mathcal{X}}$ makes the valuations of variables of $\mathcal{X}$ arbitrary, which intuitively corresponds to forgetting everything about values of variables in $\mathcal{X}$ (notice that this is a different use of $\pi_{\mathcal{X}}$ than the cylindrification since here variables of $\mathcal{X}$ *are* free variables of $\varphi$). The second step, removing suffixes of $\bar{0}$'s from the model encodings, is necessary since $\pi_{\mathcal{X}}(\mathcal{L}^{\mathbb{V}}(\varphi))$ might be missing some encodings of models of $\exists \mathcal{X} : \varphi$. For example, suppose that $\mathbb{V} = \{X, Y\}$ and the only model of $\varphi$ is $\{X \mapsto \{0\}, Y \mapsto \{1\}\}$, yielding $\mathcal{L}^{\mathbb{V}}(\varphi) = \begin{smallmatrix} X : 10 \\ Y : 01 \end{smallmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix}^*$. Then $\pi_Y(\mathcal{L}^{\mathbb{V}}(\varphi)) = \begin{smallmatrix} X : 10 \\ Y : ?? \end{smallmatrix}\begin{bmatrix} 0 \\ ? \end{bmatrix}^*$ 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 $\varphi$ where the largest number in the value of the variable $Y$ being eliminated is larger than maximum number found in the values of the free variables of $\exists Y : \varphi$. The role of the $-\bar{0}^*$ quotient 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 $\varphi$ with the set of variables $\mathbb{V}$ is to construct an automaton $\mathcal{A}_\varphi$ accepting $\mathcal{L}^{\mathbb{V}}(\varphi)$ and check emptiness of its language. The construction starts with simple pre-defined automata $\mathcal{A}_\psi$ for $\varphi$'s atomic formulae $\psi$ (see Fig. 2 for examples of automata for selected atomic formulae and e.g. [21] for more details) accepting cylindrified languages $\mathcal{L}^{\mathbb{V}}(\psi)$ of models of $\psi$. These are simple regular languages. The construction then continues by inductively constructing automata $\mathcal{A}_{\varphi'}$ accepting languages $\mathcal{L}^{\mathbb{V}}(\varphi')$ of models for all other subformulae $\varphi'$ of $\varphi$, using equations (3)–(6) above. The language operators used in the rules are implemented using standard automata-theoretic constructions (see [21]).

## 4 Satisfiability via Language Term Evaluation

This section introduces the basic version of our symbolic algorithm for deciding satisfiability of a WS1S formula $\varphi$ with a set of variables $\mathbb{V}$. Its optimized version is the subject of the next section. To simplify presentation, we 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 $\varphi$ is ground, the language $\mathcal{L}^{\mathbb{V}}(\varphi)$ is either $\Sigma_{\mathbb{V}}^*$ in the case $\varphi$ is valid, or empty if $\varphi$ is invalid. Then, to decide the validity of $\varphi$, it suffices to test if $\epsilon \in \mathcal{L}^{\mathbb{V}}(\varphi)$.

Our algorithm evaluates the so-called *language term* $t_\varphi$, a symbolic representation of the language $\mathcal{L}^{\mathbb{V}}(\varphi)$, whose structure reflects the construction of $\mathcal{A}_\varphi$. It is a (finite) term generated by the following grammar:

$$t ::= \mathcal{A} \mid t \,\dot{\cup}\, t \mid t \,\dot{\cap}\, t \mid \bar{t} \mid \pi_{\mathcal{X}}(t) \mid t \,\dot{-}\, \alpha \mid t \,\dot{-}\, \alpha^* \mid T$$

where $\mathcal{A}$ is a finite automaton over the alphabet $\Sigma_{\mathbb{V}}$, $\alpha$ is a symbol $\tau \in \Sigma_{\mathbb{V}}$ or a set $S \subseteq \Sigma_{\mathbb{V}}$ of symbols, and $T$ is a finite set of terms. We use marked variants of the operators to distinguish the syntax of language terms manipulated by our algorithm from the cases when we wish to denote the semantical meaning of the operators. A term of the form $t \,\dot{-}\, \alpha^*$ is called a *star quotient*, or shortly a *star*, and a term $t \,\dot{-}\, \tau$ is a *symbol quotient*. Both are also called *quotients*. The *language* $\mathcal{L}(t)$ *of a term* $t$ is obtained by taking the languages of the automata in its leaves and combining them using the term operators. Terms with the same language are *language-equivalent*. The special terms $T$, having the form of a set, represent intermediate states of fixpoint computations used to eliminate star quotients.The language of a set $T$ equals the *union* of the languages of its elements. The reason for having two ways of expressing a union of terms is a different treatment of $\dot{\cup}$ and $T$, which will be discussed later. We use the standard notion of isomorphism of two terms, extended with having two set terms isomorphic iff they contain isomorphic elements.

A formula $\varphi$ is initially transformed into the term $t_\varphi$ by replacing every atomic subformula $\psi$ in $\varphi$ by the automaton $\mathcal{A}_\psi$ accepting $\mathcal{L}^{\mathbb{V}}(\psi)$, and by replacing the logical connectives with dotted term operators according to equations (3)–(6) of Section 3.2. The core of our algorithm is evaluation of the $\epsilon$-membership query $\epsilon \in t_\varphi$, which will also trigger further rewriting of the term.

The $\epsilon$-membership query on a quotient-free term is evaluated using equivalences (7) to (12). Equivalences (7) to (11) reduce tests on terms to Boolean combinations of tests on their sub-terms and allow pushing the test towards the automata at the term's leaves. Equivalence (12) then reduces it to testing intersection of the initial states $I(\mathcal{A})$ and the final states $F(\mathcal{A})$ of an automaton.

$$
\begin{array}{rll}
\epsilon \in T & \text{iff} & \epsilon \in t \text{ for some } t \in T \qquad (7)\\
\epsilon \in t \Cupdot t' & \text{iff} & \epsilon \in t \text{ or } \epsilon \in t' \qquad (8)\\
\epsilon \in t \Capdot t' & \text{iff} & \epsilon \in t \text{ and } \epsilon \in t' \qquad (9)\\
\epsilon \in \overline{t} & \text{iff} & \text{not } \epsilon \in t \qquad (10)\\
\epsilon \in \pi_{\mathcal{X}}(t) & \text{iff} & \epsilon \in t \qquad (11)\\
\epsilon \in \mathcal{A} & \text{iff} & I(\mathcal{A}) \cap F(\mathcal{A}) \neq \emptyset \qquad (12)
\end{array}
$$

Equivalences (7) to (11) do not apply to quotients, which arise from quantified subformulae (cf. equation (6) in Section 3.2). A quotient is therefore (in the basic version) first rewritten into a language-equivalent quotient-free form. This rewriting corresponds to saturating the set of final states of an automaton in the explicit decision procedure with all states in their $pre^*$-image over $\bar{0}$. In our procedure, we use rules (13) and (14).

Rule (13) transforms the term into a form in which a star quotient is applied on a plain set of terms rather than on a projection. A star quotient of a set is then eliminated using a fixpoint computation that saturates the set with all quotients of its elements wrt the set of symbols $S = \pi_{\mathcal{X}}(\bar{0})$. A single iteration is implemented using rule (14). There, $T \ominus S$ is the set $\{t \mathbin{\dot{-}} \tau \mid t \in T \wedge \tau \in S\}$ of quotients of terms in $T$ wrt symbols of $S$. (Note that (14) uses the identity $S^* = \{\epsilon\} \cup S^* S$.) Termination of the fixpoint computation is decided based on the subsumption relation $\sqsubseteq$, which is some relation that under-approximates language inclusion of terms. When the condition holds, then the language of $T$ is stable wrt quotienting by $S$, i.e. $\mathcal{L}(T) = \mathcal{L}(T \mathbin{\dot{-}} S^*)$. In the basic algorithm, we use term isomorphism for $\sqsubseteq$; later, we provide a more precise subsumption relation with a good trade-off between precision and cost. Note that an iteration of rule (14) can be implemented efficiently by the standard worklist algorithm, which extends $T$ only with quotients $T' \ominus S$ of terms $T'$ that were added to $T$ in the previous iteration.

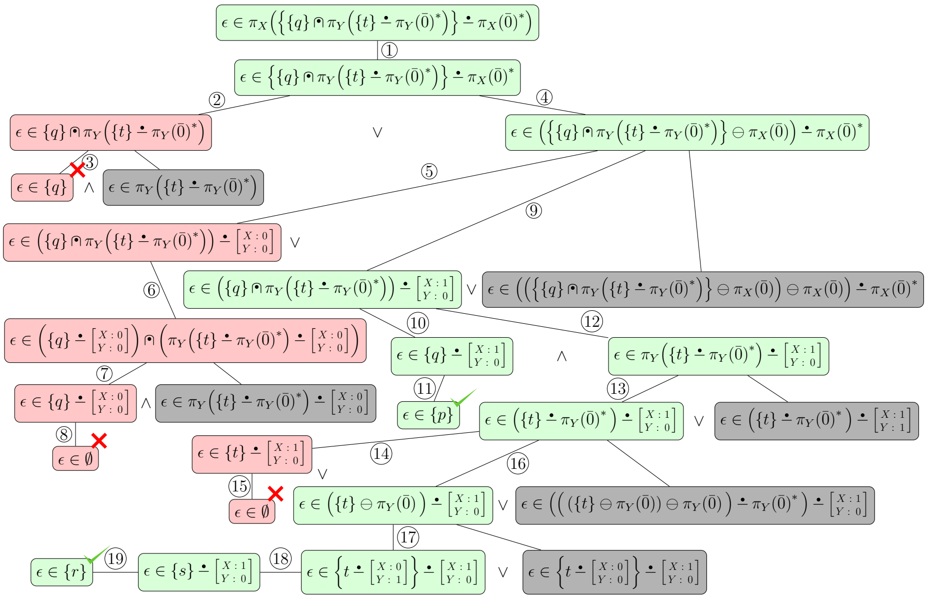
$$\pi_{\mathcal{X}}(T) \mathbin{\dot{-}} \bar{0}^* \;\rightarrow\; \pi_{\mathcal{X}}(T \mathbin{\dot{-}} \pi_{\mathcal{X}}(\bar{0})^*) \qquad (13)$$

$$T \mathbin{\dot{-}} S^* \rightarrow \begin{cases} T & \text{if } T \ominus S \sqsubseteq T \\ (T \cup (T \ominus S)) \mathbin{\dot{-}} S^* & \text{otherwise} \end{cases} \qquad (14)$$

The set $T \ominus S$ introduces quotient terms of the form $t \mathbin{\dot{-}} \tau$, for $\tau \in \Sigma_{\mathbb{V}}$, which also need to be eliminated to facilitate the $\epsilon$-membership test. This is done using rewriting rules (15) to (19), where $pre_{[\tau]}(\mathcal{A})$ is $\mathcal{A}$ with its set of final states $F$ replaced by $pre_{[\tau]}(F)$.

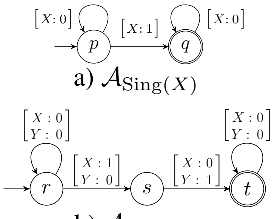
$$
\begin{array}{rcl}
(t \Cupdot t') \mathbin{\dot{-}} \tau & \rightarrow & (t \mathbin{\dot{-}} \tau) \Cupdot (t' \mathbin{\dot{-}} \tau) \qquad (15)\\
(t \Capdot t') \mathbin{\dot{-}} \tau & \rightarrow & (t \mathbin{\dot{-}} \tau) \Capdot (t' \mathbin{\dot{-}} \tau) \qquad (16)\\
\overline{t} \mathbin{\dot{-}} \tau & \rightarrow & \overline{t \mathbin{\dot{-}} \tau} \qquad (17)\\
\pi_{\mathcal{X}}(t) \mathbin{\dot{-}} \tau & \rightarrow & \pi_{\mathcal{X}}(t \mathbin{\dot{-}} \pi_{\mathcal{X}}(\tau)) \qquad (18)\\
\mathcal{A} \mathbin{\dot{-}} \tau & \rightarrow & pre_{[\tau]}(\mathcal{A}) \qquad (19)
\end{array}
$$

If $t$ is quotient-free, then rules (15)–(18) applied to $t \mathbin{\dot{-}} \tau$ push the symbol quotient down the structure of $t$ towards the automata in the leaves, where it is eliminated by rule (19). Otherwise, if $t$ is not quotient-free, it can be re-written using rules (13)–(19). In particular, if $t$ is a star quotient of a quotient-free term, then the quotient-free form of $t$ can be obtained by iterating rule (14), combined with rules (15)–(19) to transform the new terms in $T$ into a quotient-free form. Finally, terms with multiple quotients can be rewritten to the quotient-free form inductively to their structure. Every inductive step rewrites some star quotient of a quotient-free sub-term into the quotient-free form. Note that this procedure is bound to terminate since the terms generated by quotienting a star have the same structure as the original term, differing only in the states in their leaves. As the number of the states is finite, so is the number of the terms.



$\epsilon \in \pi_X\Big(\big\{\{q\} \capdot \pi_Y\big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big)\big\} \stackrel{\bullet}{-} \pi_X(\bar{0})^*\Big)$
①
$\epsilon \in \big\{\{q\} \capdot \pi_Y\big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big)\big\} \stackrel{\bullet}{-} \pi_X(\bar{0})^*$
②
④
$\epsilon \in \{q\} \capdot \pi_Y\big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big)$
$\vee$
$\epsilon \in \Big(\big\{\{q\} \capdot \pi_Y\big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big)\big\} \ominus \pi_X(\bar{0})\Big) \stackrel{\bullet}{-} \pi_X(\bar{0})^*$
③
$\epsilon \in \{q\}$
$\wedge$
$\epsilon \in \pi_Y\big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big)$
⑤
⑨
$\epsilon \in \Big(\{q\} \capdot \pi_Y\big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big)\Big) \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\right]$ $\vee$
⑥
$\epsilon \in \Big(\{q\} \capdot \pi_Y\big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big)\Big) \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$ $\vee$
$\epsilon \in \Big(\Big(\big\{\{q\} \capdot \pi_Y\big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big)\big\} \ominus \pi_X(\bar{0})\Big) \ominus \pi_X(\bar{0})\Big) \stackrel{\bullet}{-} \pi_X(\bar{0})^*$
⑩
⑫
$\epsilon \in \Big(\{q\} \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\right]\Big) \capdot \Big(\pi_Y\big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big) \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\right]\Big)$
$\epsilon \in \{q\} \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$
$\wedge$
$\epsilon \in \pi_Y\big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big) \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$
⑦
⑪
⑬
$\epsilon \in \{q\} \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\right]$
$\wedge$
$\epsilon \in \pi_Y\big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big) \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\right]$
$\epsilon \in \{p\}$
$\epsilon \in \big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big) \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$
$\vee$
$\epsilon \in \big(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\big) \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:1\\Y:1\end{smallmatrix}\right]$
⑧
$\epsilon \in \emptyset$
$\epsilon \in \{t\} \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$
⑭
⑯
$\vee$
⑮
$\epsilon \in \emptyset$
$\epsilon \in \big(\{t\} \ominus \pi_Y(\bar{0})\big) \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$
$\vee$
$\epsilon \in \Big(\big((\{t\} \ominus \pi_Y(\bar{0})) \ominus \pi_Y(\bar{0})\big) \stackrel{\bullet}{-} \pi_Y(\bar{0})^*\Big) \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$
⑰
$\epsilon \in \{r\}$
⑲
$\epsilon \in \{s\} \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$
⑱
$\epsilon \in \Big\{t \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:0\\Y:1\end{smallmatrix}\right]\Big\} \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$
$\vee$
$\epsilon \in \Big\{t \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\right]\Big\} \stackrel{\bullet}{-} \left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$

**Fig. 1.** Example of deciding validity of the formula $\varphi \equiv \exists X : \mathrm{Sing}(X) \wedge (\exists Y : Y = X + 1)$



$[X{:}0]$
$p$
$[X{:}1]$
$q$
$[X{:}0]$
$\left[\begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\right]$
$\left[\begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\right]$
$r$
$\left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$
$s$
$\left[\begin{smallmatrix}X:0\\Y:1\end{smallmatrix}\right]$
$t$

a) $\mathcal{A}_{\mathrm{Sing}(X)}$

b) $\mathcal{A}_{Y=X+1}$

**Fig. 2.** Example automata

*Example 1.* We will show the workings of our procedure using an example of testing satisfiability of the formula $\varphi \equiv \exists X.\, \mathrm{Sing}(X) \wedge (\exists Y.\, Y = X+1)$. We start by rewriting $\varphi$ into a *term* $t_\varphi$ representing its language $\mathcal{L}^{\mathbb{V}}(\varphi)$:

$$t_\varphi \equiv \pi_X(\{\{q\} \capdot \pi_Y(\{t\} \stackrel{\bullet}{-} \pi_Y(\bar{0})^*)\} \stackrel{\bullet}{-} \pi_X(\bar{0})^*)$$

(we have already used rule (13) twice). In the example, a set $R$ of states will denote an automaton obtained from $\mathcal{A}_{\mathrm{Sing}(X)}$ or $\mathcal{A}_{Y=X+1}$ (cf. Fig. 2) by setting the final states to $R$. Red nodes in the computation tree denote $\epsilon$-membership tests that failed and green nodes those that succeeded. Grey nodes denote tests that were not evaluated.

As noted previously, it holds that $\models \varphi$ iff $\epsilon \in t_\varphi$. The sequence of computation steps for determining the $\epsilon$-membership test is shown using the computation tree in Fig. 1. The nodes contain $\epsilon$-membership tests on terms and the test of each node is equivalent to a conjunction or disjunction of tests of its children. Leafs of the form $\epsilon \in R$ are evaluated as testing intersection of $R$ with the initial states of the corresponding automaton. In the example, we also use the *lazy evaluation* technique (described in Section 5.2), which allows us to evaluate $\epsilon$-membership tests on partially computed fixpoints.

The computation starts at the root of the tree and proceeds along the edges in the order given by their circled labels. Edges ② and ④ were obtained by a partial unfolding of a fixpoint computation by rule (14) and immediately applying $\epsilon$-membership test on the obtained terms. After step ③, we conclude that $\epsilon \notin \{q\}$ since $\{p\} \cap \{q\} = \emptyset$, which further refutes the whole conjunction below ②, so the overall result depends on the sub-tree starting by ④. The steps ⑤ and ⑨ are another application of rule (14), which transforms $\pi_X(\bar{0})$ to the symbols $\left[\begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\right]$ and $\left[\begin{smallmatrix}X:1\\Y:0\end{smallmatrix}\right]$ respectively. The branch ⑤ pushes the $\stackrel{\bullet}{-} \left[\begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\right]$ quotient to the leaf term using rules (16) and (9) and eventually fails because the predecessors of $\{q\}$ over the symbol $\left[\begin{smallmatrix} X:0 \\ Y:0 \end{smallmatrix}\right]$ in $\mathcal{A}_{\mathrm{Sing}(X)}$ is the empty set. On the other hand, the evaluation of the branch $\textcircled{9}$ continues using rule (16), succeeding in the branch $\textcircled{10}$. The branch $\textcircled{12}$ is further evaluated by projecting the quotient $\stackrel{\bullet}{-} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$ wrt $Y$ (rule 18) and unfolding the inner star quotient zero times ($\textcircled{14}$, failed) and once ($\textcircled{16}$). The unfolding of one symbol eventually succeeds in step $\textcircled{19}$, which leads to concluding validity of $\varphi$. Note that thanks to the lazy evaluation, none of the fixpoint computations had to be fully unfolded. □

# 5 An Efficient Algorithm

In this section, we show how to build an efficient algorithm based on the symbolic term rewriting approach from Section 4. The optimization opportunities offered by the symbolic approach are to a large degree orthogonal to those of the explicit approach. The main difference is in the available techniques for reducing the explored automata state space. While the explicit construction in MONA profits mainly from calling *automata minimization* after every step of the inductive construction, the symbolic algorithm can use generalized *subsumption* and *lazy evaluation*. None of the two approaches seems to be compatible with both these techniques (at least in their pure variant, disregarding the possibility of a combination of the two approaches discussed below).

*Efficient data structures* have a major impact on performance of the decision procedure. The efficiency of the explicit procedure implemented in MONA is to a large degree due to the BDD-based representation of automata transition relations. BDDs compactly represent transition functions over large alphabets and provide efficient implementation of operations needed in the explicit algorithm. Our symbolic algorithm can, on the other hand, benefit from a representation of terms as DAGs where all occurrences of the same sub-term are represented by a unique DAG node. Moreover, we assume the nodes to be associated with languages rather than with concrete terms (allowing the term associated with a node to change during its further processing, without a need to transform the DAG structure as long as the language of the term does not change).

We also show that despite our algorithm uses a completely different data structure than the explicit one, it can still exploit a BDD-based representation of transitions of the automata in the leaves of terms. Moreover, our symbolic algorithm can also be *combined* with the explicit algorithm. Particularly, it turns out that, sometimes, it pays off to translate to automata sub-formulae larger than the atomic ones. Our procedure can then be viewed as an extension of MONA that takes over once MONA stops managing. Lastly, optimizations on the level of formulae often have a huge impact on the performance of our algorithm. The technique that we found most helpful is the so-called *anti-prenexing*. We elaborate on all these optimizations in the rest of this section.

## 5.1 Subsumption

Our first 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 $\sqsubseteq_s$ on terms that is given by equivalences (20)–(25). Notice that, in rule (20), all terms of $T$ are tested against all terms of $T'$, while in rule (21), the left-hand side term $t_1$ is not tested against the right-hand side term $t'_2$ (and similarly for $t_2$ and $t'_1$).

$$T \sqsubseteq_s T' \text{ iff } \forall t \in T\, \exists t' \in T' : t \sqsubseteq_s t' \tag{20}$$

$$t_1 \cupdot t_2 \sqsubseteq_s t_1' \cupdot t_2' \text{ iff } t_1 \sqsubseteq_s t_1' \text{ and } t_2 \sqsubseteq_s t_2' \tag{21}$$

$$t_1 \capdot t_2 \sqsubseteq_s t_1' \capdot t_2' \text{ iff } t_1 \sqsubseteq_s t_1' \text{ and } t_2 \sqsubseteq_s t_2' \tag{22}$$

$$\overline{t} \sqsubseteq_s \overline{t'} \text{ iff } t \sqsupseteq_s t' \tag{23}$$

$$\pi_{\mathcal{X}}(t) \sqsubseteq_s \pi_{\mathcal{X}}(t') \text{ iff } t \sqsubseteq_s t' \tag{24}$$

$$\mathcal{A} \sqsubseteq_s \mathcal{A}' \text{ iff } F(\mathcal{A}) \subseteq F(\mathcal{A}') \tag{25}$$

The reason why $\cupdot$ is order-sensitive is that the terms on different sides of the $\cupdot$ are assumed to be built from automata with disjoint sets of states (originating from different sub-formulae of the original formula), and hence the subsumption test on them can never conclude positively. The subsumption under-approximates language inclusion and can therefore be used for $\sqsubseteq$ in rule (14). It is far more precise than isomorphism and its use leads to an earlier termination of fixpoint computations.

$$T \to T \setminus \{t\} \quad \text{if there is } t' \in T \setminus \{t\} \text{ with } t \sqsubseteq_s t' \tag{26}$$

Moreover, $\sqsubseteq_s$ can be used to prune star quotient terms $T \mathbin{\dot{-}} S^*$ while preserving their language. Since the semantics of the set $T$ is the union of the languages of its elements, then elements subsumed by others can be removed while preserving the language. $T$ can thus be kept in the form of an *antichain* of $\sqsubseteq_s$-incomparable terms. The pruning corresponds to using the rewriting rule (26).

### 5.2 Lazy Evaluation

The top-down nature of our technique allows us to postpone evaluation of some of the computation branches in case the so-far evaluated part is sufficient for determining the result of the evaluated $\epsilon$-membership or subsumption test. We call this optimization *lazy evaluation.* A basic variant of lazy evaluation *short-circuits* elimination of quotients from branches of $\cupdot$ and $\capdot$. When testing whether $\epsilon \in t \cupdot t'$ (rule (8)), we first evaluate, e.g., the test $\epsilon \in t$, and when it holds, we can completely avoid exploring $t'$ and evaluating quotients there. When testing $\epsilon \in t \capdot t'$, we can proceed analogously if one of the two terms is shown not to contain $\epsilon$. Rules (21) and (22) offer similar opportunities for short-circuiting evaluation of subsumption of $\cupdot$ and $\capdot$.

Let us note that subsumption is in a different position than $\epsilon$-membership since correctness of our algorithm depends on the precision of the $\epsilon$-membership test, but subsumption may be evaluated in any way that under-approximates inclusion of languages of terms (and over-approximates isomorphism in order to guarantee termination). Hence, $\epsilon$-membership test must enforce eliminating quotients until it can conclude the result, while there is a choice in the case of the subsumption. If subsumption is tested on quotients, it can either eliminate them, or it can return the (safe) negative answer. However, this choice comes with a trade-off. Subsumption eliminating quotients is more expensive but also more precise. The higher precision allows better pruning of the state space and earlier termination of fixpoint computation, which, according to our empirical experience, pays off.

Lazy evaluation can also reduce the number of iterations of a star. The iterations can be computed *on demand*, only when required by the tests. The idea is to try to conclude a test $\epsilon \in T \mathbin{\dot{-}} S^*$ based on the intermediate state $T$ of the fixpoint computation. This can be done since $\mathcal{L}(T)$ always under-approximates $\mathcal{L}(T \mathbin{\dot{-}} S^*)$, hence if $\epsilon \in \mathcal{L}(T)$, then $\epsilon \in \mathcal{L}(T \mathbin{\dot{-}} S^*)$. Continuing the fixpoint computation is then unnecessary.

The above mechanism alone is, however, rather insufficient in the case of nested stars. Assume that an inner star fixpoint computation was terminated in a state $T \mathbin{\dot{-}} S^*$

when $\epsilon$ was found in $T$ for the first time. Every unfolding of an outer star then propagates $\dot{-}\,\tau$ quotients towards $T \mathbin{\dot{-}} S^*$. We have, however, no way of eliminating it from $(T \mathbin{\dot{-}} S^*) \mathbin{\dot{-}} \tau$ other than finishing the unfolding of $T \mathbin{\dot{-}} S^*$ first (which eliminates the inner star). The need to fully unfold $T \mathbin{\dot{-}} S^*$ would render the earlier lazy evaluation of the $\epsilon$-membership test worthless. To remove this deficiency, we need a way of eliminating the $\dot{-}\,\tau$ quotient from the intermediate state of $T \mathbin{\dot{-}} S^*$.

The elimination is achieved by letting the star quotient $T \mathbin{\dot{-}} S^*$ explicitly "publish" its intermediate state $T$ using rule (27). The symbol $\succcurlyeq$ is read as "*is under-approximated by*." Rules (28)–(30) allow to conclude $\epsilon$-membership and subsumption by testing the under-approximation on its right-hand side (notice the distinction between "if" and the "iff" used in the rules earlier).

$$T \mathbin{\dot{-}} S^* \;\rightarrow\; T \mathbin{\dot{-}} S^* \succcurlyeq T \quad (27)$$

$$\epsilon \in t \succcurlyeq t' \quad \text{if} \quad \epsilon \in t' \quad (28)$$

$$t \succcurlyeq T \not\sqsupseteq_s t' \quad \text{if} \quad T \not\sqsupseteq_s t' \quad (29)$$

$$t \succcurlyeq T \sqsupseteq_s t' \quad \text{if} \quad T \sqsupseteq_s t' \quad (30)$$

Symbol quotients that come from the unfolding of an outer star can be evaluated on the approximation too using rule (31), which then applies the symbol-set quotient on the approximation $T$ of the inner term $t$, and publishes the result on the right-hand side of $\succcurlyeq$. The left-hand side still remembers the original term $t \mathbin{\dot{-}} S$.

$$(t \succcurlyeq T) \mathbin{\dot{-}} S \;\rightarrow\; ((t \succcurlyeq T) \mathbin{\dot{-}} S) \succcurlyeq T \ominus S \quad (31)$$

Terms arising from rules (27) and (31) allow an efficient update in the case an inner term $t$ spawns a new, more precise approximation. In the process, rule (32) is used to remove old outdated approximations.

$$T \succcurlyeq T' \;\rightarrow\; T \quad (32)$$

We will explain the working of the rules and their efficient implementation on an evaluation from Example 1. Note that in Example 1, the partial unfoldings of the fixpoints that are tested for $\epsilon$-membership are under-approximations of a star quotient term. For instance, branch $\textcircled{14}$ corresponds to testing $\epsilon$-membership in the right-most approximation of the term $\Big(((\{t\} \mathbin{\dot{-}} \pi_Y(\bar{0})^*) \succcurlyeq \{t\}) \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]\Big) \succcurlyeq \{t\} \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$ by rule (28) (the branch determines that $\epsilon \notin \{t\} \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$). The result of $\textcircled{14}$ cannot conclude the top-level $\epsilon$-membership test because $\{t\} \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$ is just an under-approximation of $(\{t\} \mathbin{\dot{-}} \pi_Y(\bar{0})^*) \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$. Therefore, we need to compute a better approximation of the term and try to conclude the test on it. We compute it by first applying rule (32) twice to discard obsolete approximations ($\{t\}$ and $\{t\} \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$), followed by applying rule (14) to replace $(\{t\} \mathbin{\dot{-}} \pi_Y(\bar{0})^*) \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$ with $\big((\{t\} \cup (\{t\} \ominus \pi_Y(\bar{0}))) \mathbin{\dot{-}} \pi_Y(\bar{0})^*\big) \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$. Let $\beta = \{t\} \cup (\{t\} \ominus \pi_Y(\bar{0}))$. Then, using rules (27) and (31), we can rewrite the term $\big(\beta \mathbin{\dot{-}} \pi_Y(\bar{0})^*\big) \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$ into $\Big(\big(\beta \mathbin{\dot{-}} \pi_Y(\bar{0})^* \succcurlyeq \beta\big) \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]\Big) \succcurlyeq \beta \ominus \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$, where $\beta \ominus \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$ is the approximation used in step $\textcircled{16}$, and re-evaluate the $\epsilon$-membership test on it.

Implemented naïvely, the computation of subsequent approximations of fixpoints would involve a lot of redundancy, e.g., in $\beta \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$ we would need to recompute the term $\{t\} \mathbin{\dot{-}} \left[\begin{smallmatrix} X:1 \\ Y:0 \end{smallmatrix}\right]$, which was already computed in step $\textcircled{15}$. The mechanism can, however, be implemented efficiently so that it completely avoids the redundant computations. Firstly, we can maintain a cache of already evaluated terms and never evaluate the same term repeatedly. Secondly, suppose that a term $t \mathbin{\dot{-}} S^*$ has been unfolded several times into intermediate states $(T_1 = \{t\}) \mathbin{\dot{-}} S^*, T_2 \mathbin{\dot{-}} S^*, \ldots, T_n \mathbin{\dot{-}} S^*$. One more unfolding using (14) would rewrite $T_n \mathbin{\dot{-}} S^*$ into $T_{n+1} = (T_n \cup (T_n \ominus S)) \mathbin{\dot{-}} S^*$. When computing the set $T_n \ominus S$, however, we do not need to consider the whole set $T_n$, but only those elements that are in $T_n$ and are not in $T_{n-1}$ (since $T_n = T_{n-1} \cup (T_{n-1} \ominus S)$, all elements of $T_{n-1} \ominus S$ are already in $T_n$). Thirdly, in the DAG representation of terms described in Section 5.3, a term $(T \cup (T \ominus S)) \dot{-} S^* \succcurlyeq T \cup (T \ominus S)$ is represented by the set of terms obtained by evaluating $T \ominus S$, a pointer to the term $T \dot{-} S^*$ (or rather to its associated DAG node), and the set of symbols $S$. The cost of keeping the history of quotienting together with the under-approximation (on the right-hand side of $\succcurlyeq$) is hence only a pointer and a set of symbols.

### 5.3 Efficient Data Structures

We describe two important techniques used in our implementation that concern (1) representation of terms and (2) utilisation of BDD-based symbolic representation of transition functions of automata in the leaves of the terms.

*Representation of language terms.* We keep the term in the form of a DAG such that all isomorphic instances of the same term are represented as a unique DAG node, and, moreover, when a term is rewritten into a language-equivalent one, it is still associated with the same DAG node. Newly computed sub-terms are always first compared against the existing ones, and, if possible, associated with an existing DAG node of an existing isomorphic term. The fact that isomorphic terms are always represented by the same DAG node makes it possible to test isomorphism of a new and previously processed term efficiently—it is enough to test that their direct sub-terms are represented by identical DAG nodes (let us note that we do not look for language equivalent terms because of the high cost of such a check).

We also cache results of membership and subsumption queries. The key to the cache is the identity of DAG nodes, not the represented sub-terms, which has the advantage that results of tests over a term are available in the cache even after it is rewritten according to $\rightarrow$ (as it is still represented by the same DAG node). The cache together with the DAG representation is especially efficient when evaluating a new subsumption or $\epsilon$-membership test since although the result is not in the cache, the results for its sub-terms often are. We also maintain the cache of subsumptions closed under transitivity.

*BDD-based symbolic automata.* Coping with large sets of symbols is central for our algorithm. Notice that rules (14) and (18) compute a quotient for each of the symbols in the set $\pi_{\mathcal{X}}(\tau)$ separately. Since the number of the symbols is $2^{|\mathcal{X}|}$, this can easily make the computation infeasible.

MONA resolves this by using a BDD-based symbolic representation of transition relations of automata as follows: The alphabet symbols of the automata are assignments of Boolean values to the free variables $X_1, \ldots, X_n$ of a formula. The transitions leading from a state $q$ can be expressed as a function $f_q : 2^{\{X_1,\ldots,X_n\}} \to Q$ from all assignments to states such that $(q, \tau, q') \in \delta_q$ iff $f_q(\tau) = q'$. The function $f_q$ is encoded as a multi-terminal BDD (MTBDD) with variables $X_1, \ldots, X_n$ and terminals from the set $Q$ (essentially, it is a DAG where a path from the root to a leaf encodes a set of transitions). The BDD `apply` operation is then used to efficiently implement the computation of successors of a state via a large set of symbols, and to facilitate essential constructions such as product, determinization, and minimization. We use MONA to create automata in leaves of our language terms. To fully utilize their BDD-based symbolic representation, we had to overcome the following two problems.

First, our algorithm computes predecessors of states, while the BDDs of MONA are meant to compute successors. To use `apply` to compute backwards, the BDDs would have to be turned into a representation of the inverted transition function. This is costly and, according to our experience, prone to produce much larger BDDs. We have resolved this by only inverting the edges of the original BDDs and by implementing a variant of `apply` that runs upwards from the leaves of the original BDDs, against the direction of the original BDD edges. It cannot be as efficient as the normal `apply` because, unlike standard BDDs, the DAG that arises by inverting BDD edges is nondeterministic, which brings complications. Nevertheless, it still allows an efficient implementation of $pre$ that works well in our implementation.

A more fundamental problem we are facing is that our algorithm can use `apply` to compute predecessors over the compact representation provided by BDDs only on the level of explicit automata in the leaves of terms. The symbols generated by projection during evaluation of complex terms must be, on the contrary, enumerated explicitly. For instance, the projection $\pi_{\mathcal{X}}(t)$ with $\mathcal{X} = \{X_1, \ldots, X_n\}$ generates $2^n$ symbols, with no obvious option for reduction. The idea to overcome this explosion is to treat nodes of BDDs as regular automata states. Intuitively, this means replacing words over $\Sigma_{\mathcal{X}}$ that encode models of formulae by words over the alphabet $\{0, 1\}$: every symbol $\tau \in \Sigma_{\mathcal{X}}$ is replaced by the *string* $\tau$ over $\{0, 1\}$. Then, instead of computing a quotient over, e.g., the set $\pi_{\mathcal{X}}(\bar{0})$ of the size $2^n$, we compute only quotients over the 0's and 1's. Each quotienting takes us only one level down in the BDDs representing the transition relation of the automata in the leaves of the term. For every variable $X_i$, we obtain terms over nodes on the $i$-th level of the BDDs as $-0$ and $-1$ quotients of the terms at the level $i - 1$. The maximum number of terms in each level is thus $2^i$. In the worst case, this causes roughly the same blow-up as when enumerating the "long" symbols. The advantage of this techniques is, however, that the blow-up can now be dramatically reduced by using subsumption to prune sets of terms on the individual BDD levels.

### 5.4 Combination of Symbolic and Explicit Algorithms

It is possible to replace sub-terms of a language term by a language-equivalent automaton built by the explicit algorithm before starting the symbolic algorithm. The main benefit of this is that the explicitly constructed automata have a simpler flat structure and can be minimized. The minimization, however, requires to explicitly construct the whole automaton, which might, despite the benefit of minimization, be a too large overhead. The combination hence represents a trade-off between the lazy evaluation and subsumption of the symbolic algorithm, and minimization and flat automata structure of the explicit one. The overall effect depends on the strategy of choice of the sub-formulae to be translated into automata, and, of course, on the efficiency of the implementation of the explicit algorithm (where we can leverage the extremely efficient implementation of MONA). We mention one particular strategy for choosing sub-formulae in Section 6.

### 5.5 Anti-prenexing

Before rewriting an input formula to a symbolic term, we pre-process the formula by moving quantifiers down by several language-preserving identities (which we call *anti-prenexing*). We, e.g., change $\exists X.\, (\varphi \wedge \psi)$ into $\varphi \wedge (\exists X.\, \psi)$ if $X$ is not free in $\varphi$. Moving a quantifier down in the abstract syntax tree of a formula speeds up the fixpoint computation induced by the quantifier. In effect, one costlier fixpoint computation is replaced by several cheaper computations in the sub-formulae. This is almost always helpful since if the original fixpoint computation unfolds, e.g., a union of two terms, the two fixpoint computations obtained by anti-prenexing will each unfold only one operand of the union. The number of union terms in the original fixpoint is roughly the product of the numbers of terms in the simpler fixpoints. Further, in order to push quantifiers even deeper into the formula, we reorder the formula by several heuristics (e.g. group sub-formulae with free occurrences of the same variable in a large conjunction) and move negations down in the structure towards the leaves using De Morgan's laws.

## 6 Experiments

We have implemented the proposed approach in a prototype tool GASTON[3], Our tool uses the front-end of MONA to parse input formulae, to construct their abstract syntax trees, and also to construct automata for sub-formulae (as mentioned in Section 5.4). From several heuristics for choosing the sub-formulae to be converted to automata by MONA, we converged to converting only quantifier free sub-formulae and negations of innermost quantifiers to automata since MONA can usually handle them without any explosion. GASTON, together with all the benchmarks described below and their detailed results, is freely available [22].

We compared GASTON's performance with that of MONA, DWINA implementing our older approach [18], TOSS implementing the method of [19], and the implementations of the decision procedures of [20] and [15] (which we denote as COALG and SFA, respectively).[4] In our experiments, we consider formulae obtained from various formal verification tasks as well as parametric families of formulae designed to stress-test WS1S decision procedures.[5] We performed the experiments on a machine with the Intel Core i7-2600@3.4 GHz processor and 16 GiB RAM running Debian GNU/Linux.

**Table 1.** UABE experiments

| Formula | MONA | | GASTON | |
|---|---|---|---|---|
| | Time | Space | Time | Space |
| a-a | **1.71** | 30 253 | $>_{2m}$ | $>_{2m}$ |
| ex10 | **7.71** | 131 835 | 12.67 | 82 236 |
| ex11 | 4.40 | 2 393 | **0.18** | 4 156 |
| ex12 | **0.13** | 2 591 | 6.31 | 68 159 |
| ex13 | **0.04** | 2 601 | 1.19 | 16 883 |
| ex16 | **0.04** | 3 384 | 0.28 | 3 960 |
| ex17 | 3.52 | 165 173 | **0.17** | 3 952 |
| ex18 | **0.27** | 19 463 | $>_{2m}$ | $>_{2m}$ |
| ex2 | 0.18 | 26 565 | **0.01** | 1 841 |
| ex20 | 1.46 | 1 077 | **0.27** | 12 266 |
| ex21 | **1.68** | 30 253 | $>_{2m}$ | $>_{2m}$ |
| ex4 | **0.08** | 6 797 | 0.50 | 22 442 |
| ex6 | **4.05** | 27 903 | 22.69 | 132 848 |
| ex7 | 0.90 | 857 | **0.01** | 594 |
| ex8 | 7.69 | 106 555 | **0.03** | 1 624 |
| ex9 | 7.16 | 586 447 | 9.41 | 412 417 |
| fib | **0.10** | 8 128 | 24.19 | 126 688 |

Table 1 contains results of our experiments with formulae from the recent work [24] (denoted as UABE below), which uses WS1S to reason about programs with unbounded arrays. Table 2 gives results of our experiments with formulae derived from the WS1S-based shape analysis of [2] (denoted as Strand). In the table, we use sl to denote Strand formulae over sorted lists and bs for formulae from verification of the bubble sort procedure. For this set of experiments, we considered MONA and GASTON only since the other tools were missing features (e.g., atomic predicates) needed to handle the formulae. In the UABE benchmark, GASTON was used with the last optimization

[3] 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.

[4] We are not comparing with JMOSEL [13] as we did not find it available on the Internet.

[5] We note that GASTON currently does not perform well on formulae with many Boolean variables and M2L formulae appearing in benchmarks such as Secrets [11] or Strand2 [1,23], which are not included in our experiments. To handle such formulae, further optimizations of GASTON such as MONA's treatment of Boolean variables via a dedicated transition are needed.

of Section 5.3 (treating MTBDD nodes as automata states) to efficiently handle quantifiers over large numbers of variables. In particular, without the optimization, Gaston hit 11 more timeouts. On the other hand, this optimization was not efficient (and hence not used) in `Strand`.

The tables compare the overall time (in seconds) the tools needed to decide the formulae, and they also try to characterize the sizes of the generated state spaces. For the latter, we count the overall number of states of the generated automata for Mona, and the overall number of generated sub-terms for Gaston. The tables contain just a part of the results, more can be found in [22]. We use $>_{2\text{m}}$ in case the running time exceeded 2 minutes, `oom` to denote that the tool ran out of memory, $+k$ to denote that we added $k$ quantifier alternations to the original benchmark, and `N/A` to denote that the benchmark requires some feature or atomic predicate unsupported by the given tool. On `Strand`, Gaston is mostly comparable, in two cases better, and in one case worse than Mona. On `UABE`, Gaston outperformed Mona on six out of twenty-three benchmarks, it was worse on ten formulae, and comparable on the rest. The results thus confirm that our approach can defeat Mona in practice.

**Table 2.** `Strand` experiments

| **Formula** | Mona | | Gaston | |
|---|---|---|---|---|
| | **Time** | **Space** | **Time** | **Space** |
| `bs-loop-else` | 0.05 | 14 469 | 0.04 | 2 138 |
| `bs-loop-if-else` | 0.19 | 61 883 | **0.08** | 3 207 |
| `bs-loop-if-if` | 0.38 | 127 552 | **0.18** | 5 428 |
| `sl-insert-after-loop` | **0.01** | 2 634 | 0.36 | 5 066 |
| `sl-insert-before-head` | 0.01 | 678 | 0.01 | 541 |
| `sl-insert-before-loop` | 0.01 | 1 448 | 0.01 | 656 |
| `sl-insert-in-loop` | 0.02 | 5 945 | 0.01 | 1 079 |
| `sl-reverse-after-loop` | 0.01 | 1 941 | 0.01 | 579 |
| `sl-search-in-loop` | 0.08 | 23 349 | 0.03 | 3 247 |

The second part of our experiments concerns parametric families of WS1S formulae used for evaluation in [19,18,15], and also parameterized versions of selected `UABE` formulae [24]. Each of these families has one parameter (whose meaning is explained in the respective works). Table 3 gives times needed to decide instances of the formulae for the parameter having value 20. If the tools did not manage this value of the parameter, we give in parentheses the highest value of the parameter for which the tools succeeded. More results are available in [22]. In this set of experiments, Gaston managed to win over the other tools on many of their own benchmark formulae. In the first six rows of Table 3, the superior efficiency of Gaston was caused mainly by anti-prenexing. It turns out that this optimization of the input formula is universally effective. When run on anti-prenexed formulae, the performance of the other tools was comparable to that of Gaston. The last two benchmarks (parameterized versions of formulae from `UABE`) show, however, that Gaston's performance does not stand on anti-prenexing only. Despite that its effect here was negligable (similarly as for all the original benchmarks from `UABE` and `Strand`), Gaston still clearly outperformed Mona. We could not compare with other tools on these formulae due to a missing support of the used features (e.g. constants).

**Table 3.** Experiments with parametric families of formulae

| **Benchmark** | **Src** | Mona | dWiNA | Toss | Coalg | SFA | Gaston |
|---|---|---|---|---|---|---|---|
| `HornLeq` | [15] | `oom`(18) | **0.03** | 0.08 | $>_{2\text{m}}$(08) | **0.03** | **0.01** |
| `HornLeq` (+3) | [15] | `oom`(18) | $>_{2\text{m}}$(11) | 0.16 | $>_{2\text{m}}$(07) | $>_{2\text{m}}$(11) | **0.01** |
| `HornLeq` (+4) | [15] | `oom`(18) | $>_{2\text{m}}$(13) | **0.04** | $>_{2\text{m}}$(06) | $>_{2\text{m}}$(11) | **0.01** |
| `HornIn` | [19] | `oom`(15) | $>_{2\text{m}}$(11) | 0.07 | $>_{2\text{m}}$(08) | $>_{2\text{m}}$(08) | **0.01** |
| `HornTrans` | [18] | 86.43 | $>_{2\text{m}}$(14) | `N/A` | `N/A` | 38.56 | **1.06** |
| `SetSingle` | [18] | `oom`(04) | $>_{2\text{m}}$(08) | 0.10 | `N/A` | $>_{2\text{m}}$(03) | **0.01** |
| `Ex8` | [24] | `oom`(08) | `N/A` | `N/A` | `N/A` | `N/A` | **0.15** |
| `Ex11(10)` | [24] | `oom`(14) | `N/A` | `N/A` | `N/A` | `N/A` | **1.62** |

*Acknowledgement.* We thank the anonymous reviewers for their helpful comments on how to improve the presentation in this paper. This work was supported by the Czech Science Foundation (projects 16-17538S and 16-24707Y), the BUT FIT project FIT-S-17-4014, and the IT4IXS: IT4Innovations Excellence in Science project (LQ1602).

## 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. D'Antoni, L., Veanes, M.: Minimization of symbolic automata. In: In Proc. of POPL'14. (2014) 541–554
16. Doyen, L., Raskin, J.F.: Antichain algorithms for finite automata. In: Proc. of TACAS'10. Volume 6015 of LNCS., Springer (2010) 2–22
17. 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
18. Fiedor, T., Holík, L., Lengál, O., Vojnar, T.: Nested antichains for WS1S. In: Proc. of TACAS'15. Volume 9035 of LNCS., Springer (2015)
19. 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

20. 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
21. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications. (2008)
22. Fiedor, T., Holík, L., Janků, P., Lengál, O., Vojnar, T.: GASTON (2016) Available from `http://www.fit.vutbr.cz/research/groups/verifit/tools/gaston/`.
23. Madhusudan, P., Parlato, G., Qiu, X.: Strand benchmark. `http://web.engr.illinois.edu/~qiu2/strand/` Accessed: 2014-01-29.
24. Zhou, M., He, F., Wang, B., Gu, M., Sun, J.: Array theory of bounded elements and its applications. J. Autom. Reasoning **52**(4) (2014) 379–405