Acta Informatica
https://doi.org/10.1007/s00236-018-0331-z

ORIGINAL ARTICLE

Nested antichains for WS1S

Tomáš Fiedor[1] · Lukáš Holík[1] · Ondřej Lengál[1] · Tomáš Vojnar[1]

Received: 31 December 2016 / Accepted: 17 December 2018

Abstract
We propose a novel approach for coping with alternating quantification as the main source of nonelementary complexity of deciding WS1S formulae. Our approach is applicable within the state-of-the-art automata-based WS1S decision procedure implemented e.g. in MONA. The way in which the standard decision procedure processes quantifiers involves determinization, with its worst case exponential complexity, for every quantifier alternation in the prefix of a formula. Our algorithm avoids building the deterministic automata—instead, it constructs only those of their states needed for (dis)proving validity of the formula. It uses a symbolic representation of the states, which have a deeply nested structure stemming from the repeated implicit subset construction, and prunes the search space by a nested subsumption relation, a generalization of the one used by the so-called antichain algorithms for handling nondeterministic automata. We have obtained encouraging experimental results, in some cases outperforming MONA, and some of the other recently proposed approaches, by several orders of magnitude.

1 Introduction

Weak monadic second-order logic of one successor (WS1S) is a powerful, concise, and decidable logic for describing regular properties of finite words. Despite its nonelementary worst case complexity [2], it has been shown useful in numerous applications. Most of the successful applications were due to the tool MONA [3], which implements decision procedures for the

An extended abstract of this paper was first presented in [1]. The current paper extends [1] with a more detailed presentation of the approach, the needed proofs, an illustrating example, and an extended experimental evaluation of the approach.

✉ Ondřej Lengál
lengal@fit.vutbr.cz

Tomáš Fiedor
ifiedortom@fit.vutbr.cz

Lukáš Holík
holik@fit.vutbr.cz

Tomáš Vojnar
vojnar@fit.vutbr.cz

[1] FIT, IT4Innovations Centre of Excellence, Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic

Published online: 01 January 2019

WS1S and WS2S (a generalization of WS1S to finite binary trees) based on finite automata. The authors of MONA list a multitude of its diverse applications [4], ranging from software and hardware verification through controller synthesis to computational linguistics, and further on. Among more recent applications, verification of pointer programs and deciding related logics [5–10] can be mentioned, as well as synthesis from regular specifications [11].

Despite many optimizations implemented in MONA and other tools, the worst case complexity of the problem sometimes strikes back. Authors of methods using the translation of their problem to WS1S/WS2S are then forced to either find workarounds to circumvent the complexity blowup, such as in [6], or give up translating to WS1S/WS2S altogether [12]—often for the price of restricting the input of their approach.

The decision procedure of MONA works with deterministic automata; it uses determinization extensively and relies on minimization of deterministic automata to suppress the complexity blow-up. Nevertheless, the worst case exponential complexity of determinization often significantly harms the performance of the tool. Recent works on efficient methods for handling nondeterministic automata—in particular, works on efficient testing of language inclusion and universality of finite automata [13–15] and works on reducing the size of finite automata using simulation relations [16,17]—suggest a way of alleviating this problem. Handling nondeterministic automata using these methods, while avoiding determinization, has been shown to provide great efficiency improvements in [18] (abstract regular model checking) and also [19] (shape analysis). In this paper, we make a major step towards building the entire decision procedure of WS1S on nondeterministic automata using similar techniques. We propose a generalization of the antichain algorithms of [13] that addresses the main bottleneck of the automata-based decision procedure for WS1S, which is the source of its nonelementary complexity: elimination of alternating quantifiers on the automata level.

More concretely, the automata-based decision procedure translates the input WS1S formula into a finite word automaton such that its language represents exactly all models of the formula. The automaton is built in a bottom-up manner according to the structure of the formula, starting with predefined atomic automata for literals and applying a corresponding automata operation for every logical connective and quantifier ($\wedge, \vee, \neg, \exists$). The cause of the nonelementary complexity of the procedure can be explained on an example formula of the form $\varphi' = \exists X_m \forall X_{m-1} \ldots \forall X_2 \exists X_1 : \varphi_0$. The universal quantifiers are first replaced by negation and existential quantification, which results in $\varphi = \exists X_m \neg \exists X_{m-1} \ldots \neg \exists X_2 \neg \exists X_1 : \varphi_0$. The algorithm then builds a sequence of automata for the sub-formulae $\varphi_0, \varphi_0^\sharp, \ldots, \varphi_{m-1}, \varphi_{m-1}^\sharp$ of φ where $\varphi_i^\sharp = \exists X_{i+1} : \varphi_i$ and $\varphi_{i+1} = \neg \varphi_i^\sharp$ for $0 \leq i < m$. Every automaton in the sequence is created from the previous one by applying the automata operations corresponding to negation or elimination of the existential quantifier, the latter of which may introduce nondeterminism. Negation applied on a nondeterministic automaton may then yield an exponential blowup: given an automaton for ψ, the automaton for $\neg\psi$ is constructed by the classical automata-theoretic construction consisting of determinization by the subset construction followed by swapping of the sets of final and non-final states. The subset construction is exponential in the worst case. The worst case complexity of the procedure run on φ is then a tower of exponentials with one level for every quantifier alternation in φ; note that this high computational cost cannot be avoided completely—indeed, the non-elementary complexity is an inherent property of the problem.

Overview of the proposed algorithm Our new algorithm for processing alternating quantifiers in the prefix of a formula avoids the explicit determinization of automata in the classical procedure and significantly reduces the state space explosion associated with it. It is based on a generalization of the antichain principle used for deciding universality and language inclusion of finite automata [14,15]. It generalizes the antichain algorithms so that instead of being used to process only one level of the chain of automata, it processes the whole chain of quantifications with i alternations on-the-fly. This leads to working with automata states that are sets of sets of sets … of states of the automaton representing φ_0 with the nesting depth i (this corresponds to i levels of subset construction being done on-the-fly). The algorithm uses nested symbolic terms to represent sets of such automata states and a generalized version of antichain pruning based on a notion of subsumption that descends recursively down the structure of the terms while pruning on all their levels.

Experimental evaluation We have implemented the proposed approach in a prototype tool called dWiNA and compared its performance with other publicly available WS1S solvers on both generated formulae and formulae obtained from various verification tasks. From the experiments, we have obtained encouraging results showing that there are cases in which dWiNA outperforms Mona as well as other recently proposed decision procedures.

Related work Mona is still the standard tool and the most common choice when it comes to deciding WS1S/WS2S. Its efficiency stems from many optimizations, both higher-level (such as automata minimization or the use of MTBDDs for encoding the transition relation of the automata) as well as lower-level (e.g. optimizations of hash tables) [20]. There are other related automata-based tools that are more recent—in particular, jMosel [21] for the M2L(Str) logic and the procedure using symbolic finite automata implemented within the `Automata` library of D'Antoni et al. [22]. They implement optimizations that allow them to outperform Mona on some benchmarks, however, none of them provides an evidence of being consistently more efficient.

Some other recent approaches are logic-based and completely avoid any explicit automata construction. Ganzow and Kaizer [23] developed a new decision procedure for the weak monadic second-order logic on inductive structures, a more general logic than WSkS. Their method is based on the Shelah's composition method, and it is implemented within the Toss tool. It sometimes performs better than Mona, but it lacks support of syntactic features that would allow one to perform a comparison on more benchmarks.

Traytel [24], on the other hand, uses the classical decision procedure recast in the framework of coalgebras. His work is based on testing equivalence of a pair of formulae by finding a bisimulation between its derivatives. The implementation is, however, not optimized enough, and it is easily outperformed by the rest of the recent tools.

Plan of the paper We define the WS1S logic in Sect. 2. In Sects. 3 and 4, we introduce finite word automata and describe the classical decision procedure for WS1S based on finite word automata. In Sect. 5, we introduce our method for dealing with alternating quantifiers. Finally, we give an experimental evaluation and conclude the paper in Sects. 6 and 7.

2 WS1S

In this section, we introduce the *weak monadic second-order logic of one successor* (WS1S). We introduce only its minimal syntax here, for the full standard syntax and a more thorough introduction, see Section 3.3 in [25].

WS1S is a monadic second-order logic over the set of non-negative integers $\mathbb{N}_0$. This means that the logic allows second-order *variables*, usually denoted using upper-case letters $X, Y, \ldots$, that range over finite subsets of $\mathbb{N}_0$, e.g. $X = \{0, 3, 42\}$. Atomic formulae are of the form (i) $X \subseteq Y$ and (ii) $X = Y+1$, where X and Y are variables. The atomic formulae are interpreted in turn as (i) standard set inclusion, and (ii) $X = \{x\}$ and $Y = \{y\}$ are singletons and x is the successor of y, i.e. $x = y + 1$. Formulae are built from the atomic formulae using the logical connectives $\wedge$, $\vee$, $\neg$, and the quantifier $\exists X$ (for a second-order variable X).

Given a WS1S formula $\varphi(X_1, \ldots, X_n)$ with free variables $X_1, \ldots, X_n$, the assignment $\rho = \{X_1 \mapsto S_1, \ldots, X_n \mapsto S_n\}$, where $S_1, \ldots, S_n$ are finite subsets of $\mathbb{N}_0$, *satisfies* φ, written as $\rho \models \varphi$, if the formula holds when every variable X_i is replaced with its corresponding value $S_i = \rho(X_i)$. We say that φ is *valid*, denoted as $\models \varphi$, if it is satisfied by all assignments of its free variables to finite subsets of $\mathbb{N}_0$. Observe the limitation to *finite* subsets of $\mathbb{N}_0$ (related to the adjective *weak* in the name of the logic); a WS1S formula can indeed only have finite models (although there may be infinitely many of them).

3 Preliminaries

We now provide some preliminaries on downward and upward closed sets and on finite automata.

Downward and upward closed sets For a set D and a set $\mathbb{S} \subseteq 2^D$ we use $\downarrow\mathbb{S}$ to denote the *downward closure* of $\mathbb{S}$, i.e. the set $\downarrow\mathbb{S} = \{R \subseteq D \mid \exists S \in \mathbb{S} : R \subseteq S\}$, and $\uparrow\mathbb{S}$ to denote the *upward closure* of $\mathbb{S}$, i.e. the set $\uparrow\mathbb{S} = \{R \subseteq D \mid \exists S \in \mathbb{S} : R \supseteq S\}$. The set $\mathbb{S}$ is in both cases called the set of *generators* of $\uparrow\mathbb{S}$ or $\downarrow\mathbb{S}$ respectively. A set $\mathbb{S}$ is *downward closed* if it equals its downward closure, $\mathbb{S} = \downarrow\mathbb{S}$, and *upward closed* if it equals its upward closure, $\mathbb{S} = \uparrow\mathbb{S}$. The *choice* operator $\coprod$ (sometimes also called the unordered Cartesian product) is an operator that, given a set of sets $\mathbb{D} = \{D_1, \ldots, D_n\}$, returns the set of all sets $\{d_1, \ldots, d_n\}$ obtained by taking one element d_i from every set D_i. Formally,

$$\coprod\mathbb{D} = \left\{\{d_1, \ldots, d_n\} \mid (d_1, \ldots, d_n) \in \prod_{i=1}^{n} D_i\right\} \tag{1}$$

where $\prod$ denotes the Cartesian product. We use the $\coprod$ operator to represent the complement of a downward-closed set represented using its generators: $\uparrow\coprod\overline{\mathbb{D}} = \overline{\downarrow\mathbb{D}}$. Note that for a set D, $\coprod\{D\}$ is the set of all singleton subsets of D, i.e. $\coprod\{D\} = \{\{d\} \mid d \in D\}$. Further note that if any D_i is the empty set $\emptyset$, the result is $\coprod\mathbb{D} = \emptyset$. The following lemmas show important properties of $\coprod$ that are used later.

Lemma 1 *Let $\mathbb{X}$ and $\mathbb{Y}$ be sets of sets. Then it holds that*

$$\uparrow\coprod\mathbb{X} \cap \uparrow\coprod\mathbb{Y} = \uparrow\coprod(\mathbb{X} \cup \mathbb{Y}). \tag{2}$$

Proof From the definition of the $\coprod$ operator, it holds that

$$\begin{aligned} \uparrow\coprod\mathbb{X} &= \uparrow\left\{\{x_1, \ldots, x_n\} \,\middle|\, (x_1, \ldots, x_n) \in \prod\mathbb{X}\right\} \quad \text{and} \\ \uparrow\coprod\mathbb{Y} &= \uparrow\left\{\{y_1, \ldots, y_m\} \,\middle|\, (y_1, \ldots, y_m) \in \prod\mathbb{Y}\right\}. \end{aligned} \tag{3}$$

Notice that the intersection of a pair of upward closed sets given by their generators can be constructed by taking all pairs of generators (X, Y), s.t. X is from $\coprod\mathbb{X}$ and Y is from $\coprod\mathbb{Y}$, and constructing the sets $X \cup Y$. It is easy to see that $X \cup Y$ is a generator of $\uparrow \coprod \mathbb{X} \cap \uparrow \coprod \mathbb{Y}$ and that $\uparrow \coprod \mathbb{X} \cap \uparrow \coprod \mathbb{Y}$ is generated by all such pairs, i.e. that $\uparrow \coprod \mathbb{X} \cap \uparrow \coprod \mathbb{Y}$ is equal to

$$\uparrow \left\{ \{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_m\} \,\middle|\, (x_1, \ldots, x_n) \in \prod X \wedge (y_1, \ldots, y_m) \in \prod Y \right\}. \quad (4)$$

We observe that this set can be also expressed as

$$\uparrow \left\{ \{x_1, \ldots, x_n, y_1, \ldots, y_m\} \,\middle|\, (x_1, \ldots, x_n, y_1, \ldots y_m) \in \prod (X \cup Y) \right\} \quad (5)$$

or, to conclude the proof, as $\uparrow \coprod (\mathbb{X} \cup \mathbb{Y})$. □

Lemma 2 *Let $\mathbb{R}$ be a set of sets. Then, it holds that*

$$\uparrow \coprod \mathbb{R} = \bigcap_{R_j \in \mathbb{R}} \uparrow \coprod \{R_j\}. \quad (6)$$

Proof Because intersection and union are both associative operations and $\mathbb{R} = \{R_1, \ldots, R_n\}$, this lemma is a simple consequence of Lemma 1. □

Let $\mathbb{X}$ be a 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 $\mathbb{X} = \{X_1, X_2\}$. An *alphabet* over $\mathbb{X}$ is the set of all symbols over $\mathbb{X}$, denoted as $\Sigma_{\mathbb{X}}$. For any $\mathbb{X}$ (even empty), we use $\bar{0}$ to denote the symbol which maps all variables from $\mathbb{X}$ to 0, $\bar{0} \in \Sigma_{\mathbb{X}}$.

Finite automata A (nondeterministic) *finite* (word) *automaton* (abbreviated as FA in the following) over a set of variables $\mathbb{X}$ is a quadruple $\mathcal{A} = (Q, \Delta, I, F)$ where Q is a finite set of states, $I \subseteq Q$ is a set of *initial* states, $F \subseteq Q$ is a set of *final* states, and Δ is a set of transitions of the form (p, τ, q) where $p, q \in Q$ and $\tau \in \Sigma_{\mathbb{X}}$. We use $p \xrightarrow{\tau} q \in \Delta$ to denote that $(p, \tau, q) \in \Delta$. Note that for an FA $\mathcal{A}$ over $\mathbb{X} = \emptyset$, $\mathcal{A}$ is a unary FA with the alphabet $\Sigma_{\mathbb{X}} = \{\bar{0}\}$.

A *run* r of $\mathcal{A}$ over a word $w = \tau_1\tau_2 \ldots \tau_n \in \Sigma_{\mathbb{X}}^*$ from the state $p \in Q$ to the state $s \in Q$ is a sequence of states $r = q_0q_1 \ldots q_n \in Q^+$ such that $q_0 = p$, $q_n = s$ and for all $1 \leq i \leq n$ there is a transition $q_{i-1} \xrightarrow{\tau_i} q_i$ in Δ. If $s \in F$, we say that r is an *accepting run*. We write $p \overset{w}{\Longrightarrow} s$ to denote that there exists a run from the state p to the state s over the word w. The *language* accepted by a state q is defined by $\mathcal{L}_{\mathcal{A}}(q) = \{w \mid q \overset{w}{\Longrightarrow} q_f, q_f \in F\}$, while the language of a set of states $S \subseteq Q$ is defined as $\mathcal{L}_{\mathcal{A}}(S) = \bigcup_{q \in S} \mathcal{L}_{\mathcal{A}}(q)$. When it is clear which FA $\mathcal{A}$ we refer to, we only write $\mathcal{L}(q)$ or $\mathcal{L}(S)$. The language of $\mathcal{A}$ is defined as $\mathcal{L}(\mathcal{A}) = \mathcal{L}_{\mathcal{A}}(I)$. We say that the state q accepts w and that the automaton $\mathcal{A}$ accepts w to express $w \in \mathcal{L}_{\mathcal{A}}(q)$ and $w \in \mathcal{L}(\mathcal{A})$ respectively. We call a language $L \subseteq \Sigma_{\mathbb{X}}^*$ *universal* iff $L = \Sigma_{\mathbb{X}}^*$.

For a set of states $S \subseteq Q$, we define

$$\begin{aligned} post[\Delta,\tau](S) &= \bigcup_{s \in S} \{t \mid s \xrightarrow{\tau} t \in \Delta\}, \\ pre[\Delta,\tau](S) &= \bigcup_{s \in S} \{t \mid t \xrightarrow{\tau} s \in \Delta\}, \text{ and} \\ cpre[\Delta,\tau]\,(S) &= \{t \mid post[\Delta,\tau](\{t\}) \subseteq S\}. \end{aligned}$$

The *complement* of $\mathcal{A}$ is the automaton $\mathcal{A}_{\mathcal{C}} = (2^Q, \Delta_{\mathcal{C}}, \{I\}, \downarrow\{Q \backslash F\})$ where $\Delta_{\mathcal{C}} = \left\{ P \xrightarrow{\tau} post[\Delta,\tau](P) \,\middle|\, P \subseteq Q \right\}$; this corresponds to the standard procedure that first determinizes $\mathcal{A}$ by the subset construction and then swaps its sets of final and non-final states, and

$\downarrow\{Q \setminus F\}$ is the set of all subsets of Q that do not contain a final state of $\mathcal{A}$. The language of $\mathcal{A}_{\mathcal{C}}$ is the complement of the language of $\mathcal{A}$, i.e., $\mathcal{L}(\mathcal{A}_{\mathcal{C}}) = \overline{\mathcal{L}(\mathcal{A})}$.

For a set of variables $\mathbb{X}$ and a variable X, the *projection* of X from $\mathbb{X}$, denoted as $\pi_{[X]}(\mathbb{X})$, is the set $\mathbb{X} \setminus \{X\}$. For a symbol τ, the projection of X from τ, denoted $\pi_{[X]}(\tau)$, is obtained from τ by restricting τ to the domain $\pi_{[X]}(\mathbb{X})$. For a transition relation Δ, the projection of X from Δ, denoted as $\pi_{[X]}(\Delta)$, is the transition relation $\left\{p \xrightarrow{\pi_{[X]}(\tau)} q \mid p \xrightarrow{\tau} q \in \Delta\right\}$.

4 Deciding WS1S with finite automata

The classical decision procedure for WS1S [26] (as described in Section 3.3 of [25]) is based on a logic-automata connection and decides validity (satisfiability) of a WS1S formula $\varphi(X_1, \ldots, X_n)$ by constructing the FA $\mathcal{A}_\varphi$ over $\{X_1, \ldots, X_n\}$ which recognizes encodings of exactly the models of φ. The automaton is built in a bottom-up manner, according to the structure of φ, starting with predefined atomic automata for literals and applying a corresponding automata operation for every logical connective and quantifier ($\wedge$, $\vee$, $\neg$, $\exists$). Hence, for every sub-formula ψ of φ, the procedure will compute the automaton $\mathcal{A}_\psi$ such that $\mathcal{L}(\mathcal{A}_\psi)$ represents exactly all models of ψ, terminating with the result $\mathcal{A}_\varphi$.

The alphabet of $\mathcal{A}_\varphi$ consists of all symbols over the set $\mathbb{X} = \{X_1, \ldots, X_n\}$ of free variables of φ (for $a, b \in \{0, 1\}$ and $\mathbb{X} = \{X_1, X_2\}$, we use $\substack{X_1 : a \\ X_2 : b}$ to denote the symbol $\{X_1 \mapsto a, X_2 \mapsto b\}$). A word w from the language of $\mathcal{A}_\varphi$ is a sequence of these symbols, e.g. $\substack{X_1 : \epsilon \\ X_2 : \epsilon}$, $\substack{X_1 : 011 \\ X_2 : 101}$, or $\substack{X_1 : 01100 \\ X_2 : 10100}$. We denote the ith symbol of w as $w[i]$, for $i \in \mathbb{N}_0$. An assignment $\rho : \mathbb{X} \to 2^{\mathbb{N}_0}$ mapping free variables $\mathbb{X}$ of φ to subsets of $\mathbb{N}_0$ is encoded into a word w_ρ of symbols over $\mathbb{X}$ in the following way: w_ρ contains 1 in the jth position of the row for X_i iff $j \in X_i$ in ρ. Formally, for every $i \in \mathbb{N}_0$ and $X_j \in \mathbb{X}$, if $i \in \rho(X_j)$, then $w_\rho[i]$ maps $X_j \mapsto 1$. On the other hand, if $i \notin \rho(X_j)$, then either $w_\rho[i]$ maps $X_j \mapsto 0$, or the length of w is smaller than or equal to i. Notice that there exist an infinite number of encodings of ρ. The shortest one is w_ρ^s 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_ρ^s extended with an arbitrary number of $\overline{0}$ symbols appended to its end.

For example, $\substack{X_1 : 0 \\ X_2 : 1}$, $\substack{X_1 : 00 \\ X_2 : 10}$, $\substack{X_1 : 000 \\ X_2 : 100}$, $\substack{X_1 : 000\ldots0 \\ X_2 : 100\ldots0}$ are all encodings of the assignment $\rho = \{X_1 \mapsto \emptyset, X_2 \mapsto \{0\}\}$. For the soundness of the decision procedure, it is important that $\mathcal{A}_\varphi$ always accepts either all encodings of ρ or none of them.

The automata $\mathcal{A}_{\varphi \wedge \psi}$ and $\mathcal{A}_{\varphi \vee \psi}$ are constructed from $\mathcal{A}_\varphi$ and $\mathcal{A}_\psi$ by standard automata-theoretic union and intersection operations, preceded by the so-called cylindrification which unifies the alphabets of $\mathcal{A}_\varphi$ and $\mathcal{A}_\psi$. Since these operations, as well as the automata for the atomic formulae, are not the subject of the contribution proposed in this paper, we refer the interested reader to [25] for details.

The part of the procedure which is central for this paper is processing negation and existential quantification; we will therefore describe it in detail. The FA $\mathcal{A}_{\neg\varphi}$ is constructed as the complement of $\mathcal{A}_\varphi$. Then, all encodings of the assignments that were accepted by $\mathcal{A}_\varphi$ are rejected by $\mathcal{A}_{\neg\varphi}$ and vice versa. The FA $\mathcal{A}_{\exists X:\varphi}$ is obtained from the FA $\mathcal{A}_\varphi = (Q, \Delta, I, F)$ by first projecting X from the transition relation Δ, yielding the FA $\mathcal{A}'_\varphi = (Q, \pi_{[X]}(\Delta), I, F)$. However, $\mathcal{A}'_\varphi$ cannot be directly used as $\mathcal{A}_{\exists X:\varphi}$. The reason is that $\mathcal{A}'_\varphi$ may now be inconsistent in accepting some encodings of an assignment ρ while rejecting other encodings of ρ. For example, suppose that $\mathcal{A}_\varphi$ accepts exactly all words starting with $\begin{smallmatrix}X_1: 010\\X_2: 001\end{smallmatrix}$, i.e., $\mathcal{L}(\mathcal{A}_\varphi) = \left\{\begin{smallmatrix}X_1: 010\\X_2: 001\end{smallmatrix}, \begin{smallmatrix}X_1: 0100\\X_2: 0010\end{smallmatrix}, \ldots, \begin{smallmatrix}X_1: 0100\ldots0\\X_2: 0010\ldots0\end{smallmatrix}\right\}$. When computing the FA for $\exists X_2 : \varphi$, we remove the X_2 row from all symbols of all words in $\mathcal{L}(\mathcal{A}_\varphi)$ and obtain the FA $\mathcal{A}'_\varphi$ that accepts the language $\mathcal{L}(\mathcal{A}'_\varphi) = \{X_1: 010, X_1: 0100, \ldots, X_1: 0100\ldots0\}$, but does not accept the word $X_1: 01$ that encodes the same assignment (because $\begin{smallmatrix}X_1: 01\\X_2: ??\end{smallmatrix} \notin \mathcal{L}(A_\varphi)$ for any values in the place of '?'s). As a remedy for this situation, we further need to modify $\mathcal{A}'_\varphi$ to also accept the rest of the encodings of ρ. This is done by enlarging the set of final states of $\mathcal{A}'_\varphi$ to also contain all states that can reach a final state of $\mathcal{A}'_\varphi$ by a sequence of $\bar{0}$ symbols.

Formally, the automaton $\mathcal{A}_{\exists X:\varphi} = (Q, \pi_{[X]}(\Delta), I, F^\sharp)$ is obtained from $\mathcal{A}'_\varphi = (Q, \pi_{[X]}(\Delta), I, F)$ by computing $F^\sharp$ from F using the fixpoint computation $F^\sharp = \mu Z \,.\, F \cup pre_{[\pi_{[X]}(\Delta),\bar{0}]}(Z)$. Intuitively, the least fixpoint denotes the set of states backward-reachable from F following transitions of $\pi_{[X]}(\Delta)$ labelled by $\bar{0}$.

The procedure returns an automaton $\mathcal{A}_\varphi$ that accepts exactly all encodings of the models of φ. This means that the language of $\mathcal{A}_\varphi$ is (i) universal iff φ is valid, (ii) non-universal iff φ is invalid, (iii) empty iff φ is unsatisfiable, and (iv) non-empty iff φ is satisfiable. Notice that in the particular case of *ground* formulae (i.e. formulae without free variables), the language of $\mathcal{A}_\varphi$ is either $\mathcal{L}(\mathcal{A}_\varphi) = \{\bar{0}\}^*$ in the case φ is valid, or $\mathcal{L}(\mathcal{A}_\varphi) = \emptyset$ in the case φ is invalid.

5 Nested antichain-based approach for alternating quantifiers

We now present our approach for dealing with alternating quantifiers in WS1S formulae. We consider a ground formula φ of the form

$$\varphi = \underbrace{\neg\exists\mathcal{X}_m \neg \ldots \neg\exists\mathcal{X}_2 \underbrace{\neg\exists\mathcal{X}_1 : \varphi_0(\mathbb{X})}_{\varphi_1}}_{\varphi_m} \tag{7}$$

where each $\mathcal{X}_i$ is a set of variables $\{X_a, \ldots, X_b\}$, $\exists\mathcal{X}_i$ is an abbreviation for a non-empty sequence $\exists X_a \ldots \exists X_b$ of consecutive existential quantifications, and φ_0 is an arbitrary formula called the *matrix* of φ. Note that the problem of checking validity or satisfiability of a formula with free variables can be easily reduced to this form.

The classical procedure presented in Sect. 4 computes a sequence of automata $\mathcal{A}_{\varphi_0}, \mathcal{A}_{\varphi_0^\sharp}, \ldots, \mathcal{A}_{\varphi_{m-1}^\sharp}, \mathcal{A}_{\varphi_m}$ where for all $0 \le i \le m-1$, $\varphi_i^\sharp = \exists\mathcal{X}_{i+1} : \varphi_i$ and $\varphi_{i+1} = \neg\varphi_i^\sharp$. The φ_i's are the subformulae of φ shown in (7). Since eliminating existential quantification on the automata level introduces nondeterminism (due to the projection on the transition relation), every $\mathcal{A}_{\varphi_i^\sharp}$ may be nondeterministic. The computation of $\mathcal{A}_{\varphi_{i+1}}$ then involves subset construction and becomes exponential. The worst case complexity of eliminating the prefix is therefore the tower of exponentials of the height m. Even though the construction may be optimized, e.g. by minimizing every $\mathcal{A}_{\varphi_i}$ (which is implemented by MONA), the size of the generated automata can quickly become intractable.

The main idea of our algorithm is inspired by the antichain algorithms [13] for testing language universality of an automaton $\mathcal{A}$. In a nutshell, testing universality of $\mathcal{A}$ is testing whether in the complement $\overline{\mathcal{A}}$ of $\mathcal{A}$ (which is created by determinization via subset construction, followed by swapping final and non-final states), an initial state can reach a final state. The crucial idea of the antichain algorithms is based on the following: (i) the search can be done on the fly while constructing $\overline{\mathcal{A}}$. (ii) The sets of states that arise during the search are closed (upward or downward, depending on the variant of the algorithm). (iii) The computation can be done symbolically on the generators of these closed sets. It is enough to keep only the extreme generators of the closed sets (maximal for downward closed, minimal for upward closed). The generators that are not extreme (we say that they are *subsumed*) can be pruned away, which vastly reduces the search space.

We notice that individual steps of the algorithm for constructing $\mathcal{A}_\varphi$ are very similar to testing universality. Automaton $\mathcal{A}_{\varphi_i}$ arises by subset construction from $\mathcal{A}_{\varphi_{i-1}^\sharp}$, and to compute $\mathcal{A}_{\varphi_i^\sharp}$, it is necessary to compute the set of final states $F_i^\sharp$. Those are states backward reachable from the final states of $\mathcal{A}_{\varphi_i}$ via a subset of transitions of Δ_i (those labelled by symbols projected to $\bar{0}$ by π_{i+1}). To compute $F_i^\sharp$, the antichain algorithms could be actually taken off-the-shelf and run with $\mathcal{A}_{\varphi_{i-1}^\sharp}$ in the role of the input $\mathcal{A}$; then, $\mathcal{A}_{\varphi_i^\sharp}$ would be in the role of $\overline{\mathcal{A}}$. This approach, however, has the following two problems. First, antichain algorithms do not produce the automaton $\overline{\mathcal{A}}$ (here $\mathcal{A}_{\varphi_i^\sharp}$), but only a symbolic representation of a set of (backward) reachable states (here of $F_i^\sharp$). Since $\mathcal{A}_{\varphi_i^\sharp}$ is the input of the construction of $\mathcal{A}_{\varphi_{i+1}}$, the construction of $\mathcal{A}_\varphi$ could not continue. The other problem is that the size of the input $\mathcal{A}_{\varphi_{i-1}^\sharp}$ of the antichain algorithm is only limited by the tower of exponentials of the height $i-1$, and this might be already far out of reach.

The main contribution of our paper is an algorithm that alleviates the two problems mentioned above. It is based on a novel way of performing not only one, but all the $2m$ steps of the construction of $\mathcal{A}_\varphi$ on the fly. It uses a nested symbolic representation of sets of states and a form of nested subsumption pruning on all levels of their structure. This is achieved by a substantial refinement of the basic ideas of antichain algorithms.

5.1 Structure of the algorithm

Let us now start explaining the architecture of our on-the-fly algorithm for handling quantifier alternation. Following the construction of automata described in Sect. 4, the structure of the automata from the previous section, $\mathcal{A}_{\varphi_0}, \mathcal{A}_{\varphi_0^\sharp}, \ldots, \mathcal{A}_{\varphi_{m-1}^\sharp}, \mathcal{A}_{\varphi_m}$, can be described using the following recursive definition. We use $\pi_i(C)$ for any mathematical object C to denote projection of all variables in $\mathcal{X}_1 \cup \cdots \cup \mathcal{X}_i$ from C.

Let $\mathcal{A}_{\varphi_0} = (Q_0, \Delta_0, I_0, F_0)$ be an FA over $\mathbb{X}$. Then, for each $0 \le i < m$, the FAs $\mathcal{A}_{\varphi_i^\sharp}$ and $\mathcal{A}_{\varphi_{i+1}}$ are over the alphabet $\pi_{i+1}(\mathbb{X})$ and have from the construction the following structure:

$$\begin{array}{l|l} \mathcal{A}_{\varphi_i^\sharp} = (Q_i, \Delta_i^\sharp, I_i, F_i^\sharp) \text{ where} & \mathcal{A}_{\varphi_{i+1}} = (Q_{i+1}, \Delta_{i+1}, I_{i+1}, F_{i+1}) \text{ where} \\ \quad \Delta_i^\sharp = \pi_{i+1}(\Delta_i) \text{ and} & \quad \Delta_{i+1} = \left\{ R \xrightarrow{\tau} post[\Delta_i^\sharp, \tau](R) \,\middle|\, R \in Q_{i+1} \right\}, \\ \quad F_i^\sharp = \mu Z \,.\, F_i \cup pre[\Delta_i^\sharp, \bar{0}](Z). & \quad Q_{i+1} = 2^{Q_i}, \quad I_{i+1} = \{I_i\}, \text{ and } F_{i+1} = \Downarrow\{Q_i \setminus F_i^\sharp\}. \end{array}$$

We recall that $\mathcal{A}_{\varphi_i^\sharp}$ directly corresponds to existential quantification of all variables in $\mathcal{X}_i$ (cf. Sect. 4), and $\mathcal{A}_{\varphi_{i+1}}$ directly corresponds to the complement of $\mathcal{A}_{\varphi_i^\sharp}$ (cf. Sect. 3).

A crucial observation behind our approach is that, because φ is ground, $\mathcal{A}_\varphi$ is an FA over an empty set of variables, and, therefore, $\mathcal{L}(\mathcal{A}_\varphi)$ is either the empty set $\emptyset$ or the set $\{\bar{0}\}^*$ (as described in Sect. 4). Therefore, we need to distinguish between these two cases only. To determine which of them holds, we do not need to explicitly construct the automaton $\mathcal{A}_\varphi$. Instead, it suffices to check whether $\mathcal{A}_\varphi$ accepts the empty string ϵ. This is equivalent to checking existence of a state that is at the same time final and initial, that is

$$\models \varphi \quad \text{iff} \quad I_m \cap F_m \neq \emptyset. \tag{8}$$

To compute I_m from I_0 is straightforward (it equals $\{\{\ldots\{\{I_0\}\}\ldots\}\}$ nested m-times). In the rest of the section, we will describe how to compute F_m (in the form of its symbolic representation), and how to test whether it intersects with I_m.

The algorithm takes advantage of the fact that to represent final states, one can use their complement, the set of non-final states. For $0 \leq i \leq m$, we write N_i and $N_i^\sharp$ to denote the sets of non-final states $Q_i \setminus F_i$ of $\mathcal{A}_i$ and $Q_i \setminus F_i^\sharp$ of $\mathcal{A}_i^\sharp$ respectively. The algorithm will then instead of computing the sequence of automata $\mathcal{A}_{\varphi_0}$, $\mathcal{A}_{\varphi_0^\sharp}$, ..., $\mathcal{A}_{\varphi_{m-1}^\sharp}$, $\mathcal{A}_{\varphi_m}$ compute the sequence $F_0, F_0^\sharp, N_1, N_1^\sharp, \ldots$ up to either F_m (if m is even) or N_m (if m is odd), which suffices for testing the validity of φ. The algorithm starts with F_0 and uses the following recursive equations:

$$\begin{array}{ll} \text{(i)}\ F_{i+1} = \downarrow\{N_i^\sharp\}, & \text{(ii)}\ F_i^\sharp = \mu Z\,.\,F_i \cup pre_{[\Delta_i^\sharp,\bar{0}]}(Z), \\ \text{(iii)}\ N_{i+1} = \uparrow\coprod\{F_i^\sharp\}, & \text{(iv)}\ N_i^\sharp = \nu Z\,.\,N_i \cap cpre_{[\Delta_i^\sharp,\bar{0}]}(Z)\,. \end{array} \tag{9}$$

Intuitively, (i) and (ii) are directly from the definition of $\mathcal{A}_i$ and $\mathcal{A}_i^\sharp$. (iii) is a dual of (i): N_{i+1} contains all subsets of Q_i that contain at least one state from $F_i^\sharp$ (cf. the definition of the $\coprod$ operator). Finally, (iv) is a dual of (ii): in the kth iteration of the greatest fixpoint computation, the current set of states Z will contain all states that cannot reach an F_i state over $\bar{0}$ within k steps. In the next iteration, only those states of Z are kept such that all their $\bar{0}$-successors are in Z. Hence, the new value of Z is the set of states that cannot reach F_i over $\bar{0}$ in $k+1$ steps, and the computation stabilises with the set of states that cannot reach F_i over $\bar{0}$ in any number of steps.

In the next two sections, we will show that both of the above fixpoint computations can be carried out symbolically on representatives of upward and downward closed sets. Particularly, in Sects. 5.2 and 5.3, we show how the fixpoints from (ii) and (iv) can be computed symbolically, using subsets of Q_{i-1} as representatives (generators) of upward/downward closed subsets of Q_i. Section 5.4 explains how the above symbolic fixpoint computations can be carried out using nested terms of depth i as a symbolic representation of computed states of Q_i. Section 5.5 shows how to test emptiness of $I_m \cap F_m$ on the symbolic terms, and Sect. 5.6 describes the subsumption relation used to minimize the symbolic term representation used within computations of (ii) and (iv).

5.2 Computing $N_i^\sharp$ on representatives of $\uparrow\coprod\mathcal{R}$-sets

Computing $N_i^\sharp$ at each odd level of the hierarchy of automata is done by computing the greatest fixpoint of the function from Eq. 9(iv):

$$f_{N_i^\sharp}(Z) = N_i \cap cpre_{[\Delta_i^\sharp,\bar{0}]}(Z)\,. \tag{10}$$

We will show that the whole fixpoint computation from Eq. 9(iv) can be carried out symbolically on the representatives of Z due to the following two properties: (a) all intermediate values of Z have the form $\uparrow\coprod\mathcal{R}$, where $\mathcal{R} \subseteq Q_i$, so the sets $\mathcal{R}$ can be used as their symbolic representatives, and (b) *cpre* and $\cap$ can be computed on such a symbolic representation efficiently.

Let us start with the computation of $cpre_{[\Delta_i^\sharp,\tau]}(Z)$ where $\tau \in \pi_{i+1}(\bar{0})$, assuming that Z is of the form $Z = \uparrow\coprod\mathcal{R}$, represented by $\mathcal{R} = \{R_1, \ldots, R_n\}$. From Lemma 2, we have that a set of symbolic representatives $\mathcal{R}$ stands for the intersection of denotations of individual representatives, that is

$$\uparrow\coprod\mathcal{R} = \bigcap_{R_j\in\mathcal{R}} \uparrow\coprod\{R_j\}. \tag{11}$$

The set $cpre_{[\Delta_i^\sharp,\tau]}(Z)$ can thus be written as the *cpre*-image $cpre_{[\Delta_i^\sharp,\tau]}\left(\bigcap\mathcal{S}\right)$ of the intersection of the elements of a set $\mathcal{S} = \left\{\uparrow\coprod\{R_1\},\ldots,\uparrow\coprod\{R_n\}\right\}$. Further, because *cpre* distributes over $\cap$, we can compute the *cpre*-image of an intersection by computing intersection of the *cpre*-images, i.e.

$$cpre_{[\Delta_i^\sharp,\tau]}\left(\bigcap\mathcal{S}\right) = \bigcap_{S\in\mathcal{S}} cpre_{[\Delta_i^\sharp,\tau]}(S)\,. \tag{12}$$

By the definition of $\Delta_i^\sharp$ (where $\Delta_i^\sharp = \pi_{i+1}(\Delta_i)$), the set $cpre_{[\Delta_i^\sharp,\tau]}(S)$ can be computed using the transition relation Δ_i for the price of further refining the intersection. In particular,

$$cpre_{[\Delta_i^\sharp,\tau]}(S) = \bigcap_{\omega\in\pi_{i+1}^{-1}(\tau)} cpre_{[\Delta_i,\omega]}(S)\,. \tag{13}$$

Intuitively, $cpre_{[\Delta_i^\sharp,\tau]}(S)$ contains states from which every transition labelled by *any* symbol that is projected to τ by π_{i+1} has its target in S.

Using (12), (13), and the fact that $Z = \bigcap\left\{\uparrow\coprod\{R_j\}\mid R_j\in\mathcal{R}\right\}$, we obtain

$$cpre_{[\Delta_i^\sharp,\tau]}(Z) = \bigcap_{\substack{R_j\in\mathcal{R}\\ \omega\in\pi_{i+1}^{-1}(\tau)}} cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right). \tag{14}$$

To compute the individual conjuncts $cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$, we take advantage of the special form of the operand $\uparrow\coprod\{R_j\}$ and the fact that Δ_i is, by its definition (obtained from determinization via subset construction), *monotone* w.r.t. $\supseteq$. That is, if $P\xrightarrow{\omega}P'\in\Delta_i$ for some $P, P'\in Q_i$, then for every $R\supseteq P$, there is $R'\supseteq P'$ s.t. $R\xrightarrow{\omega}R'\in\Delta_i$. Due to monotonicity, the $cpre_{[\Delta_i,\omega]}$-image of an upward closed set is also upward closed (proved below). Moreover, we observe that it can be computed symbolically using *pre* on elements of its generators. Particularly, for a set $\uparrow\coprod\{R_j\}$, we get the following lemma:

Lemma 3 *Let $R_j\subseteq Q_{i-1}$ and ω be a symbol over $\pi_i(\mathbb{X})$ for $i>0$. Then*

$$cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right) = \uparrow\coprod\left\{pre_{[\Delta_{i-1}^\sharp,\omega]}(R_j)\right\}. \tag{15}$$

Proof First, we show that the set $cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$ is upward closed. Second, we show that all elements of the set $\coprod\left\{pre_{[\Delta_{i-1}^\sharp,\omega]}(R_j)\right\}$ are contained in $cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$. Finally, we show that for every element T in the set $cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$, the set $\coprod\left\{pre_{[\Delta_{i-1}^\sharp,\omega]}(R_j)\right\}$ contains a smaller or equal element S.

1. Proving that $cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$ is upward closed: Consider a state $S\in Q_i$ s.t. $S\in cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$. From the definition of *cpre*, it holds that

$$post_{[\Delta_i,\omega]}(\{S\}) \subseteq \uparrow\coprod\{R_j\}, \tag{16}$$

and from the definition of Δ_i, it holds that

$$post_{[\Delta_i,\omega]}(\{S\}) = \{post_{[\Delta_{i-1}^\sharp,\omega]}(S)\}. \tag{17}$$

For $T \supseteq S$, it clearly holds that

$$post_{[\Delta^\sharp_{i-1},\omega]}(T) \supseteq post_{[\Delta^\sharp_{i-1},\omega]}(S) \tag{18}$$

and, therefore, it also holds that

$$post_{[\Delta_i,\omega]}(\{T\}) = \{post_{[\Delta^\sharp_{i-1},\omega]}(T)\} \subseteq \uparrow\coprod\{R_j\}. \tag{19}$$

Therefore, $T \in cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$ and the set $cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$ is upward closed.

2. Proving that for all elements $S \in \coprod\left\{pre_{[\Delta^\sharp_{i-1},\omega]}(R_j)\right\}$ it holds that $S \in cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$: From the properties of $\coprod$, it holds that $S = \{s\}$ is a singleton. Because $s \in pre_{[\Delta^\sharp_{i-1},\omega]}(R_j)$, there is a transition $s \xrightarrow{\omega} r \in \Delta^\sharp_{i-1}$ for some $r \in R_j$. Since $post_{[\Delta^\sharp_{i-1},\omega]}(S) \supseteq \{r\}$, it follows from the definition of Δ_i that $post_{[\Delta_i,\omega]}(\{S\}) = \{T\}$ where $T \supseteq \{r\}$, and so $T \in \uparrow\coprod\{R_j\}$ and $post_{[\Delta_i,\omega]}(\{S\}) \subseteq \uparrow\coprod\{R_j\}$. We use the definition of *cpre* to conclude that $S \in cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$.
3. Proving that for every $T \in cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$ there exists some element $S \in \coprod\left\{pre_{[\Delta^\sharp_{i-1},\omega]}(R_j)\right\}$ such that $S \subseteq T$: From $T \in cpre_{[\Delta_i,\omega]}\left(\uparrow\coprod\{R_j\}\right)$ and the definition of Δ_i, we have that

$$post_{[\Delta_i,\omega]}(\{T\}) = \{P\} \subseteq \uparrow\coprod\{R_j\} \tag{20}$$

for P s.t. $post_{[\Delta^\sharp_{i-1},\omega]}(T) = P$. Since $P \in \uparrow\coprod\{R_j\}$, there exist $r \in R_j \cap P$ and $t \in T$ s.t. $t \xrightarrow{\omega} r \in \Delta^\sharp_{i-1}$. Because $t \in pre_{[\Delta^\sharp_{i-1},\omega]}(\{r\})$, we choose $S = \{t\}$ and we are done.

□

Intuitively, the sets with *post*-images above a singleton $\{p\} \in \left\{\{p\} \mid p \in R_j\right\} = \uparrow\coprod\{R_j\}$ in the ordering $\subseteq$ are those that contain at least one state $q \in Q_{i-1}$ s.t. $q \xrightarrow{\omega} p \in \Delta^\sharp_{i-1}$. Combining (14) and Lemma 3 yields

$$cpre_{[\Delta^\sharp_i,\tau]}(Z) = \bigcap_{\substack{R_j\in\mathcal{R}\\ \omega\in\pi^{-1}_{i+1}(\tau)}} \uparrow\coprod\left\{pre_{[\Delta^\sharp_{i-1},\omega]}(R_j)\right\}. \tag{21}$$

By applying Lemma 2, we get the final formula for $cpre_{[\Delta^\sharp_i,\tau]}(Z)$:

$$cpre_{[\Delta^\sharp_i,\tau]}\left(Z = \uparrow\coprod\mathcal{R}\right) = \uparrow\coprod\left\{pre_{[\Delta^\sharp_{i-1},\omega]}(R_j) \mid \omega \in \pi^{-1}_{i+1}(\tau),\, R_j \in \mathcal{R}\right\}. \tag{22}$$

In order to compute $f_{N^\sharp_i}(Z)$, it remains to intersect $cpre_{[\Delta^\sharp_i,\bar{0}]}(Z)$, computed using (22), with N_i. By Eq. 9(iii), N_i equals $\uparrow\coprod\{F^\sharp_{i-1}\}$, and, by Lemma 2, the intersection can be done symbolically as

$$f_{N^\sharp_i}(Z) = \uparrow\coprod\left(\{F^\sharp_{i-1}\} \cup \left\{pre_{[\Delta^\sharp_{i-1},\omega]}(R_j) \mid \omega \in \pi^{-1}_{i+1}(\bar{0}),\, R_j \in \mathcal{R}\right\}\right). \tag{23}$$

Finally, note that a symbolic application of $f_{N^\sharp_i}$ to $Z = \uparrow\coprod\mathcal{R}$ represented as the set $\mathcal{R}$ reduces to computing *pre*-images of the elements of $\mathcal{R}$, which are then put next to each other, together with $F^\sharp_{i-1}$. The computation starts from $N_i = \uparrow\coprod\{F^\sharp_{i-1}\}$, represented by $\{F^\sharp_{i-1}\}$, and each of its steps, implemented by (23), preserves the form of sets $\uparrow\coprod\mathcal{R}$, represented by $\mathcal{R}$.

5.3 Computing $F_i^\sharp$ on representatives of $\downarrow\mathcal{R}$-sets

Similarly as in the previous section, computation of $F_i^\sharp$ at each even level of the automata hierarchy is done by computing the least fixpoint of the function

$$f_{F_i^\sharp}(Z) = F_i \cup pre_{[\Delta_i^\sharp,\bar{0}]}(Z). \tag{24}$$

We will show that the whole fixpoint computation from Eq. 9(ii) can be again carried out symbolically due to the following two properties: (a) all intermediate values of Z are of the form $\downarrow\mathcal{R}, \mathcal{R} \subseteq Q_i$, meaning that the sets $\mathcal{R}$ can be used as their symbolic representatives, and (b) *pre* and $\cup$ can be computed efficiently on such a symbolic representation. The computation is a simpler analogy of the one in Sect. 5.2.

We start with the computation of $pre_{[\Delta_i^\sharp,\tau]}(Z)$ where $\tau \in \pi_{i+1}(\mathbb{X})$, assuming that Z is of the form $\downarrow\mathcal{R}$, represented by $\mathcal{R} = \{R_1, \ldots, R_n\}$. A simple analogy to (11) and (12) of Sect. 5.2 is that the union of downward closed sets is a downward closed set generated by the union of their generators, i.e.

$$\downarrow\mathcal{R} = \bigcup_{R_j \in \mathcal{R}} \downarrow\{R_j\} \tag{25}$$

and that *pre* distributes over union, i.e.

$$pre_{[\Delta_i^\sharp,\tau]}(\downarrow\mathcal{R}) = \bigcup_{R_j \in \mathcal{R}} pre_{[\Delta_i^\sharp,\tau]}(\downarrow\{R_j\}). \tag{26}$$

An analogy of (13) holds too:

$$pre_{[\Delta_i^\sharp,\tau]}(S) = \bigcup_{\omega \in \pi_{i+1}^{-1}(\tau)} pre_{[\Delta_i,\omega]}(S). \tag{27}$$

Intuitively, $pre_{[\Delta_i^\sharp,\tau]}(S)$ contains states from which *at least one* transition labelled by *any* symbol that is projected to τ by π_{i+1} leaves with the target in S. Using (26), (27), and the fact that $Z = \bigcup \left\{\downarrow\{R_j\} \mid R_j \in \mathcal{R}\right\}$, we obtain

$$pre_{[\Delta_i^\sharp,\tau]}(Z) = \bigcup_{\substack{R_j \in \mathcal{R} \\ \omega \in \pi_{i+1}^{-1}(\tau)}} pre_{[\Delta_i,\omega]}(\downarrow\{R_j\}). \tag{28}$$

To compute the individual disjuncts $pre_{[\Delta_i,\omega]}(\downarrow\{R_j\})$, we take advantage of the fact that every $\downarrow\{R_j\}$ is downward closed, and that Δ_i is, by its definition (determinization by subset construction), *monotone* w.r.t. $\subseteq$. That is, if $P \xrightarrow{\omega} P' \in \Delta_i$ for some $P, P' \in Q_i$, then for every $R \subseteq P$, there is $R' \subseteq P'$ s.t. $R \xrightarrow{\omega} R' \in \Delta_i$. Due to monotonicity, the $pre_{[\Delta_i,\omega]}$-image of a downward closed set is downward closed (proved below). Moreover, we observe that it can be computed symbolically using *cpre* on elements of its generators. In particular, for a set $\downarrow\{R_j\}$, we get the following lemma, which is a dual of Lemma 3:

Lemma 4 *Let $R_j \subseteq Q_{i-1}$ and ω be a symbol over $\pi_i(\mathbb{X})$ for $i > 0$. Then*

$$pre_{[\Delta_i,\omega]}(\downarrow\{R_j\}) = \downarrow\{cpre_{[\Delta_{i-1}^\sharp,\omega]}\left(R_j\right)\}. \tag{29}$$

Proof First, we show that $pre_{[\Delta_i,\omega]}(\downarrow\{R_j\})$ is downward closed. Second, we show that $S = cpre_{[\Delta_{i-1}^\sharp,\omega]}\left(R_j\right)$ is in $pre_{[\Delta_i,\omega]}(\downarrow\{R_j\})$. Finally, we show that every element T in $pre_{[\Delta_i,\omega]}(\downarrow\{R_j\})$ is smaller than or equal to S.

1. Proving that $pre[\Delta_i,\omega](\downarrow\{R_j\})$ is downward closed: Consider a state $S' \in Q_i$ s.t. $S' \in pre[\Delta_i,\omega](\downarrow\{R_j\})$. From the definitions of *pre* and Δ_i, it holds that
$$post[\Delta_i,\omega](\{S'\}) = \{post[\Delta^\sharp_{i-1},\omega](S')\} \subseteq \downarrow\{R_j\}, \tag{30}$$
(note that Δ_i is deterministic) and, therefore, $post[\Delta^\sharp_{i-1},\omega](S') \in \downarrow\{R_j\}$. For $T \subseteq S'$, it clearly holds that
$$post[\Delta^\sharp_{i-1},\omega](T) \subseteq post[\Delta^\sharp_{i-1},\omega](S') \tag{31}$$
and so it also holds that
$$post[\Delta_i,\omega](\{T\}) = \{post[\Delta^\sharp_{i-1},\omega](T)\} \subseteq \downarrow\{R_j\}. \tag{32}$$
Therefore, $T \in pre[\Delta_i,\omega](\downarrow\{R_j\})$ and $pre[\Delta_i,\omega](\downarrow\{R_j\})$ is downward closed.
2. Proving that $S = cpre[\Delta^\sharp_{i-1},\omega]\left(R_j\right) \in pre[\Delta_i,\omega](\downarrow\{R_j\})$: From the definition of *cpre*, it holds that
$$post[\Delta^\sharp_{i-1},\omega](S) = S' \subseteq R_j. \tag{33}$$
Further, from the definition of Δ_i, it holds that $S \xrightarrow{\omega} S' \in \Delta_i$ and, therefore, $S \in pre[\Delta_i,\omega](\downarrow\{R_j\})$.
3. Proving that for every $T \in pre[\Delta_i,\omega](\downarrow\{R_j\})$ it holds that $T \subseteq S$: From $T \in pre[\Delta_i,\omega](\downarrow\{R_j\})$, we have that $T \xrightarrow{\omega} P \in \Delta_i$ for $P \subseteq R_j$, and, from the definition of Δ_i, we have that $P = post[\Delta^\sharp_{i-1},\omega](T)$. From $P = post[\Delta^\sharp_{i-1},\omega](T)$ and the definition of *cpre*, it is easy to see that $T \subseteq cpre[\Delta^\sharp_{i-1},\omega]\,(P)$, and, moreover,
$$P \subseteq R_j \quad \Longrightarrow \quad cpre[\Delta^\sharp_{i-1},\omega]\,(P) \subseteq cpre[\Delta^\sharp_{i-1},\omega]\left(R_j\right). \tag{34}$$
Therefore, we can conclude that $T \subseteq cpre[\Delta^\sharp_{i-1},\omega]\left(R_j\right) = S$. □

Intuitively, the sets with the *post*-images below R_j in the ordering $\subseteq$ are those that do not have an outgoing transition leading outside R_j. The largest such a set is $cpre[\Delta^\sharp_{i-1},\omega]\left(R_j\right)$. Combining (28) with Lemma 4 yields
$$pre[\Delta^\sharp_i,\tau](Z) = \bigcup_{\substack{R_j \in \mathcal{R} \\ \omega \in \pi^{-1}_{i+1}(\tau)}} \downarrow\{cpre[\Delta^\sharp_{i-1},\omega]\left(R_j\right)\}. \tag{35}$$
Using (25), we get the final formula for $pre[\Delta^\sharp_i,\tau](Z)$:
$$pre[\Delta^\sharp_i,\tau](Z = \downarrow\mathcal{R}) \;=\; \downarrow\{cpre[\Delta^\sharp_{i-1},\omega]\left(R_j\right) \mid \omega \in \pi^{-1}_{i+1}(\tau),\, R_j \in \mathcal{R}\}. \tag{36}$$
To compute $f_{F^\sharp_i}(Z)$, it remains to unite $pre[\Delta^\sharp_i,\bar{0}](Z)$, computed using (36), with F_i. From Eq. 9(i), F_i equals $\downarrow\{N^\sharp_{i-1}\}$, so the union can be done symbolically as
$$f_{F^\sharp_i}(Z) = \downarrow\left(\{N^\sharp_{i-1}\} \cup \left\{cpre[\Delta^\sharp_{i-1},\omega]\left(R_j\right) \mid \omega \in \pi^{-1}_{i+1}(\bar{0}),\, R_j \in \mathcal{R}\right\}\right). \tag{37}$$
Therefore, a symbolic application of $f_{F^\sharp_i}$ to $Z = \downarrow\mathcal{R}$ represented by the set $\mathcal{R}$ reduces to computing *cpre*-images of elements of $\mathcal{R}$, which are put next to each other, together with $N^\sharp_{i-1}$. The computation starts from $F_i = \downarrow\{N^\sharp_{i-1}\}$, represented by $\{N^\sharp_{i-1}\}$, and each of its steps, implemented by (37), preserves the form of sets $\downarrow\mathcal{R}$, which are represented by $\mathcal{R}$.

5.4 Computation of $F_i^\sharp$ and $N_i^\sharp$ on symbolic terms

Sections 5.2 and 5.3 show how sets of states arising within the fixpoint computations from Eqs. 9(ii) and 9(iv) can be represented symbolically using representatives that are sets of states of the lower level. The sets of states of the lower level will be again represented symbolically. When computing the fixpoint of level i, we will work with a nested symbolic representation of states of depth i. Particularly, sets of states of Q_k, for $0 \leq k \leq i$, are represented by *terms of level k* where a term of level 0 is a subset of Q_0, a term of level $2j+1$, for $j \geq 0$, is of the form $\uparrow\coprod\{t_1, \ldots, t_n\}$ where $t_1, \ldots, t_n$ are terms of level $2j$, and a term of level $2j$, for $j > 0$, is of the form $\downarrow\{t_1, \ldots, t_n\}$ where $t_1, \ldots, t_n$ are terms of level $2j-1$.

The computation of *cpre* and $f_{N_{2j+1}^\sharp}$ on a term of level $2j+1$ and computation of *pre* and $f_{F_{2j}^\sharp}$ on a term of level $2j$ then becomes a recursive procedure that descends via the structure of the terms and produces again a term of level $2j+1$ or $2j$ respectively. In the case of *cpre* and $f_{N_{2j+1}^\sharp}$ called on a term of level $2j+1$, Eq. (22) reduces the computation to a computation of *pre* on its sub-terms of level $2j$, which is again reduced by (36) to a computation of *cpre* on terms of level $2j-1$, and so on until the bottom level where the algorithm computes *pre* on the terms of level 0 (subsets of Q_0). The case of *pre* and $f_{F_{2j}^\sharp}$ called on a term of level $2j$ is symmetrical.

Example 1 We will demonstrate the run of our algorithm on the following example formula:

$$\varphi \equiv \underbrace{\neg\exists X\neg\exists Y\underbrace{\neg\underbrace{\exists Z : \underbrace{X < Y \wedge Y < Z}_{\varphi_0}}_{\varphi_0^\sharp}}_{\varphi_1 \quad \iddots}}_{\varphi_3}$$

Note that we extend the minimal syntax introduced in Sect. 2 with two additional atomic predicates and one additional logical connective (added to easily obtain automata suitable for the demonstration of our algorithm). The semantics of the atomic formula $X < Y$ is defined as

$$\begin{aligned} X < Y \quad \equiv \quad & \Big(\exists x \in X : \forall y \in Y : \exists W : \\ & \quad (\exists u \in W : y = u + 1) \wedge \\ & \quad \big(\forall w \in W : (\exists w' \in W : w = w' + 1) \vee w = x\big)\Big) \\ & \wedge \exists y' \in Y : \mathit{true}, \end{aligned} \tag{38}$$

where we use first-order variable quantification in the standard meaning. Informally, $X < Y$ denotes that both X and Y are non-empty and that the least element of X is strictly smaller than every element of Y.

We build the base automaton $\mathcal{A}_{\varphi_0}$ corresponding to the base formula $\varphi_0 \equiv X < Y \wedge Y < Z$ by (i) cylindrification of the atomic automata $\mathcal{A}_{X<Y}$ and $\mathcal{A}_{Y<Z}$ depicted in Fig. 1a, b, respectively, and by (ii) constructing the intersection automaton $\mathcal{A}_0 = \mathcal{A}_{X<Y} \cap \mathcal{A}_{Y<Z}$. The minimal non-deterministic automaton $\mathcal{A}_0$ is depicted in Fig. 2. The symbol ? denotes that the value on the given track can contain both 0 or 1.

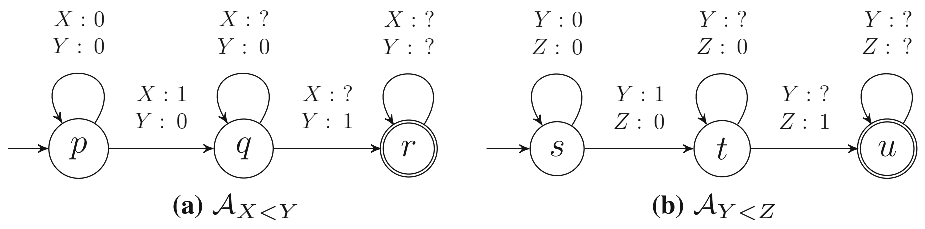

Fig. 1 Atomic automata $\mathcal{A}_{X<Y}$ and $\mathcal{A}_{Y<Z}$

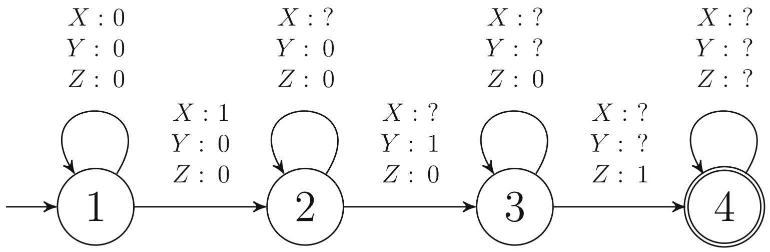

Fig. 2 Automaton $\mathcal{A}_0$ for the formula $\varphi_0 \equiv X < Y \wedge Y < Z$

Recall that our method decides validity of φ by computing symbolically the sequence of sets $F_0^\sharp$, N_1, $N_1^\sharp$, F_2, $F_2^\sharp$, N_3, corresponding to the sequence of automata $\mathcal{A}_{\varphi_0^\sharp}$, $\mathcal{A}_{\varphi_1}$, $\mathcal{A}_{\varphi_1^\sharp}$, $\mathcal{A}_{\varphi_2}$, $\mathcal{A}_{\varphi_2^\sharp}$, $\mathcal{A}_{\varphi_3}$, with each of the sets represented using a symbolic term, and then finally checks whether $I_3 \cap N_3 \neq \emptyset$.

Let us show how the sequence is computed. Once we have constructed the base automaton, we first process the existential quantification of the variable Z, i.e. the subformula $\varphi_0^\sharp \equiv \exists Z : \varphi_0$. The first set in the sequence, $F_0^\sharp$, is obtained using a fixpoint computation given by Eq. 9(ii), that is,

$$F_0^\sharp = \mu W \,.\, F_0 \cup pre_{[\Delta_0^\sharp, \bar{0}]}(W).$$

This computation returns the set of states backward-reachable from F_0 via $\bar{0}$ transitions of $\Delta_0^\sharp$. Here, the zero symbol $\bar{0}$ corresponds to the mapping $\substack{X\,:\,0\\Y\,:\,0}$ of the free variables of the subformula $\varphi_0^\sharp$. The set F_0 of states of the base automaton $\mathcal{A}_0$, from which the computation starts, equals $\{4\}$. Since we are processing $\exists Z$ in the formula, the transition relation $\Delta_0^\sharp$ can be obtained by removing the track corresponding to the variable Z from Δ_0. For instance, from the transition $1 \xrightarrow{\substack{X\,:\,1\\Y\,:\,0\\Z\,:\,0}} 2$, we obtain $1 \xrightarrow{\substack{X\,:\,1\\Y\,:\,0}} 2$. However, according to (28), instead of removing the track, our algorithm rather computes the predecessors on the original transition relation Δ_0 according to symbols where the value in the concerned Z-track is arbitrary. The set of such symbols is obtained using the inverse operation of projection. In particular, the inverse operation of projection $\pi_{[Z]}^{-1}(\substack{X\,:\,0\\Y\,:\,0})$, which is used in the fixpoint computation of $F_0^\sharp$, equals the set $\left\{\substack{X\,:\,0\\Y\,:\,0\\Z\,:\,0}, \substack{X\,:\,0\\Y\,:\,0\\Z\,:\,1}\right\}$. The fixpoint computation is then carried out as follows:

$$\begin{aligned}
F_0^\sharp &= F_0 \cup pre\Big[\Delta_0^\sharp, \begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\Big](F_0) \cup pre^2\Big[\Delta_0^\sharp, \begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\Big](F_0) \cup \cdots \\
&= F_0 \cup \Bigg(\bigcup_{\substack{q\in F_0=\{4\}\\ \omega\in\pi^{-1}_{[Z]}(\begin{smallmatrix}X:0\\Y:0\end{smallmatrix})}} pre[\Delta_0,\omega](q)\Bigg) \cup \cdots && \text{[by (28)]}\\
&= F_0 \cup \left(pre\left[\Delta_0, \begin{smallmatrix}X:0\\Y:0\\Z:0\end{smallmatrix}\right](4) \cup pre\left[\Delta_0, \begin{smallmatrix}X:0\\Y:0\\Z:1\end{smallmatrix}\right](4)\right) \cup \cdots \\
&= \{4\} \cup (\{3,4\} \cup \{4\}) \cup \cdots
\end{aligned}$$

After two iterations, the fixpoint is fully computed, yielding the term

$$t_{[F_0^\sharp]} = F_0^\sharp = \{3,4\}.$$

Next, we have to process the negation in the subformula $\varphi_1 \equiv \neg\exists Z : \varphi_0$, which leads to computation of the term $t_{[N_1]}$ using Eq. 9(iii), yielding the term

$$t_{[N_1]} = \uparrow\coprod\{F_0^\sharp\} = \uparrow\coprod\big\{\{3,4\}\big\}.$$

The algorithm continues by computing the term for the set of states $N_1^\sharp$, corresponding to the subformula $\varphi_1^\sharp \equiv \exists Y : \varphi_1$, which implies a need to process another quantifier level (namely, that of variable Y). Similarly to the previous computation, the transition relation $\Delta_1^\sharp$ can be obtained by removing the track corresponding to the variable Y. This means that the fixpoint computation needs to compute *cpre* with the symbol $\overline{0}$ that now corresponds to the symbol $X:0$. Instead of that, however, a computation over Δ_1 with symbols with arbitrary values of Y will be used. In particular, the set of such symbols will be obtained by Eq. (14) using the inverse projection of Y, which yields the set $\pi^{-1}_{[Y]}(X:0) = \{\begin{smallmatrix}X:0\\Y:0\end{smallmatrix}, \begin{smallmatrix}X:0\\Y:1\end{smallmatrix}\}$. More concretely, the computation of $N_1^\sharp$ is performed according to Eq. 9(iv) as follows:

$$N_1^\sharp = \nu W \,.\, N_1 \cap cpre_{[\Delta_1^\sharp,\overline{0}]}\,(W)\,.$$

To compute the above, Eq. (23) is used to transform the problem of computing the $cpre_{[\Delta_1,\omega']}$-image of a term into a computation of a series of $pre_{[\Delta_0^\sharp,\omega]}$-images of its sub-terms in the same way as Eq. (37) is used when computing $t_{[F_0^\sharp]}$, resulting in the following fixpoint computation:

$$\begin{aligned}
N_1^\sharp &= N_1 \cap cpre\Big[\Delta_1^\sharp, X:0\Big](N_1) \cap cpre^2\Big[\Delta_1^\sharp, X:0\Big](N_1) \cap \cdots \\
&= N_1 \cap \Bigg(\bigcap_{\substack{Q\in N_1\\ \omega\in\pi^{-1}_{[Y]}(X:0)}} cpre[\Delta_1,\omega](Q)\Bigg) \cap \cdots && \text{[by (14)]}\\
&= N_1 \cap \Big(cpre\Big[\Delta_1, \begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\Big](\uparrow\coprod\{\{3,4\}\}) \cap cpre\Big[\Delta_1, \begin{smallmatrix}X:0\\Y:1\end{smallmatrix}\Big](\uparrow\coprod\{\{3,4\}\})\Big) \cap \cdots \\
&= N_1 \cap \Big(\uparrow\coprod\Big\{pre\Big[\Delta_0^\sharp, \begin{smallmatrix}X:0\\Y:0\end{smallmatrix}\Big](\{3,4\})\Big\} \cap \uparrow\coprod\Big\{pre\Big[\Delta_0^\sharp, \begin{smallmatrix}X:0\\Y:1\end{smallmatrix}\Big](\{3,4\})\Big\}\Big) \cap \cdots && \text{[by Lemma 3]}\\
&= N_1 \cap \Bigg(\uparrow\coprod\Bigg\{ \bigcup_{\substack{q\in\{3,4\}\\ \omega\in\pi^{-1}_{[Z]}(\begin{smallmatrix}X:0\\Y:0\end{smallmatrix})}} pre[\Delta_0,\omega](q)\Bigg\} \cap \uparrow\coprod\Bigg\{ \bigcup_{\substack{q\in\{3,4\}\\ \omega\in\pi^{-1}_{[Z]}(\begin{smallmatrix}X:0\\Y:1\end{smallmatrix})}} pre[\Delta_0,\omega](q)\Bigg\}\Bigg) \cap \cdots && \text{[by (28)]}\\
&= N_1 \cap \Bigg(\uparrow\coprod\Bigg\{ pre\left[\Delta_0, \begin{smallmatrix}X:0\\Y:0\\Z:0\end{smallmatrix}\right](3) \cup pre\left[\Delta_0, \begin{smallmatrix}X:0\\Y:0\\Z:1\end{smallmatrix}\right](3) \cup
\end{aligned}$$

$$
\begin{aligned}
&\quad pre\left[\Delta_0, \begin{smallmatrix} X:0\\ Y:0\\ Z:0\end{smallmatrix}\right](4) \cup pre\left[\Delta_0, \begin{smallmatrix} X:0\\ Y:0\\ Z:1\end{smallmatrix}\right](4)\Big\} \cap \\
&\quad \uparrow\coprod\Big\{pre\left[\Delta_0, \begin{smallmatrix} X:0\\ Y:1\\ Z:0\end{smallmatrix}\right](3) \cup pre\left[\Delta_0, \begin{smallmatrix} X:0\\ Y:1\\ Z:1\end{smallmatrix}\right](3) \cup \\
&\quad pre\left[\Delta_0, \begin{smallmatrix} X:0\\ Y:1\\ Z:0\end{smallmatrix}\right](4) \cup pre\left[\Delta_0, \begin{smallmatrix} X:0\\ Y:1\\ Z:1\end{smallmatrix}\right](4)\Big\}\Big) \cap \cdots \\
&= \uparrow\coprod\big\{\{3,4\}\big\} \cap \big(\uparrow\coprod\big\{\{3,4\}\big\} \cap \uparrow\coprod\big\{\{2,3,4\}\big\}\big) \cap \cdots \\
&= \uparrow\coprod\big\{\{3,4\}\big\} \cap \big(\uparrow\coprod\big\{\{3,4\} \cup \{2,3,4\}\big\}\big) \cap \cdots && \text{[by Lemma 1]} \\
&= \uparrow\coprod\big\{\{3,4\}\big\} \cap \uparrow\coprod\big\{\{2,3,4\}\big\} \cap \cdots
\end{aligned}
$$

Note that we do not have to compute explictly the term $pre\left[\Delta_0^\sharp, \begin{smallmatrix} X:0\\ Y:0\end{smallmatrix}\right](\{3,4\})$ as it was computed in the previous iteration of the algorithm, and thus we can use caching of intermediate results to obtain an even more efficient decision procedure. We end up with the term

$$t[N_1^\sharp] = \uparrow\coprod\big\{\{3,4\}, \{2,3,4\}\big\}.$$

We continue with processing of the second negation by computing the term corresponding to the set F_2 of automaton A_{φ_2} for the subformula $\varphi_2 \equiv \neg\exists Y : \varphi_1$ using Eq. 9(i) to obtain the term

$$t[F_2] = \downarrow\{N_1^\sharp\} = \downarrow\Big\{\uparrow\coprod\big\{\{3,4\}, \{2,3,4\}\big\}\Big\}.$$

Next, we process the last quantifier corresponding to the formula $\varphi_2^\sharp \equiv \exists X : \varphi_2$. The symbolic fixpoint computation of $F_2^\sharp$ from Eq. 9(ii) then starts from F_2 and uses an iterative application of $pre[\Delta_2^\sharp,\bar{0}]$ according to the equation

$$F_2^\sharp = \mu W \,.\, F_2 \cup pre[\Delta_2^\sharp,\bar{0}](W).$$

Note that, since in $\varphi_2^\sharp$, all of the variables are projected away, the zero symbol $\bar{0}$ now corresponds to the mapping $\emptyset$ of the empty set of free variables to the set $\{0,1\}$. The inverse projection of the symbol $\bar{0}$ is then the set $\pi_{[X]}^{-1}(\emptyset) = \{\, X:0\,,\ X:1\,\}$. The fixpoint computation proceedes similarly to the computation of $t[F_0^\sharp]$. Using (36), we transform the computation of the image of $pre[\Delta_2^\sharp,\omega'']$ into the computation of a series of $cpre[\Delta_1^\sharp,\omega']$-images of the sub-terms of $t[N_1^\sharp]$. These are in turn transformed by (22) into a computation of a series of $pre[\Delta_0^\sharp,\omega]$-images of sub-sub-terms of $t[F_0^\sharp]$, i.e. subsets of Q_0. For our example, this yields a fixpoint computation analogous to the previous computation of the $t[F_0^\sharp]$, resulting in the term

$$t[F_2^\sharp] = \downarrow\Big\{\uparrow\coprod\big\{\{3,4\}, \{2,3,4\}\big\}, \uparrow\coprod\big\{\{3,4\}, \{2,3,4\}, \{1,2,3,4\}\big\}\Big\}.$$

Finally, using Eq. 9(iii), we process the last negation corresponding to the formula $\varphi \equiv \varphi_3 \equiv \neg\exists X : \varphi_2$, which yields the final term representing N_3, namely,

$$t[N_3] = \uparrow\coprod\Big\{\downarrow\Big\{\uparrow\coprod\big\{\{3,4\}, \{2,3,4\}\big\}, \uparrow\coprod\big\{\{3,4\}, \{2,3,4\}, \{1,2,3,4\}\big\}\Big\}\Big\}.$$

Now, it remains to check whether $I_3 \cap F_3 \neq \emptyset$ using the computed term $t[N_3]$. We will show how to evaluate this intersection in the next section. □

5.5 Testing $I_m \cap F_m \overset{?}{\neq} \emptyset$ on symbolic terms

Due to the special form of the set I_m (every I_i, where $1 \leq i \leq m$, is the singleton set $\{I_{i-1}\}$, cf. Sect. 5.1), the test $I_m \cap F_m \neq \emptyset$ can be done efficiently over the symbolic terms representing F_m. Since $I_m = \{I_{m-1}\}$ is a singleton set, testing $I_m \cap F_m \neq \emptyset$ is equivalent to testing $I_{m-1} \in F_m$. If m is odd, our approach computes the symbolic representation of N_m instead of F_m. Obviously, since N_m is the complement of F_m, it holds that $I_{m-1} \in F_m \iff I_{m-1} \notin N_m$. Our way of testing $I_{m-1} \in \downarrow\mathbb{S}$ on a symbolic representation of the set $\downarrow\mathbb{S}$ of level m is based on the following equations:

$$\{q\} \in \downarrow\mathbb{S} \iff \exists S \in \mathbb{S} : q \in S \tag{44}$$

$$\{q\} \in \uparrow\coprod\mathbb{S} \iff \forall S \in \mathbb{S} : q \in S \tag{45}$$

and, for $i = 0$,

$$I_0 \in \uparrow\coprod\mathbb{S} \iff \forall S \in \mathbb{S} : I_0 \cap S \neq \emptyset. \tag{46}$$

Given a symbolic term $t[R]_m$ of level m representing a set $R_m \subseteq Q_m$, testing emptiness of $I_m \cap R_m$ or $I_m \subseteq R_m$ can be done over $t[R_m]$ by a recursive procedure that descends along the structure of $t[R_m]$ using (44) and (45), essentially generating an AND-OR tree, terminating the descent by an application of (46).

Example 2 To finish Example 1, we need to test whether $I_3 \cap F_3 = \emptyset$. This is equivalent to checking whether $I_3 \subseteq N_3$, i.e., whether $\{\{\{\{1\}\}\}\} \subseteq N_3$, which holds iff $I_2 = \{\{\{1\}\}\} \in N_3$, using $t[N_3] = \uparrow\coprod\{F_2^\sharp\}$ to represent N_3. From (45), we get that

$$I_2 = \{\{\{1\}\}\} \in \uparrow\coprod\{F_2^\sharp\} \iff I_1 = \{\{1\}\} \in F_2^\sharp$$

because $F_2^\sharp$ is the denotation of the only sub-term $t[F_2^\sharp]$ of $t[N_3]$. Equation (44) establishes that

$$\begin{aligned} &I_1 = \{\{1\}\} \in F_2^\sharp \iff \\ &\quad \{1\} \in \uparrow\coprod\big\{\{3,4\},\{2,3,4\}\big\} \;\vee\; \{1\} \in \uparrow\coprod\big\{\{3,4\},\{2,3,4\},\{1,2,3,4\}\big\}. \end{aligned}$$

Each of the disjuncts can then be further reduced by (45) into a conjunction of membership queries on the base level, which is solved using (46) as follows:

$$\begin{aligned} &I_1 = \{\{1\}\} \in F_2^\sharp \iff \\ &\quad (1 \in \{3,4\} \wedge 1 \in \{2,3,4\}) \;\vee\; (1 \in \{3,4\} \wedge 1 \in \{2,3,4\} \wedge 1 \in \{1,2,3,4\}) \end{aligned}$$

Since none of the disjuncts is satisfied, we have that $I_1 \notin F_2^\sharp$, so $I_2 \notin N_3$, implying that $I_2 \in F_3$. We conclude that $I_3 \subseteq N_3$ and hence $\models \varphi$. □

5.6 Subsumption of symbolic terms

Although the use of symbolic terms instead of an explicit enumeration of sets of states itself considerably reduces the searched space, an even greater degree of reduction can be obtained using subsumption inside the symbolic representatives to reduce their size, similarly as in the antichain algorithms [14]. For any set of sets $\mathbb{S}$ containing a pair of distinct elements $R, T \in \mathbb{S}$ s.t. $R \subseteq T$, the following holds:

$$\downarrow\mathbb{S} = \downarrow(\mathbb{S}\backslash\{R\}) \quad \text{and} \quad \uparrow\coprod\mathbb{S} = \uparrow\coprod(\mathbb{S}\backslash\{T\}). \tag{47}$$

Therefore, if $\mathbb{S}$ is used to represent the set $\downarrow\mathbb{S}$, the element R is *subsumed* by T and can be removed from $\mathbb{S}$ without changing its denotation. Likewise, if $\mathbb{S}$ is used to represent $\uparrow\coprod\mathbb{S}$, the element T is *subsumed* by R and can be removed from $\mathbb{S}$ without changing its denotation.

We can thus simplify any symbolic term by pruning out its sub-terms that represent elements subsumed by elements represented by other sub-terms, without changing the denotation of the term.

Computing subsumption on terms can be done using the following two equations:

$$\downarrow\mathbb{R} \subseteq \downarrow\mathbb{S} \iff \forall R \in \mathbb{R} : \exists S \in \mathbb{S} : R \subseteq S \tag{48}$$

$$\uparrow\coprod\mathbb{R} \subseteq \uparrow\coprod\mathbb{S} \iff \forall S \in \mathbb{S} : \exists R \in \mathbb{R} : R \subseteq S. \tag{49}$$

Using (48) and (49), testing subsumption of terms of level i reduces to testing subsumption of terms of level $i-1$. The procedure for testing subsumption of two terms descends along the structure of the term, using (48) and (49) on levels greater than 0, and on level 0, where terms are subsets of Q_0, it tests subsumption by set inclusion.

Example 3 In Example 1, we can use the inclusions of $\{3,4\} \subseteq \{2,3,4\} \subseteq \{1,2,3,4\}$ and (47) to reduce $t[N_1^\sharp] = \uparrow\coprod\big\{\{3,4\},\{2,3,4\}\big\}$ and the intermediate term $t = \uparrow\coprod\big\{\{3,4\},\{2,3,4\},\{1,2,3,4\}\big\}$ to the terms

$$t[N_1^\sharp]' = \uparrow\coprod\big\{\{2,3,4\}\big\} \text{ and}$$
$$t' = \uparrow\coprod\big\{\{1,2,3,4\}\big\} \text{ respectively.}$$

Moreover, Eq. (49) implies that the term $t' = \uparrow\coprod\big\{\{1,2,3,4\}\big\}$ is subsumed by the term $t[N_1^\sharp]' = \uparrow\coprod\big\{\{2,3,4\}\big\}$, and so we can reduce the term $t[F_2^\sharp] = \downarrow\Big\{\uparrow\coprod\big\{\{2,3,4\}\big\}, \uparrow\coprod\big\{\{1,2,3,4\}\big\}\Big\}$ to the term

$$t[F_2^\sharp]' = \downarrow\Big\{\uparrow\coprod\big\{\{2,3,4\}\big\}\Big\}.$$

□

6 Experimental evaluation

We have implemented a prototype of the presented approach in the tool dWiNA [27]. It uses the frontend of Mona to parse the input formula, and it handles FAs encoded using the MTBDD-based representation from the `libvata` library [28]. It has two modes of operation. In Mode I, we use Mona to generate the minimal deterministic automaton $\mathcal{A}_{\varphi_0}$ corresponding to the matrix of the tested formula. Since the input formula may not be in the prenex normal form (i.e., a prefix of quantifiers followed by a quantifier-free matrix), the matrix here corresponds to the subformula under the topmost quantifier, or, if there is no single top-most quantifier, to the entire formula. The automaton is then translated into the `libvata` format, and our algorithm is run on top of the `libvata`-represented automaton. In Mode II, we first transform the input formula into the prenex normal form where the occurence of negation in the matrix is limited to literals, and then construct a nondeterministic automaton $\mathcal{A}_{\varphi_0}$ for the matrix directly using `libvata`.

We evaluated dWiNA against two classes of benchmarks: formulae arising in verification of pointer programs using the method based on the logic Strand [6], and several parametric families of manually constructed formulae, from which some were originally designed as show cases for other tools. The main focus of our experiment was on comparing dWiNA with Mona, but we carried out some comparison with other available tools too. Namely, we compared with an implementation of the coalgebraic decision procedure [24], which we refer to as Coalg, a decision procedure based on symbolic automata [22], which we refer to

Table 1 Results for formulae obtained from verification tasks of STRAND [6]

Benchmark	Time (s)		Space (states)	
	`Mona`	dWiNA	`Mona`	dWiNA
bubblesort-else	0.01	0.01	1285	19
bubblesort-if-else	0.02	0.23	4260	234
bubblesort-if-if	0.12	1.14	8390	28
sorted-list-insert-after-loop	0.01	0.01	167	36
sorted-list-insert-before-head	0.01	0.01	43	45
sorted-list-insert-before-loop	0.01	0.01	103	47
sorted-list-insert-error-error	0.01	0.01	103	47
sorted-list-insert-in-loop	0.01	0.01	463	59
sorted-list-reverse-after-loop	0.01	0.01	179	110
sorted-list-reverse-before-loop	0.01	0.01	179	110
sorted-list-reverse-in-loop	0.02	0.02	1311	271
sorted-list-search-after-loop	0.01	0.01	90	274
sorted-list-search-before-loop	0.01	0.01	90	274
sorted-list-search-in-loop	0.01	0.02	1311	84

as SFA, and the tool TOSS implementing a procedure based on the Shelah's decomposition [23].

A comparison of dWiNA **with** MONA **on the** STRAND **formulae** Table 1 shows the comparison of dWiNA and MONA against formulae arising in the shape analysis based on the logic STRAND [6]. dWiNA was used in Mode I. We measured the time the tools took for processing the quantifier prefix of the formulae. Overall, dWiNA was comparable and sometimes slightly slower than MONA. We then compared the sum of the numbers of states of all automata generated by MONA when processing the quantifier prefix with the number of symbolic terms generated by dWiNA. The state spaces generated by dWiNA are about one or two orders of magnitude smaller than those generated by MONA. This makes us believe that with enough optimization, dWiNA could become better even time-wise.

An attempt to run dWiNA on this benchmark in Mode II was unsuccessful since `libvata` was not able to construct the matrix automaton in a reasonable time. This is because the construction implemented within `libvata`, which is based on nondeterministic automata, is not optimized. In particular, it uses no automata reduction (whereas deterministic minimization is one of the key features of MONA).

A comparison of dWiNA **with** MONA **on synthetic benchmarks** To demonstrate that our approach can scale significantly better than the explicit automata construction, we created several parameterized families of WS1S formulae. Their basic formulae express relations among subsets of $\mathbb{N}_0$, such as the existence of certain transitive relations, singleton sets, or intervals (their definitions can be found in [27]). From these, we algorithmically generate families of formulae with a larger quantifier depth, regardless of the meaning of the generated formulae (though their semantics is still nontrivial).

Table 2 Results from experiments with the `HornSub` formulae

k	Time (s)		Space (states)	
	MONA	DWINA	MONA	DWINA
2	0.20	0.01	25,517	44
3	0.57	0.01	60,924	50
4	1.79	0.02	145,765	58
5	4.98	0.02	349,314	70
6	∞	0.47	∞	90

In Table 2, we give results that we obtained from experimenting with one of the families, called `HornSub`, where the basic formula expresses existence of an ascending chain of n sets ordered w.r.t. $\subset$ [1]:

$$\exists Y : \neg\exists X_1 \neg \ldots \neg\exists X_k, \ldots, X_n : \bigwedge_{1 \le i < n} \left(X_i \subseteq Y \wedge X_i \subset X_{i+1}\right) \Rightarrow X_{i+1} \subseteq Y.$$

The parameter k stands for the number of alternations in the prefix of the formula. We see that DWINA clearly outperforms MONA. We use ∞ in the case the time exceeded 2 min or when the tool ran out of memory. We carried out these experiments in Mode II of DWINA (the experiment in Mode I was not successful due to a too costly conversion of a large matrix automaton from MONA to `libvata`).

All of the experiments above, targeted to compare the performance of DWINA and MONA only, were carried out on an Intel Core i7-4770@3.4 GHz processor with 32 GiB RAM.

A comparison of DWINA *with other tools.* Our last set of experiments aims at a comparison with other available implementations of WS1S decision procedures, namely TOSS [23], SFA [22], and COALG [24]. Since the tools support a limited set of syntactic features, we could only use a subset of the available benchmark formulae. Namely, we took the parametric families of formulae `HornLeq` from [22] and `HornIn` from [23], originally proposed to evaluate the performance of SFA and TOSS, respectively, and our parametric family of formulae `SetClosed`.[2] The basic formula of the `SetClosed` family expresses the non-existence of an interval set. The parameter n stands for the number of existential quantifications in the prefix of the formula:

$$\exists X_1, \ldots, X_n : \forall x : \neg\forall y, z : \bigwedge_{1 \le i \le n} \Big((x \in X_i \wedge x \le y \wedge y \le z \wedge z \in X_i) \Rightarrow y \in X_i\Big)$$

This experiment had to be evaluated on a different machine with a system that meets the requirements of all the tools, with an Intel Core i7-4770@3.4 GHz processor and 16GiB RAM, running Debian GNU/Linux. Table 3 gives the run times of the tools. We use ∞ in case the time exceeded 2 min and `oom` to denote that the tool ran out of memory. While TOSS performs best on their own benchmarks, DWINA outperforms the other tools on the rest of the formulae.

[1] Results for the other families are very similar and hence skipped here. An interested reader is referred to [27].

[2] Note that the `HornSub` family is not supported by TOSS and COALG, and thus we chose a comparably complex family of `SetClosed` to present the overall comparison.

Table 3 Experiments with parametric families of formulae (times are given in seconds)

Benchmark	MONA	TOSS	COALG	SFA	DWINA
`HornLeq` [22]					
horn-leq06	0.01	0.02	1.10	0.01	0.01
horn-leq07	0.01	0.02	11.09	0.01	0.01
horn-leq08	0.01	0.02	101.48	0.01	0.01
horn-leq09	0.01	0.02	∞	0.01	0.01
horn-leq10	0.01	0.03	∞	0.02	0.01
horn-leq11	0.05	0.03	∞	0.02	0.01
horn-leq12	0.09	0.04	∞	0.02	0.01
horn-leq13	0.19	0.04	∞	0.02	0.01
horn-leq14	0.45	0.04	∞	0.02	0.01
horn-leq15	1.19	0.05	∞	0.03	0.02
horn-leq16	3.35	0.05	∞	0.03	0.02
horn-leq17	9.07	0.05	∞	0.03	0.02
horn-leq18	22.89	0.06	∞	0.03	0.02
horn-leq19	`oom`	0.06	∞	0.03	0.03
`HornIn` [23]					
horn-in04	0.01	0.01	0.02	0.27	0.01
horn-in05	0.01	0.01	0.14	0.76	0.03
horn-in06	0.01	0.02	1.07	2.65	0.13
horn-in07	0.01	0.02	8.50	8.31	0.29
horn-in08	0.01	0.02	68.05	32.44	1.16
horn-in09	0.03	0.03	∞	∞	3.42
horn-in10	0.09	0.04	∞	∞	18.40
horn-in11	0.20	0.04	∞	∞	54.74
horn-in12	0.48	0.04	∞	∞	∞
horn-in13	1.20	0.04	∞	∞	∞
horn-in14	2.95	0.05	∞	∞	∞
horn-in15	7.26	0.05	∞	∞	∞
horn-in16	`oom`	0.06	∞	∞	∞
`SetClosed`					
set-closed01	0.01	0.02	0.04	0.01	0.01
set-closed02	0.01	0.02	∞	0.13	0.01
set-closed03	0.01	0.18	∞	0.14	0.01
set-closed04	0.34	∞	∞	13.96	0.01
set-closed05	∞	∞	∞	∞	0.01
set-closed06	∞	∞	∞	∞	0.01
set-closed07	∞	∞	∞	∞	0.01
set-closed08	∞	∞	∞	∞	0.03
set-closed09	∞	∞	∞	∞	0.10
set-closed10	∞	∞	∞	∞	0.27
set-closed11	∞	∞	∞	∞	0.95

Table 3 continued

Benchmark	MONA	TOSS	COALG	SFA	DWINA
set-closed12	∞	∞	∞	∞	3.61
set-closed13	∞	∞	∞	∞	14.30
set-closed14	∞	∞	∞	∞	69.08
set-closed15	∞	∞	∞	∞	∞

7 Conclusion and future work

We presented a new approach for dealing with alternating quantifications within the automata-based decision procedure for WS1S. Our approach is based on a generalization of the idea of the so-called antichain algorithm for testing universality or language inclusion of finite automata. Our approach processes a prefix of the formula with an arbitrary number of quantifier alternations on-the-fly using an efficient symbolic representation of the state space, enhanced with subsumption pruning. Our experimental results are encouraging and show that the direction started in this paper—using modern techniques for nondeterministic automata in the context of deciding WS1S formulae—is promising.

An interesting direction of further development seems to be lifting the symbolic *pre*/*cpre* operators to a more general notion of terms that allow working with general sub-formulae (that may include logical connectives and nested quantifiers). The algorithm could then be run over arbitrary formulae, without the need of the transformation into the prenex form. This would open a way of adopting optimizations used in other tools as well as syntactical optimizations of the input formula such as anti-prenexing. Another way of improvement would be to use simulation-based techniques to reduce the generated automata as well as to weaken the term-subsumption relation (an efficient algorithm for computing simulation over BDD-represented automata is needed). We also plan to extend the algorithms to WS*k*S and tree-automata, and perhaps even further to more general inductive structures.

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

References

1. Fiedor, T., Holík, L., Lengál, O., Vojnar, T.: Nested antichains for WS1S. In: TACAS'15. Volume 9035 of LNCS. Springer, pp. 658–674 (2015)
2. Meyer, A.R.: Weak monadic second order theory of successor is not elementary-recursive. In Parikh, R., (ed.) Proceedings of Logic Colloquium—Symposium on Logic Held at Boston, 1972–1973. Volume 453 of Lecture Notes in Mathematics. Springer, pp. 132–154 (1972)
3. Elgaard, J., Klarlund, N., Møller, A.: MONA 1.x: new techniques for WS1S and WS2S. In: Proceedings of CAV'98. Volume 1427 of Lecture Notes in Computer Science. Springer, pp. 516–520 (1998)
4. Klarlund, N., Møller, A.: MONA Version 1.4 User Manual. BRICS, Department of Computer Science, Aarhus University. Notes Series NS-01-1. http://www.brics.dk/mona/ (2001) . Revision of BRICS NS-98-3
5. Madhusudan, P., Parlato, G., Qiu, X.: Decidable logics combining heap structures and data. In: Proceedings of POPL'11. ACM, pp. 611–622 (2011)
6. Madhusudan, P., Qiu, X.: Efficient decision procedures for heaps using STRAND. In: Proceedings of SAS'11. Volume 6887 of Lecture Notes in Computer Science. Springer, pp. 43–59 (2011)

7. 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, pp. 21–38 (2013)
8. 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), 1006–1036 (2012)
9. Zee, K., Kuncak, V., Rinard, M.C.: Full functional verification of linked data structures. In: Proceedings of POPL'08. ACM, pp. 349–361 (2008)
10. Zhou, M., He, F., Wang, B., Gu, M., Sun, J.: Array theory of bounded elements and its applications. J. Autom. Reason. **52**(4), 379–405 (2014)
11. Hamza, J., Jobstmann, B., Kuncak, V.: Synthesis for regular specifications over unbounded domains. In: Proceedings of FMCAD'10. IEEE, pp. 101–109 (2010)
12. Wies, T., Muñiz, M., Kuncak, V.: An efficient decision procedure for imperative tree data structures. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Proceedings of CADE'11. Volume 6803 of Lecture Notes in Computer Science. Springer, pp. 476–491 (2011)
13. Doyen, L., Raskin, J.F.: Antichain algorithms for finite automata. In: Proceedings of TACAS'10. Volume 6015 of LNCS. Springer, pp. 2–22 (2010)
14. Wulf, M.D., Doyen, L., Henzinger, T.A., Raskin, J.F.: Antichains: a new algorithm for checking universality of finite automata. In: Proceedings of CAV'06. Volume 4144 of LNCS. Springer, pp. 17–30 (2006)
15. Abdulla, P.A., Chen, Y.F., Holík, L., Mayr, R., Vojnar, T.: When simulation meets antichains (on checking language inclusion of nondeterministic finite (tree) automata). In: Esparza, J., Majumdar, R. (eds.) Proceedings of TACAS'10. Volume 6015 of Lecture Notes in Computer Science. Springer, pp. 158–174 (2010)
16. Bustan, D., Grumberg, O.: Simulation based minimization. In: Proceedings of CADE'00. Volume 1831 of Lecture Notes in Computer Science. Springer, pp. 255–270 (2000)
17. Abdulla, P.A., Bouajjani, A., Holík, L., Kaati, L., Vojnar, T.: Computing simulations over tree automata: efficient techniques for reducing tree automata. In: Proceedings of TACAS'08. Volume 4963 of LNCS. Springer, pp. 93–108 (2008)
18. Bouajjani, A., Habermehl, P., Holík, L., Touili, T., Vojnar, T.: Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In: Proceedings of CIAA'08. Volume 5148 of LNCS. Springer, pp. 57–67 (2008)
19. Habermehl, P., Holík, L., Rogalewicz, A., Simácek, J., Vojnar, T.: Forest automata for verification of heap manipulation. Form. Methods Syst. Des. **41**(1), 83–106 (2012)
20. Klarlund, N., Møller, A., Schwartzbach, M.I.: MONA implementation secrets. Int. J. Found. Comput. Sci. **13**(4), 571–586 (2002)
21. Topnik, C., Wilhelm, E., Margaria, T., Steffen, B.: jMosel: A stand-alone tool and jABC plugin for M2L(Str). In: Proceedings of SPIN'06. Volume 3925 of Lecture Notes in Computer Science. Springer, pp. 293–298 (2006)
22. D'Antoni, L., Veanes, M.: Minimization of symbolic automata. In: Proceedings of POPL'14, pp. 541–554 (2014)
23. Ganzow, T., Kaiser, L.: New algorithm for weak monadic second-order logic on inductive structures. In: Proceedings of CSL'10. Volume 6247 of Lecture Notes in Computer Science. Springer, pp. 366–380 (2010)
24. 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, pp. 487–503 (2015)
25. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2008)
26. Büchi, J.R.: Weak second-order arithmetic and finite automata. Technical report, The University of Michigan (1959). http://hdl.handle.net/2027.42/3930 (2010)
27. Fiedor, T., Holík, L., Lengál, O., Vojnar, T.: `dWiNA`. http://www.fit.vutbr.cz/research/groups/verifit/tools/dWiNA/ (2014)
28. Lengál, O., Šimáček, J., Vojnar, T.: VATA: a library for efficient manipulation of non-deterministic tree automata. In: Proceedings of TACAS'12. Volume 7214 of Lecture Notes in Computer Science. Springer, pp. 79–94 (2012)