Nested Antichains for WS1S

FIT BUT Technical Report Series

Tomáš Fiedor, Lukáš Holík,
Ondřej Lengál, and Tomáš Vojnar

Technical Report No. FIT-TR-2014-06
Faculty of Information Technology, Brno University of Technology

Last modified: January 19, 2015

NOTE: This technical report contains an extended version of a paper with the same name accepted to TACAS'15.

Nested Antichains for WS1S

Tomáš Fiedor, Lukáš Holík, Ondřej Lengál, and Tomáš Vojnar

FIT, Brno University of Technology, IT4Innovations Centre of Excellence,
Czech Republic

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 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 [1], it has been shown useful in numerous applications. Most of the successful applications were due to the tool MONA [2], which implements a finite automata-based decision procedure for WS1S and WS2S (a generalization of WS1S to finite binary trees). The authors of MONA list a multitude of its diverse applications [3], 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 [4,5,6,7,8] can be mentioned, as well as synthesis from regular specifications [9]. MONA is still the standard tool and the most common choice when it comes to deciding WS1S/WS2S. There are other related automata-based tools that are more recent, such as jMosel [10] for a logic M2L(Str), and other than automata-based approaches, such as [11]. They implement optimizations that allow to outperform MONA on some benchmarks, however, none provides an evidence of being consistently more efficient. Despite many optimizations implemented in MONA and the 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 [5], or, often restricting the input of their approach, give up translating to WS1S/WS2S altogether [12].

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. However, the worst case exponential complexity of determinization often significantly harms the performance of the tool. Recent works on efficient methods for handling nondeterministic automata suggest a way of alleviating this problem, in particular works on efficient testing of language inclusion and universality of finite automata [13,14,15] and size reduction [16,17] based on a simulation relation. 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 also 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 for $0 \leq i < m$, $\varphi_i^\sharp = \exists X_{i+1} : \varphi_i$, and $\varphi_{i+1} = \neg \varphi_i^\sharp$. 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 we cannot do much better—this non-elementary complexity is an inherent property of the problem.

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 of 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 subsumption pruning which descends recursively down the structure of the terms while pruning on all its levels.

Our nested antichain algorithm can be in its current form used only to process a quantifier prefix of a formula, after which we return the answer to the validity query, but not an automaton representing all models of the input formula. That is, we cannot use the optimized algorithm for processing inner negations and alternating quantifiers which are not a part of the quantifier prefix. However, despite this and the fact that our implementation is far less mature than that of MONA, our experimental results still show significant improvements over its performance, especially in terms of generated state space. We consider this a strong indication that using techniques for nondeterministic automata to decide WS1S (and WSkS) is highly promising. There are many more opportunities of improving the decision procedure based on nondeterministic automata, by using techniques such as simulation relations or bisimulation up-to congruence [20], and applying them to process not only the quantifier prefix, but all logical connectives of a formula. We consider this paper to be the first step towards a decision procedure for WS1S/WSkS with an entirely different scalability than the current state-of-the-art.

Plan of the paper. We define the logic WS1S in Section 2. In Sections 3 and 4, we introduce finite word automata and describe the classical decision procedure for WS1S based on finite word automata. In Section 5, we introduce our method for dealing with alternating quantifiers. Finally, we give an experimental evaluation and conclude the paper in Sections 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 [21].

WS1S is a monadic second-order logic over the universe of discourse $\mathbb{N}_0$. This means that the logic allows second-order *variables*, usually denoted using uppercase letters $X, Y, \dots$, that range over finite subsets of $\mathbb{N}_0$, e.g. $X = \{0, 3, 42\}$. Atomic formulae are of the form (i) $X \subseteq Y$, (ii) $\mathrm{Sing}(X)$, (iii) $X = \{0\}$, and

(iv) $X = Y + 1$, where X and Y are variables. The atomic formulae are interpreted in turn as (i) standard set inclusion, (ii) the singleton predicate, (iii) X is a singleton containing 0, and (iv) $X = \{x\}$ 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 and Finite Automata

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 to 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. 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 $\coprod\mathbb{D} = \emptyset$.

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 $\overline{0}$ to denote the symbol which maps all variables from $\mathbb{X}$ to 0, $\overline{0} \in \Sigma_{\mathbb{X}}$.

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 $\Sigma_{\mathbb{X}} = \{\overline{0}\}$.

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 that $w \in \mathcal{L}_\mathcal{A}(q)$ 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

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

The *complement* of $\mathcal{A}$ is the automaton $\mathcal{A}_\mathcal{C} = (2^Q, \Delta_\mathcal{C}, \{I\}, \downarrow\{Q \setminus F\})$ where $\Delta_\mathcal{C} = \left\{P \xrightarrow{\tau} \mathit{post}_{[\Delta,\tau]}(P) \;\middle|\; P \subseteq Q\right\}$; 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 [22] (as described in Section 3.3 of [21]) 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 subformula 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. $\begin{smallmatrix}X_1:\epsilon\\X_2:\epsilon\end{smallmatrix}$, $\begin{smallmatrix}X_1:011\\X_2:101\end{smallmatrix}$, or $\begin{smallmatrix}X_1:01100\\X_2:10100\end{smallmatrix}$. We denote the i-th 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 j-th 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, $\begin{smallmatrix}X_1:0\\X_2:1\end{smallmatrix}$, $\begin{smallmatrix}X_1:00\\X_2:10\end{smallmatrix}$, $\begin{smallmatrix}X_1:000\\X_2:100\end{smallmatrix}$, $\begin{smallmatrix}X_1:000\ldots0\\X_2:100\ldots0\end{smallmatrix}$ are all encodings of the assignment $\rho = \{X_1 \mapsto \emptyset, X_2 \mapsto \{0\}\}$. 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 [21] 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 the words $\begin{smallmatrix}X_1:010\\X_2:001\end{smallmatrix}$, $\begin{smallmatrix}X_1:0100\\X_2:0010\end{smallmatrix}$, $\begin{smallmatrix}X_1:0100\ldots0\\X_2:0010\ldots0\end{smallmatrix}$ and we are computing the FA for $\exists X_2 : \varphi$. When we remove the X_2 row from all symbols, we obtain the FA $\mathcal{A}'_\varphi$ that accepts the words $X_1:010$, $X_1:0100$, $X_1:0100\ldots0$, but does not accept the word $X_1:01$ that encodes the same assignment (because $\begin{smallmatrix}X_1:01\\X_2:??\end{smallmatrix} \notin \mathcal{L}(\mathcal{A}_\varphi)$ for any values in the places of "?"s). As a remedy for this situation, we 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 $\overline{0}$ symbols. Formally, $\mathcal{A}_{\exists X:\varphi} = (Q, \pi_{[X]}(\Delta), I, F^\sharp)$ is obtained from $\mathcal{A}'_\varphi = (Q, \pi_{[X]}(\Delta), I, F)$ by computing $F^\sharp$ from F using the fixpoint computation $F^\sharp = \mu Z \,.\, F \cup \mathit{pre}_{[\pi_{[X]}(\Delta),\overline{0}]}(Z)$. Intuitively, the least fixpoint denotes the set of states backward-reachable from F following transitions of $\pi_{[X]}(\Delta)$ labelled by $\overline{0}$.

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

$$\varphi = \neg\, \underbrace{\exists \mathcal{X}_m \neg \ldots \neg \exists \mathcal{X}_2 \underbrace{\neg \exists \mathcal{X}_1 : \varphi_0(\mathbb{X})}_{\varphi_1} \quad \iddots}_{\varphi_m} \tag{2}$$

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 Section 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 $0 \leq i \leq m-1$, $\varphi_i^\sharp = \exists \mathcal{X}_{i+1} : \varphi_i$ and $\varphi_{i+1} = \neg \varphi_i^\sharp$. The φ_i's are the subformulae of φ shown in Equation 2. 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 $\mathcal{A}_{\varphi^\sharp_{i-1}}$, and to compute $\mathcal{A}_{\varphi^\sharp_i}$, it is necessary to compute the set of final states $F^\sharp_i$. 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 $\overline{0}$ by π_{i+1}). To compute $F^\sharp_i$, the antichain algorithms could be actually taken off-the-shelf and run with $\mathcal{A}_{\varphi^\sharp_{i-1}}$ in the role of the input $\mathcal{A}$ and $\mathcal{A}_{\varphi^\sharp_i}$ in the role of $\overline{\mathcal{A}}$. However, this approach has the following two problems. First, antichain algorithms do not produce the automaton $\overline{\mathcal{A}}$ (here $\mathcal{A}_{\varphi^\sharp_i}$), but only a symbolic representation of a set of (backward) reachable states (here of $F^\sharp_i$). Since $\mathcal{A}_{\varphi^\sharp_i}$ 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 $\mathcal{A}_{\varphi^\sharp_{i-1}}$ 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 our on-the-fly algorithm for handling quantifier alternation. Following the construction of automata described in Section 4, the structure of the automata from the previous section, $\mathcal{A}_{\varphi_0}, \mathcal{A}_{\varphi^\sharp_0}, \ldots, \mathcal{A}_{\varphi^\sharp_{m-1}}, \mathcal{A}_{\varphi_m}$, can be described using the following recursive definition. We use $\pi_i(C)$ for any mathematical structure 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 $0 \leq i < m$, $\mathcal{A}_{\varphi^\sharp_i}$ and $\mathcal{A}_{\varphi_{i+1}}$ are FAs over $\pi_{i+1}(\mathbb{X})$ that have from the construction the following structure:

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

We recall that $\mathcal{A}_{\varphi^\sharp_i}$ directly corresponds to existential quantification of the variable X_i (cf. Section 4), and $\mathcal{A}_{\varphi_{i+1}}$ directly corresponds to the complement of $\mathcal{A}_{\varphi^\sharp_i}$ (cf. Section 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 $\{\overline{0}\}^*$ (as described in Section 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

$$\models \varphi \quad \text{iff} \quad I_m \cap F_m \neq \emptyset. \tag{3}$$

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 (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 $\mathcal{A}_{\varphi_0}, \mathcal{A}_{\varphi_0^\sharp}, \ldots, \mathcal{A}_{\varphi_{m-1}^\sharp}, \mathcal{A}_{\varphi_m}$ 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:

$$\begin{aligned} &\text{(i) } F_{i+1} = \downarrow\{N_i^\sharp\}, && \text{(ii) } F_i^\sharp = \mu Z \,.\, F_i \cup pre_{[\Delta_i^\sharp,\overline{0}]}(Z), \\ &\text{(iii) } N_{i+1} = \uparrow\coprod\{F_i^\sharp\}, && \text{(iv) } N_i^\sharp = \nu Z \,.\, N_i \cap cpre_{[\Delta_i^\sharp,\overline{0}]}(Z). \end{aligned} \tag{4}$$

Intuitively, Equations (i) and (ii) are directly from the definition of $\mathcal{A}_i$ and $\mathcal{A}_i^\sharp$. Equation (iii) is a dual of Equation (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, Equation (iv) is a dual of Equation (ii): in the k-th iteration of the greatest fixpoint computation, the current set of states Z will contain all states that cannot reach an F_i state over $\overline{0}$ within k steps. In the next iteration, only those states of Z are kept such that all their $\overline{0}$-successors are in Z. Hence, the new value of Z is the set of states that cannot reach F_i over $\overline{0}$ in $k+1$ steps, and the computation stabilises with the set of states that cannot reach F_i over $\overline{0}$ 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/downward closed sets. Particularly, in Sections 5.2 and 5.3, we show how the fixpoints from Equations (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 Section 5.6 describes the subsumption relation used to minimize the symbolic term representation used within computations of Equations (ii) and (iv). Proofs of the lemmas and used equations can be found in Appendix A.

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 Equation 4(iv):

$$f_{N_i^\sharp}(Z) = N_i \cap \mathit{cpre}_{[\Delta_i^\sharp,\bar{0}]}(Z). \tag{5}$$

We will show that the whole fixpoint computation from Equation 4(iv) can be carried out symbolically on the representatives of Z. We will explain that: (a) All intermediate values of Z have the form $\uparrow\coprod\mathcal{R}$, $\mathcal{R} \subseteq Q_i$, so the sets $\mathcal{R}$ can be used as their symbolic representatives. (b) *cpre* and $\cap$ can be computed on such symbolic representation efficiently.

Let us start with the computation of $\mathit{cpre}_{[\Delta_i^\sharp,\tau]}(Z)$ where $\tau \in \pi_{i+1}(\mathbb{X})$, assuming that Z is of the form $\uparrow\coprod\mathcal{R}$, represented by $\mathcal{R} = \{R_1, \ldots, R_n\}$. Observe that a set of symbolic representatives $\mathcal{R}$ stands for the intersection of denotations of individual representatives, that is

$$\uparrow\coprod\mathcal{R} = \bigcap_{R_j \in \mathcal{R}} \uparrow\coprod\{R_j\}. \tag{6}$$

Z can thus be written as the *cpre*-image $\mathit{cpre}_{[\Delta_i^\sharp,\tau]}(\bigcap\mathcal{S})$ of the intersection of the elements of a set $\mathcal{S}$ having the form $\uparrow\coprod\{R_j\}, R_j \in \mathcal{R}$. Further, because *cpre* distributes over $\cap$, we can compute the *cpre*-image of an intersection by computing intersection of the *cpre*-images, i.e.

$$\mathit{cpre}_{[\Delta_i^\sharp,\tau]}(\bigcap\mathcal{S}) = \bigcap_{S \in \mathcal{S}} \mathit{cpre}_{[\Delta_i^\sharp,\tau]}(S). \tag{7}$$

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

$$\mathit{cpre}_{[\Delta_i^\sharp,\tau]}(S) = \bigcap_{\omega \in \pi_{i+1}^{-1}(\tau)} \mathit{cpre}_{[\Delta_i,\omega]}(S). \tag{8}$$

Intuitively, $\mathit{cpre}_{[\Delta_i^\sharp,\tau]}(S)$ 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 Equations 6, 7, and 8, we can write $\mathit{cpre}_{[\Delta_i^\sharp,\tau]}(Z)$ as

$$\bigcap_{\substack{S \in \mathcal{S} \\ \omega \in \pi_{i+1}^{-1}(\tau)}} \mathit{cpre}_{[\Delta_i,\omega]}(S). \tag{9}$$

To compute the individual conjuncts $\mathit{cpre}_{[\Delta_i,\omega]}(S)$, we take advantage of the fact that every S is in the special form $\uparrow\coprod\{R_j\}$, and 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 $\mathit{cpre}_{[\Delta_i,\omega]}$-image of an upward closed set is also upward closed. Moreover, we observe that it can be computed symbolically using *pre* on elements of its generators. Particularly, for a set of singletons $S = \uparrow\coprod\{R_j\}$, we get the following equation:

$$cpre_{[\Delta_i,\omega]}(\uparrow\coprod\{R_j\}) = \uparrow\coprod\left\{pre_{[\Delta^\sharp_{i-1},\omega]}(R_j)\right\}. \tag{10}$$

Intuitively, the sets with *post*-images above a singleton $\{p\} \in \big\{\{p\} \mid p \in R_j\big\} = \uparrow\coprod\{R_j\}$ are those that contain at least one state $q \in Q_{i-1}$ s.t. $q \xrightarrow{\omega} p \in \Delta^\sharp_{i-1}$. Using Equation 10, $cpre_{[\Delta^\sharp_i,\tau]}(Z)$ can be rewritten as

$$\bigcap_{\substack{R\in\mathcal{R}\\ \omega\in\pi^{-1}_{i+1}(\tau)}} \uparrow\coprod\left\{pre_{[\Delta^\sharp_{i-1},\omega]}(R_j)\right\}. \tag{11}$$

By applying Equation 6, we get the final formula for $cpre_{[\Delta^\sharp_i,\tau]}$ shown in the lemma below.

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

In order to compute $f_{N^\sharp_i}(Z)$, it remains to intersect $cpre_{[\Delta^\sharp_i,\overline{0}]}(Z)$, computed using Lemma 1, with N_i. By Equation 4(iii), N_i equals $\uparrow\coprod\{F^\sharp_{i-1}\}$, and, by Equation 6, the intersection can be done symbolically as

$$f_{N^\sharp_i}(Z) = \uparrow\coprod\left(\{F^\sharp_{i-1}\} \cup \left\{pre_{[\Delta^\sharp_{i-1},\omega]}(R_j) \mid \omega \in \pi^{-1}_{i+1}(\overline{0}), R_j \in \mathcal{R}\right\}\right). \tag{12}$$

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 Equation 12, preserves the form of sets $\uparrow\coprod\mathcal{R}$, represented by $\mathcal{R}$.

5.3 Computing $F^\sharp_i$ on Representatives of $\downarrow\mathcal{R}$-sets

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

$$f_{F^\sharp_i}(Z) = F_i \cup pre_{[\Delta^\sharp_i,\overline{0}]}(Z). \tag{13}$$

We will show that the whole fixpoint computation from Equation 4(ii) can be again carried out symbolically. We will explain the following: (a) All intermediate values of Z are of the form $\downarrow\mathcal{R}$, $\mathcal{R} \subseteq Q_i$, so the sets $\mathcal{R}$ can be used as their symbolic representatives. (b) *pre* and $\cup$ can be computed efficiently on such a symbolic representation. The computation is a simpler analogy of the one in Section 5.2.

We start with the computation of $pre_{[\Delta^\sharp_i,\tau]}(Z)$ 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 Equations 6 and 7 of Section 5.2 is that the union of downward closed sets is a downward closed set generated by the union of their generators, i.e. $\downarrow\mathcal{R} = \bigcup_{R_j \in \mathcal{R}} \downarrow\{R_j\}$ and that *pre* distributes over union, i.e.

$$pre_{[\Delta_i^\sharp,\tau]}(\bigcup\mathcal{R}) = \bigcup_{R_j\in\mathcal{R}} pre_{[\Delta_i^\sharp,\tau]}(\downarrow\{R_j\}). \tag{14}$$

An analogy of Equation 8 holds too:

$$pre_{[\Delta_i^\sharp,\tau]}(S) = \bigcup_{\omega\in\pi_{i+1}^{-1}(\tau)} pre_{[\Delta_i,\omega]}(S). \tag{15}$$

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 Equation 15, we can write $pre_{[\Delta_i^\sharp,\tau]}(Z)$ as

$$\bigcup_{\substack{R_j\in\mathcal{R}\\ \omega\in\pi_{i+1}^{-1}(\tau)}} pre_{[\Delta_i,\omega]}(\downarrow\{R_j\}). \tag{16}$$

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. 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 equation, which is a dual of Equation 10:

$$pre_{[\Delta_i,\omega]}(\downarrow\{R_j\}) = \downarrow\{cpre_{[\Delta_{i-1}^\sharp,\omega]}(R_j)\}. \tag{17}$$

Intuitively, the sets with the *post*-images below R_j are those which do not have an outgoing transition leading outside R_j. The largest such set is $cpre_{[\Delta_{i-1}^\sharp,\omega]}(R_j)$. Using Equation 17, $pre_{[\Delta_i^\sharp,\tau]}(Z)$ can be rewritten as

$$\bigcup_{\substack{R_j\in\mathcal{R}\\ \omega\in\pi_{i+1}^{-1}(\tau)}} \downarrow\{cpre_{[\Delta_{i-1}^\sharp,\omega]}(R_j)\} \tag{18}$$

which gives us the final formula for $pre_{[\Delta_i^\sharp,\tau]}$ described in Lemma 2.

Lemma 2. $pre_{[\Delta_i^\sharp,\tau]}(\downarrow\mathcal{R}) = \downarrow\{cpre_{[\Delta_{i-1}^\sharp,\omega]}(R_j) \mid \omega \in \pi_{i+1}^{-1}(\tau), R_j \in \mathcal{R}\}$.

To compute $f_{F_i^\sharp}(Z)$, it remains to unite $pre_{[\Delta_i^\sharp,\bar{0}]}(Z)$, computed using Lemma 2, with F_i. From Equation 4(i), F_i equals $\downarrow\{N_{i-1}^\sharp\}$, so the union can be done symbolically as

$$f_{F_i^\sharp}(Z) = \downarrow\Big(\{N_{i-1}^\sharp\} \cup \big\{cpre_{[\Delta_{i-1}^\sharp,\omega]}(R_j) \mid \omega \in \pi_{i+1}^{-1}(\bar{0}), R_j \in \mathcal{R}\big\}\Big). \tag{19}$$

Therefore, a symbolic application of $f_{F_i^\sharp}$ to $Z = \downarrow\mathcal{R}$ represented using the set $\mathcal{R}$ reduces to computing *cpre*-images of elements of $\mathcal{R}$, which are put next to each other, together with $N_{i-1}^\sharp$. The computation starts from $F_i = \downarrow\{N_{i-1}^\sharp\}$, represented by $\{N_{i-1}^\sharp\}$, and each of its steps, implemented by Equation 19, preserves the form of sets $\downarrow\mathcal{R}$, 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 Equations 4(ii) and 4(iv) can be represented symbolically using representatives which 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 nested symbolic representation of states of depth i. Particularly, sets of states of $Q_k, 0 \leq k \leq i$, are represented by *terms of level k* where a term of level 0 is a subset of Q_0, a term of level $2j+1$, $j \geq 0$, 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 $2j$, $j > 0$, 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 $2j+1$, Lemma 1 reduces the computation to a computation of *pre* on its sub-terms of level $2j$, which is again reduced by Lemma 2 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. We will demonstrate the run of our algorithm on the following abstract example. Consider a ground WS1S formula $\varphi = \neg\exists\mathcal{X}_3\neg\exists\mathcal{X}_2\neg\exists\mathcal{X}_1 : \varphi_0$ and an FA $\mathcal{A}_0 = (Q_0, \Delta_0, I_0 = \{a\}, F_0 = \{a, b\})$ that represents φ_0. Recall that our method decides validity of φ by computing symbolically the sequence of sets $F_0^\sharp, N_1, N_1^\sharp, F_2, F_2^\sharp, N_3$, each of them represented using a symbolic term, and then checks if $I_3 \cap N_3 \neq \emptyset$. In the following paragraph, we will show how such a sequence is computed and interleave the description with examples of possible intermediate results.

The fixpoint computation from Equation 4(ii) of the first set in the sequence, $F_0^\sharp$, is an explicit computation of the set of states backward-reachable from F_0 via $\bar{0}$ transitions of $\Delta_0^\sharp$. It is done using Equation 13, yielding, e.g. the term

$$t_{[F_0^\sharp]} = F_0^\sharp = \{a, b, c\}.$$

The fixpoint computation of $N_1^\sharp$ from Equation 4(iv) is done symbolically. It starts from the set N_1 represented using Equation 4(iii) as the term $t_{[N_1]} = \uparrow\coprod\big\{\{a, b, c\}\big\}$, and each of its iterations is carried out using Equation 12. Equation 12 transforms the problem of computing $cpre_{[\Delta_1, \omega']}$-image of a term into a computation of a series of $pre[\Delta_0^\sharp,\omega]$-images of its sub-terms, which is carried out using Equation 13 in the same way as when computing $t[F_0^\sharp]$, ending with, e.g. the term

$$t[N_1^\sharp] = \uparrow\coprod\{\{a,b,c\},\{b,c\},\{c,d\}\}.$$

The term representing F_2 is then $t[F_2] = \downarrow\{t[N_1^\sharp]\}$, due to Equation 4(i). The symbolic fixpoint computation of $F_2^\sharp$ from Equation 4(ii) then starts from $t[F_2]$, in our example

$$t[F_2] = \downarrow\Big\{\uparrow\coprod\{\{a,b,c\},\{b,c\},\{c,d\}\}\Big\}.$$

Its steps are computed using Equation 19, which transforms the computation of the image of $pre[\Delta_2^\sharp,\omega'']$ into computations of a series of $cpre[\Delta_1^\sharp,\omega']$-images of sub-terms. These are in turn transformed by Lemma 1 into computations of $pre[\Delta_0^\sharp,\omega]$-images of sub-sub-terms, subsets of Q_0, in our example yielding, e.g. the term

$$t[F_2^\sharp] = \downarrow\Big\{\uparrow\coprod\{\{a,b,c\},\{b,c\},\{c,d\}\},\uparrow\coprod\{\{b\},\{d\}\},\uparrow\coprod\{\{a\},\{c,d\}\}\Big\}.$$

Using Equation 4(iv), the final term representing N_3 is then

$$t[N_3] = \uparrow\coprod\Bigg\{\downarrow\Big\{\uparrow\coprod\{\{a,b,c\},\{b,c\},\{c,d\}\},\uparrow\coprod\{\{b\},\{d\}\},\uparrow\coprod\{\{a\},\{c,d\}\}\Big\}\Bigg\}.$$

In the next section, we will describe how we check whether $I_3 \cap F_3 \neq \emptyset$ using the computed term $t[N_3]$.

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 $I_i, 1 \leq i \leq m$, is the singleton set $\{I_{i-1}\}$, cf. Section 5.1), the test $I_m \cap F_m \neq \emptyset$ can be done efficiently over the symbolic terms representing F_m. Because $I_m = \{I_{m-1}\}$ 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 $I_{m-1} \in Y_m$ on a symbolic representation of the set Y_m of level m is based on the following equations:

$$\{x\} \in \downarrow\mathbb{Y} \iff \exists Y \in \mathbb{Y} : x \in Y \tag{20}$$

$$\{x\} \in \uparrow\coprod\mathbb{Y} \iff \forall Y \in \mathbb{Y} : x \in Y \tag{21}$$

$$\text{and for } i = 0, \qquad I_0 \in \uparrow\coprod\mathbb{Y} \iff \forall Y \in \mathbb{Y} : I_0 \cap Y \neq \emptyset. \tag{22}$$

Given a symbolic term $t[X]$ of level m representing a set $X \subseteq Q_m$, testing emptiness of $I_m \cap F_m$ or $I_m \cap N_m$ can be done over $t[X]$ by a recursive procedure that descends along the structure of $t[X]$ using Equations 20 and 21, essentially generating an AND-OR tree, terminating the descent by the use of Equation 22.

Example. In the example of Section 5.4, we would test whether $\{\{\{\{a\}\}\}\}\cap N_3 = \emptyset$ over $t_{[N_3]}$. This is equivalent to testing whether $I_2 = \{\{\{a\}\}\} \in N_3$. From Equation 21 we get that

$$I_2 \in N_3 \iff I_1 = \{\{a\}\} \in F_2^\sharp \tag{23}$$

because $F_2^\sharp$ is the denotation of the only sub-term $t_{[F_2^\sharp]}$ of $t_{[N_3]}$. Equation 20 implies that

$$I_1 = \{\{a\}\} \in F_2^\sharp \iff \{a\} \in N_1^\sharp \vee \{a\} \in \uparrow\coprod\big\{\{b\},\{d\}\big\} \vee \{a\} \in \uparrow\coprod\big\{\{a\},\{c,d\}\big\}. \tag{24}$$

Each of the disjuncts could then be further reduced by Equation 21 into a conjunction of membership queries on the base level which would be solved by Equation 22. Since none of the disjuncts is satisfied, we conclude that $I_1 \notin F_2^\sharp$, so $I_2 \notin N_3$, implying that $I_2 \in F_3$ and thus obtain the result $\models \varphi$.

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 $\mathbb{X}$ containing a pair of distinct elements $Y, Z \in \mathbb{X}$ s.t. $Y \subseteq Z$, it holds that

$$\downarrow\mathbb{X} = \downarrow(\mathbb{X} \setminus Y) \quad \text{and} \quad \uparrow\coprod\mathbb{X} = \uparrow\coprod(\mathbb{X} \setminus Z). \tag{25}$$

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

$$\downarrow\mathbb{X} \subseteq \downarrow\mathbb{Y} \iff \forall X \in \mathbb{X} \exists Y \in \mathbb{Y} : X \subseteq Y \tag{26}$$

$$\uparrow\coprod\mathbb{X} \subseteq \uparrow\coprod\mathbb{Y} \iff \forall Y \in \mathbb{Y} \exists X \in \mathbb{X} : X \subseteq Y. \tag{27}$$

Using Equations 26 and 27, 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 Equations 26 and 27 on levels greater than 0, and on level 0, where terms are subsets of Q_0, it tests subsumption by set inclusion.

Example. In the example from Section 5.4, we can use the inclusion $\{b,c\} \subseteq \{a,b,c\}$ and Equation 25 to reduce $t_{[N_1^\sharp]} = \uparrow\coprod\big\{\{a,b,c\},\{b,c\},\{c,d\}\big\}$ to the term

$$t_{[N_1]}{}' = \uparrow\coprod\big\{\{b,c\},\{c,d\}\big\}.$$

Table 1. Results for practical examples

Benchmark	**Time [s]**		**Space [states]**	
	`MONA`	`dWiNA`	`MONA`	`dWiNA`
`reverse-before-loop`	0.01	0.01	179	47
`insert-in-loop`	0.01	0.01	463	110
`bubblesort-else`	0.01	0.01	1 285	271
`reverse-in-loop`	0.02	0.02	1 311	274
`bubblesort-if-else`	0.02	0.23	4 260	1 040
`bubblesort-if-if`	0.12	1.14	8 390	2 065

Moreover, Equation 27 implies that $\uparrow\coprod\big\{\{b,c\},\{c,d\}\big\}$ is subsumed by the term $\uparrow\coprod\big\{\{b\},\{d\}\big\}$, and, therefore, we can reduce the term $t_{[F_2^\sharp]}$ to the term

$$t_{[F_2^\sharp]}' = \downarrow\Big\{\uparrow\coprod\big\{\{b\},\{d\}\big\},\uparrow\coprod\big\{\{a\},\{c,d\}\big\}\Big\}.$$

6 Experimental Evaluation

We implemented a prototype of the presented approach in the tool `dWiNA` [23] and evaluated it in a benchmark of both practical and generated examples. The tool uses the frontend of MONA to parse input formulae and also for the construction of the base automaton $\mathcal{A}_{\varphi_0}$, and further uses the MTBDD-based representation of FAs from the `libvata` [24] library. The tool supports the following two modes of operation.

In mode I, we use MONA to generate the deterministic automaton $\mathcal{A}_{\varphi_0}$ corresponding to the matrix of the formula φ, translate it to `libvata` and run our algorithm for handling the prefix of φ using `libvata`. In mode II, we first translate the formula φ into the formula φ' in prenex normal form (i.e. it consists of a quantifier prefix and a quantifier-free matrix) where the occurence of negation in the matrix is limited to literals, and then construct the nondeterministic automaton $\mathcal{A}_{\varphi_0}$ directly using `libvata`.

Our experiments were performed on an Intel Core i7-4770@3.4 GHz processor with 32 GiB RAM. The practical formulae for our experiments that we report on here were obtained from the shape analysis of [5] and evaluated using mode I of our tool; the results are shown in Table 1 (see [23] for additional experimental results). We measure the time of runs of the tools for processing only the prefix of the formulae. We can observe that w.r.t. the speed, we get comparable results; in some cases `dWiNA` is slower than MONA, which we attribute to the fact that our prototype implementation is, when compared with MONA, quite immature. Regarding space, we compare the sum of the number of states of all automata generated by MONA when processing the prefix of φ with the number of symbolic terms generated by `dWiNA` for processing the same. We can observe a significant reduction in the generated state space. We also tried to run `dWiNA`

Table 2. Results for generated formulae

	Time [s]		**Space [states]**	
k	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

on the modified formulae in mode II but ran into the problem that we were not able to construct the nondeterministic automaton for the quantifier-free matrix φ_0 in reasonable time. This was because after transformation of φ into prenex normal form, if φ_0 contains many conjunctions, the sizes of the automata generated using intersection grow too large (one of the reasons for this is that `libvata` in its current version does not support efficient reduction of automata).

To better evaluate the scalability of our approach, we created several parameterized families of WS1S formulae. We start with basic formulae encoding interesting relations among subsets of $\mathbb{N}_0$, such as existence of certain transitive relations, singleton sets, or intervals (their full definition can be found in [23]). From these we algorithmically create families of formulae with larger quantifier depth, regardless of the meaning of the created formulae (though their semantics is still nontrivial). In Table 6, we give the results for one of the families where the basic formula expresses existence of an ascending chain of n sets ordered w.r.t. $\subset$. The parameter k stands for the number of alternations in the prefix of the formulae:

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

We ran the experiments in mode II of `dWiNA` (the experiment in mode I was not successful due to a too costly conversion of a large base automaton from MONA to `libvata`).

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 (our tool often outperforms MONA) 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 is using 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 WSkS and tree-automata, and perhaps even further to more general inductive structures.

Acknowledgement. The work in this technical report was supported by the Czech Science Foundation (projects 14-11384S and 202/13/37876P), the BUT FIT project FIT-S-14-2486, and the EU/Czech IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070.

References

1. Meyer, A.R.: Weak monadic second order theory of successor is not elementary-recursive. In Parikh, R., ed.: Proc. of Logic Colloquium—Symposium on Logic Held at Boston, 1972–73. Volume 453 of Lecture Notes in Mathematics., Springer (1972) 132–154
2. Elgaard, J., Klarlund, N., Møller, A.: MONA 1.x: new techniques for WS1S and WS2S. In: Proc. of CAV'98. Volume 1427 of Lecture Notes in Computer Science., Springer (1998) 516–520
3. Klarlund, N., Møller, A.: MONA Version 1.4 User Manual. BRICS, Department of Computer Science, Aarhus University. (January 2001) Notes Series NS-01-1. Available from `http://www.brics.dk/mona/`. Revision of BRICS NS-98-3.
4. Madhusudan, P., Parlato, G., Qiu, X.: Decidable logics combining heap structures and data. In: Proc. of POPL'11, ACM (2011) 611–622
5. Madhusudan, P., Qiu, X.: Efficient decision procedures for heaps using STRAND. In: Proc. of SAS'11. Volume 6887 of Lecture Notes in Computer Science., Springer (2011) 43–59
6. 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
7. 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
8. Zee, K., Kuncak, V., Rinard, M.C.: Full functional verification of linked data structures. In: Proc. of POPL'08, ACM (2008) 349–361
9. Hamza, J., Jobstmann, B., Kuncak, V.: Synthesis for regular specifications over unbounded domains. In: Proc. of FMCAD'10, IEEE (2010) 101–109

10. Topnik, C., Wilhelm, E., Margaria, T., Steffen, B.: jMosel: A stand-alone tool and jABC plugin for M2L(Str). In: Proc. of SPIN'06. Volume 3925 of Lecture Notes in Computer Science., Springer (2006) 293–298
11. Ganzow, T., Kaiser, L.: New algorithm for weak monadic second-order logic on inductive structures. In: Proc. CSL'10. Volume 6247 of Lecture Notes in Computer Science., Springer (2010) 366–380
12. Wies, T., Muñiz, M., Kuncak, V.: An efficient decision procedure for imperative tree data structures. In Bjørner, N., Sofronie-Stokkermans, V., eds.: Proc. of CADE'11. Volume 6803 of Lecture Notes in Computer Science., Springer (2011) 476–491
13. Doyen, L., Raskin, J.F.: Antichain algorithms for finite automata. In: Proc. of TACAS'10. Volume 6015 of LNCS., Springer (2010) 2–22
14. Wulf, M.D., Doyen, L., Henzinger, T.A., Raskin, J.F.: Antichains: A new algorithm for checking universality of finite automata. In: Proc. of CAV'06. Volume 4144 of LNCS., Springer (2006) 17–30
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.: Proc. of TACAS'10. Volume 6015 of Lecture Notes in Computer Science., Springer (2010) 158–174
16. Bustan, D., Grumberg, O.: Simulation based minimization. In: Proc. of CADE'00. Volume 1831 of Lecture Notes in Computer Science., Springer (2000) 255–270
17. Abdulla, P.A., Bouajjani, A., Holík, L., Kaati, L., Vojnar, T.: Computing simulations over tree automata: Efficient techniques for reducing tree automata. In: Proc. of TACAS'08. Volume 4963 of LNCS., Springer (2008) 93–108
18. Bouajjani, A., Habermehl, P., Holík, L., Touili, T., Vojnar, T.: Antichain-based universality and inclusion testing over nondeterministic finite tree automata. In: Proc. of CIAA'08. Volume 5148 of LNCS., Springer (2008) 57–67
19. Habermehl, P., Holík, L., Rogalewicz, A., Simácek, J., Vojnar, T.: Forest automata for verification of heap manipulation. Formal Methods in System Design **41**(1) (2012) 83–106
20. Bonchi, F., Pous, D.: Checking NFA equivalence with bisimulations up to congruence. In: Proc. of POPL'13, ACM (2013) 457–468
21. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications. (2008)
22. Büchi, J.R.: Weak second-order arithmetic and finite automata. Technical report, The University of Michigan (1959) Available at URL: `http://hdl.handle.net/2027.42/3930` (May 2010).
23. Fiedor, T., Holík, L., Lengál, O., Vojnar, T.: `dWiNA` (2014) Available from `http://www.fit.vutbr.cz/research/groups/verifit/tools/dWiNA/`.
24. Lengál, O., Šimáček, J., Vojnar, T.: VATA: A library for efficient manipulation of non-deterministic tree automata. In: Proc. of TACAS'12. Volume 7214 of Lecture Notes in Computer Science., Springer (2012) 79–94

A Proofs for Section 5

Lemma 3. *Let $\mathcal{X}$ and $\mathcal{Y}$ be sets of sets. Then it holds that*

$$\uparrow\coprod\mathbb{X} \cap \uparrow\coprod\mathbb{Y} = \uparrow\coprod(\mathbb{X} \cup \mathbb{Y}). \tag{28}$$

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

$$\begin{aligned} \uparrow\coprod\mathbb{X} &= \uparrow\big\{\{x_1,\ldots,x_n\} \,\big|\, (x_1,\ldots,x_n) \in \prod \mathbb{X}\big\} \quad \text{and} \\ \uparrow\coprod\mathbb{Y} &= \uparrow\big\{\{y_1,\ldots,y_m\} \,\big|\, (y_1,\ldots,y_m) \in \prod \mathbb{Y}\big\}. \end{aligned} \tag{29}$$

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 set $X \cup Y$. 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

$$\uparrow\big\{\{x_1,\ldots,x_n\} \cup \{y_1,\ldots,y_m\} \,\big|\, (x_1,\ldots,x_n) \in \prod X \wedge (y_1,\ldots,y_m) \in \prod Y\big\}. \tag{30}$$

We observe that this set can be also expressed as

$$\uparrow\big\{\{x_1,\ldots,x_n,y_1,\ldots,y_m\} \,\big|\, (x_1,\ldots,x_n,y_1,\ldots y_m) \in \prod(X \cup Y)\big\} \tag{31}$$

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

Lemma 4. *(Equation 6) Let $\mathbb{R}$ be a set of sets. Then, it holds that*

$$\uparrow\coprod\mathbb{R} = \bigcap_{R_j \in \mathbb{R}} \uparrow\coprod\{R_j\}. \tag{32}$$

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 3. □

Lemma 5. *(Equation 10) Let $R_j \subseteq Q_{i-1}$ and ω be a symbol over $\pi_i(\mathbb{X})$ for $i > 0$. Then*

$$\mathit{cpre}_{[\Delta_i,\omega]}(\uparrow\coprod\{R_j\}) = \uparrow\coprod\big\{\mathit{pre}_{[\Delta^\sharp_{i-1},\omega]}(R_j)\big\}. \tag{33}$$

Proof. First, we show that the set $\mathit{cpre}_{[\Delta_i,\omega]}(\uparrow\coprod\{R_j\})$ is upward closed. Second, we show that all elements of the set $\coprod\big\{\mathit{pre}_{[\Delta^\sharp_{i-1},\omega]}(R_j)\big\}$ are contained in $\mathit{cpre}_{[\Delta_i,\omega]}(\uparrow\coprod\{R_j\})$. Finally, we show that for every element T in the set $\mathit{cpre}_{[\Delta_i,\omega]}(\uparrow\coprod\{R_j\})$ there is a smaller element S in the set $\coprod\big\{\mathit{pre}_{[\Delta^\sharp_{i-1},\omega]}(R_j)\big\}$.

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

$$\mathit{post}_{[\Delta_i,\omega]}(\{S\}) \subseteq \uparrow\coprod\{R_j\}, \tag{34}$$

and from the definition of Δ_i, it holds that

$$post_{[\Delta_i,\omega]}(\{S\}) = \{post_{[\Delta^\sharp_{i-1},\omega]}(S)\}. \tag{35}$$

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

$$post_{[\Delta^\sharp_{i-1},\omega]}(T) \supseteq post_{[\Delta^\sharp_{i-1},\omega]}(S) \tag{36}$$

and, therefore, it also holds that

$$post_{[\Delta_i,\omega]}(\{T\}) = \{post_{[\Delta^\sharp_{i-1},\omega]}(T)\} \subseteq \uparrow\coprod\{R_j\}. \tag{37}$$

Therefore, $T \in cpre_{[\Delta_i,\omega]}(\uparrow\coprod\{R_j\})$ and the set $cpre_{[\Delta_i,\omega]}(\uparrow\coprod\{R_j\})$ 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 $S \in cpre_{[\Delta_i,\omega]}(\uparrow\coprod\{R_j\})$: 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 $S \in cpre_{[\Delta_i,\omega]}(\uparrow\coprod\{R_j\})$.
3. Proving that for every $T \in cpre_{[\Delta_i,\omega]}(\uparrow\coprod\{R_j\})$ there exists some element $S \in \coprod\left\{pre_{[\Delta^\sharp_{i-1},\omega]}(R_j)\right\}$ such that $S \subseteq T$: From $T \in cpre_{[\Delta_i,\omega]}(\uparrow\coprod\{R_j\})$ and the definition of Δ_i, we have that

$$post_{[\Delta_i,\omega]}(\{T\}) = \{P\} \subseteq \uparrow\coprod\{R_j\} \tag{38}$$

for P s.t. $post_{[\Delta^\sharp_{i-1},\omega]}(T) = P$. Since $P \in \uparrow\coprod\{R_j\}$, there exists $r \in R_j \cap P$ 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. □

Lemma 6. *(Equation 17) Let $R_j \subseteq Q_{i-1}$ and ω be a symbol over $\pi_i(\mathbb{X})$ for $i > 0$. Then*

$$pre_{[\Delta_i,\omega]}(\downarrow\{R_j\}) = \downarrow\{cpre_{[\Delta^\sharp_{i-1},\omega]}(R_j)\}. \tag{39}$$

Proof. First, we show that $pre_{[\Delta_i,\omega]}(\downarrow\{R_j\})$ is downward closed. Second, we show that $S = cpre_{[\Delta^\sharp_{i-1},\omega]}(R_j)$ 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 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

$$post_{[\Delta_i,\omega]}(\{S'\}) = \{post_{[\Delta^\sharp_{i-1},\omega]}(S')\} \subseteq \downarrow\{R_j\}, \tag{40}$$

and, therefore, $post_{[\Delta^\sharp_{i-1},\omega]}(S') \in \downarrow\{R_j\}$. For $T \subseteq S'$, it clearly holds that

$$post_{[\Delta^\sharp_{i-1},\omega]}(T) \subseteq post_{[\Delta^\sharp_{i-1},\omega]}(S') \tag{41}$$

and so it also holds that

$$post_{[\Delta_i,\omega]}(\{T\}) = \{post_{[\Delta^\sharp_{i-1},\omega]}(T)\} \subseteq \downarrow\{R_j\}. \tag{42}$$

Therefore, $T \in pre_{[\Delta_i,\omega]}(\downarrow\{R_j\})$ and $pre_{[\Delta_i,\omega]}(\downarrow\{R_j\})$ is downward closed.

2. Proving that $S = cpre_{[\Delta^\sharp_{i-1},\omega]}(R_j) \in pre_{[\Delta_i,\omega]}(\downarrow\{R_j\})$: From the definition of *cpre*, it holds that
$$post_{[\Delta^\sharp_{i-1},\omega]}(S) = S' \subseteq R_j. \tag{43}$$
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
$$P \subseteq R_j \quad \Longrightarrow \quad cpre_{[\Delta^\sharp_{i-1},\omega]}(P) \subseteq cpre_{[\Delta^\sharp_{i-1},\omega]}(R_j). \tag{44}$$
Therefore, we can conclude that $T \subseteq cpre_{[\Delta^\sharp_{i-1},\omega]}(R_j) = S$. □