




Automata Terms in a Lazy $WSkS$ Decision Procedure

Vojtěch Havlena · Lukáš Holík^{} ·
Ondřej Lengál^{} · Tomáš Vojnar^{}

Received: date / Accepted: date

Abstract We propose a lazy decision procedure for the logic $WSkS$. It builds a term-based symbolic representation of the state space of the tree automaton (TA) constructed by the classical $WSkS$ decision procedure. The classical decision procedure transforms the symbolic representation into a TA via a bottom-up traversal and then tests its language non-emptiness, which corresponds to satisfiability of the formula. On the other hand, we start evaluating the representation from the top, construct the state space on the fly, and utilize opportunities to prune away parts of the state space irrelevant to the language emptiness test. In order to do so, we needed to extend the notion of *language terms* (denoting language derivatives) used in our previous procedure for the linear fragment of the logic (the so-called $WS1S$) into *automata terms*. We implemented our decision procedure and identified classes of formulae on which our prototype implementation is significantly faster than the classical procedure implemented in the MONA tool.

Keywords $WSkS$ · tree automata · automata term · finite automata · monadic second-order logic

1 Introduction

Weak monadic second-order logic of k successors ($WSkS$) is a logic for describing regular properties of finite k -ary trees. In addition to talking about trees, $WSkS$ can also encode complex properties of a rich class of general graphs by referring to their tree backbones [28]. $WSkS$ offers extreme succinctness for the price of non-elementary worst-case complexity. As noticed first by the authors of [16] in the context of $WS1S$ (a restriction that speaks about finite words only), the trade-off between complexity and succinctness may, however, be turned significantly favourable in many practical

Faculty of Information Technology, IT4I Centre of Excellence
Brno University of Technology
Božetěchova 2
612 00 Brno
Czech Republic
E-mail: lengal@fit.vutbr.cz

cases through a use of clever implementation techniques and heuristics. Such techniques were then elaborated in the tool MONA [12, 22], the best-known implementation of decision procedures for WS1S and WS2S. MONA has found numerous applications in verification of programs with complex dynamic linked data structures [28, 25, 26, 8, 42], string programs [34], array programs [43], parametric systems [3, 4, 6], distributed systems [24, 32], hardware verification [2], automated synthesis [31, 20, 18], and even computational linguistics [29].

Despite the extensive research and engineering effort invested into MONA, due to which it still offers the best all-around performance among existing WS1S/WS2S decision procedures, it is, however, easy to reach its scalability limits. Particularly, MONA implements the classical WS1S/WS2S decision procedures that build a word/tree automaton representing models of the given formula and then check emptiness of the automaton’s language. The non-elementary complexity manifests in that the size of the automaton is prone to explode, which is caused mainly by the repeated determinisation (needed to handle negation and alternation of quantifiers) and synchronous product construction (used to handle conjunctions and disjunctions). Users of WS k S are then forced to either find workarounds, such as in [26], or, often restricting the input of their approach, give up using WS k S altogether [38].

As in MONA, we further consider WS2S only (this does not change the expressive power of the logic since k -ary trees can be easily encoded into binary ones). We revisit the use of tree automata (TAs) in the WS2S decision procedure and obtain a new decision procedure that is much more efficient in certain cases. It is inspired by works on *antichain algorithms* for efficient testing of universality and language inclusion of finite automata [11, 39, 5, 1], which implement the operations of testing emptiness of a complement (universality) or emptiness of a product of one automaton with the complement of the other one (language inclusion) via an *on-the-fly* determinisation and product construction. The on-the-fly approach allows one to achieve significant savings by pruning the state space that is irrelevant for the language emptiness test. The pruning is achieved by early termination when detecting non-emptiness (which represents a simple form of *lazy evaluation*), and *subsumption* (which basically allows one to disregard proof obligations that are implied by other ones). Antichain algorithms and their generalizations have shown great efficiency improvements in applications such as abstract regular model checking [5], shape analysis [17], LTL model checking [40], or game solving [41].

Our work generalizes the above mentioned approaches of on-the-fly automata construction, subsumption, and lazy evaluation for the needs of deciding WS2S. In our procedure, the TAs that are constructed explicitly by the classical procedure are represented symbolically by the so-called *automata terms*. More precisely, we build automata terms for subformulae that start with a quantifier (and for the top-level formula) only—unlike the classical procedure, which builds a TA for every subformula. Intuitively, automata terms specify the set of leaf states of the TAs of the appropriate (sub)formulae. The leaf states themselves are then represented by *state terms*, whose structure records the automata constructions (corresponding to Boolean operations and quantification on the formula level) used to create the given TAs from base TAs corresponding to atomic formulae. The leaves of the terms correspond to states of the base automata. Automata terms may be used as state terms over which further automata terms of an even higher level are built. Non-leaf states, the transition relation, and root states are then given implicitly by the transition relations of the base automata and the structure of the state terms.

Our approach is a generalization of our earlier work [13] on WS1S. Although the term structure and the generalized algorithm may seem close to [13], the reasoning behind it is significantly more involved. Particularly, [13] is based on defining the semantics (language) of terms as a function of the semantics of their sub-terms. For instance, the semantics of the term $\{q_1, \dots, q_n\}$ is defined as the union of languages of the state terms q_1, \dots, q_n , where the language of a state of the base automaton consists of the words *accepted at that state*. With TAs, it is, however, not meaningful to talk about trees accepted from a leaf state, instead, we need to talk about a given state and its *context*, i.e., other states that could be obtained via a bottom-up traversal over the given set of symbols. Indeed, trees have multiple leaves, which may be accepted by a number of different states, and so a tree is *accepted from a set of states*, not from any single one of them alone. We therefore cannot define the semantics of a state term as a tree language, and so we cannot define the semantics of an automata term as the union of the languages of its state sub-terms. This problem seems critical at first because without a sensible notion of the meaning of terms, a straightforward generalization of the algorithm of [13] to trees is not possible. The solution we present here is based on defining the semantics of terms *not* as functions of languages of their sub-terms, but, instead, via the automata constructions they represent.

Unlike the classical decision procedure, which builds a TA corresponding to a formula *bottom-up*, i.e. from the atomic formulae, we build automata terms *top-down*, i.e., from the top-level formula. This approach offers a lot of space for various optimisations. Most importantly, we test non-emptiness of the terms *on the fly* during their construction and construct the terms *lazily*. In particular, we use *short-circuiting* for dealing with the \wedge and \vee connectives and *early termination* with possible *continuation* when implementing the fixpoint computations needed when dealing with quantifiers. That is, we terminate the fixpoint computation whenever the emptiness can be decided in the given computation context and continue with the computation when such a need appears once the context is changed on some higher-level term. Further, we define a notion of *subsumption* of terms, which, intuitively, compares the terms with respect to the sets of trees they represent, and allows us to discard terms that are subsumed by others.

We have implemented our approach in a prototype tool. When experimenting with it, we have identified multiple parametric families of WS2S formulae where our implementation can—despite its prototypical form—significantly outperform MONA. We find this encouraging since there is a lot of space for further optimisations and, moreover, our implementation can be easily combined with MONA by treating automata constructed by MONA in the same way as if they were obtained from atomic predicates.

This paper is an extended version of the paper with the same name that appeared in the proceedings of CADE-27 [19], containing more examples and complete proofs of the presented lemmas and theorems, as well as one more optimization of our efficient decision procedure (cf. Section 4.5).

2 Preliminaries

In this section, we introduce basic notation, trees, and tree automata, and give a quick introduction to the *weak monadic second-order logic of two successors* (WS2S) and

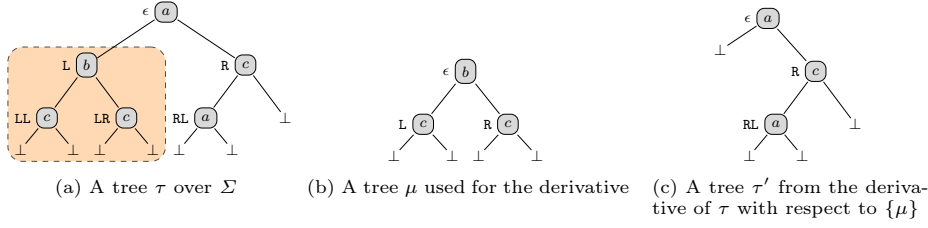


Fig. 1: An example of the derivative. Consider trees τ and μ over the alphabet $\Sigma = \{a, b, c\}$ given in (a) and (b) respectively. The derivative of τ with respect to $\{\mu\}$ is the set $\{\tau, \tau'\}$ where τ' is given in (c).

its classical decision procedure. We give the minimal syntax of WS2S only; see, e.g., Comon *et al.* [9] for more details.

2.1 Basics, Trees, and Tree Automata.

Let Σ be a finite set of symbols, called an *alphabet*. The set Σ^* of *words* over Σ consists of finite sequences of symbols from Σ . The *empty word* is denoted by ϵ , with $\epsilon \notin \Sigma$. The *concatenation* of two words u and v is denoted by $u.v$ or simply uv . The *domain* of a partial function $f : X \rightarrow Y$ is the set $\text{dom}(f) = \{x \in X \mid \exists y : x \mapsto y \in f\}$, its *image* is the set $\text{img}(f) = \{y \in Y \mid \exists x : x \mapsto y \in f\}$, and its *restriction* to a set Z is the function $f|_Z = f \cap (Z \times Y)$. For a binary operator \bullet , we write $A[\bullet]B$ to denote the augmented product $\{a \bullet b \mid (a, b) \in A \times B\}$ of A and B .

We will consider ordered binary trees. We call a word $p \in \{L, R\}^*$ a *tree position* and $p.L$ and $p.R$ its *left* and *right child*, respectively. Given an alphabet Σ such that $\perp \notin \Sigma$, a *tree* over Σ is a finite partial function $\tau : \{L, R\}^* \rightarrow (\Sigma \cup \{\perp\})$ such that (i) $\text{dom}(\tau)$ is non-empty and prefix-closed, and (ii) for all positions $p \in \text{dom}(\tau)$, either $\tau(p) \in \Sigma$ and p has both children, or $\tau(p) = \perp$ and p has no children, in which case it is called a *leaf*. We let $\text{leaf}(\tau)$ be the set of all leaves of τ . The position ϵ is called the *root*, and we write Σ^* to denote the set of all trees over Σ . (Intuitively, the $[\cdot]^*$ operator can be seen as a generalization of the Kleene star to tree languages. The symbol \star is the Chinese character for a tree.) We abbreviate $\{a\}^*$ as a^* for $a \in \Sigma$.

The *sub-tree* of τ rooted at a position $p \in \text{dom}(\tau)$ is the tree $\tau' = \{p' \mapsto \tau(p.p') \mid p.p' \in \text{dom}(\tau)\}$. A *prefix* of τ is a tree τ' such that $\tau'|_{\text{dom}(\tau') \setminus \text{leaf}(\tau')} \subseteq \tau|_{\text{dom}(\tau) \setminus \text{leaf}(\tau)}$. The *derivative* of a tree τ with respect to a set of trees $S \subseteq \Sigma^*$ is the set $\tau - S$ of all prefixes τ' of τ such that, for each position $p \in \text{leaf}(\tau')$, the sub-tree of τ at p either belongs to S or it is a leaf of τ . Intuitively, $\tau - S$ are all prefixes of τ obtained from τ by removing some of the sub-trees in S . The derivative of a set of trees $T \subseteq \Sigma^*$ with respect to S is the set $\bigcup_{\tau \in T} (\tau - S)$. See Fig. 1 for an example of the derivative.

A (binary) *tree automaton* (TA) over an alphabet Σ is a quadruple $\mathcal{A} = (Q, \delta, I, R)$ where Q is a finite set of *states*, $\delta : Q^2 \times \Sigma \rightarrow 2^Q$ is a *transition function*, $I \subseteq Q$ is a set of *leaf states*, and $R \subseteq Q$ is a set of *root states*. We use $(q, r) \cdot \{a\} \rightarrow s$ to denote that $s \in \delta((q, r), a)$. A *run* of \mathcal{A} on a tree τ is a total map $\rho : \text{dom}(\tau) \rightarrow Q$ such that if $\tau(p) = \perp$, then $\rho(p) \in I$, else $(\rho(p.L), \rho(p.R)) \cdot \{a\} \rightarrow \rho(p)$ with $a = \tau(p)$. The run ρ is *accepting* if $\rho(\epsilon) \in R$, and the *language* $\mathcal{L}(\mathcal{A})$ of \mathcal{A} is the set of all trees on which \mathcal{A} has an accepting run. \mathcal{A} is *deterministic* if $|I| = 1$ and $\forall q, r \in Q, a \in \Sigma : |\delta((q, r), a)| \leq 1$, and *complete*

if $I \geq 1$ and $\forall q, r \in Q, a \in \Sigma : |\delta((q, r), a)| \geq 1$. Last, for $a \in \Sigma$, we shorten $\delta((q, r), a)$ as $\delta_a(q, r)$, and we use $\delta_F(q, r)$ to denote $\bigcup \{\delta_a(q, r) \mid a \in F\}$ for a set $F \subseteq \Sigma$.

2.2 Syntax and Semantics of WS2S.

WS2S is a logic that allows quantification over second-order *variables*, which are denoted by upper-case letters X, Y, \dots and range over *finite sets* of tree positions in $\{\mathbf{L}, \mathbf{R}\}^*$ (the finiteness of variable assignments is reflected in the name *weak*). See Fig. 2a for an example of a set of positions assigned to a variable. Atomic formulae (atoms) of WS2S are of the form: (i) $X \subseteq Y$, (ii) $X = S_L(Y)$, and (iii) $X = S_R(Y)$. Informally, the $S_L(Y)$ function returns all positions from Y shifted to their left child and the $S_R(Y)$ function returns all positions from Y shifted to their right child. Formulae are constructed from atoms using the logical connectives \wedge, \vee, \neg , and the quantifier $\exists \mathbb{X}$ where \mathbb{X} is a finite set of variables (we write $\exists X$ when \mathbb{X} is the singleton set $\{X\}$). Other connectives (such as \Rightarrow or \forall) and predicates (such as the predicate $\text{Sing}(X)$ for the singleton set X) can be obtained as syntactic sugar (e.g., we can define the emptiness predicate $X = \emptyset$ as $\forall Y. X \subseteq Y$ and the singleton predicate $\text{Sing}(X)$ as $\forall Y. Y \subseteq X \Rightarrow (Y = X \vee Y = \emptyset)$).

A *valuation* of a set of variables \mathbb{X} is an assignment $\nu : \mathbb{X} \rightarrow 2^{\{\mathbf{L}, \mathbf{R}\}^*}$ of \mathbb{X} to finite subsets of $\{\mathbf{L}, \mathbf{R}\}^*$. We use $\nu \triangleleft \{X \mapsto S\}$ to denote a valuation obtained from ν by changing the assignment of X to S . A *model* of a WS2S formula $\varphi(\mathbb{X})$ with the set of free variables \mathbb{X} is a valuation of \mathbb{X} for which the formula is *satisfied*, written $\nu \models \varphi$. Satisfaction of formulae is defined as follows:

- (i) $\nu \models X \subseteq Y$ if and only if $\nu(X)$ is a subset of $\nu(Y)$,
- (ii) $\nu \models X = S_L(Y)$ if and only if $\nu(X)$ is $\{p.\mathbf{L} \mid p \in \nu(Y)\}$,
- (iii) $\nu \models X = S_R(Y)$ if and only if $\nu(X)$ is $\{p.\mathbf{R} \mid p \in \nu(Y)\}$,
- (iv) $\nu \models \neg \varphi$ if and only if not $\nu \models \varphi$,
- (v) $\nu \models \varphi \wedge \psi$ if and only if $\nu \models \varphi$ and $\nu \models \psi$,
- (vi) $\nu \models \varphi \vee \psi$ if and only if $\nu \models \varphi$ or $\nu \models \psi$, and
- (vii) $\nu \models \exists X. \varphi$ if and only if there is a finite $S \subseteq \{\mathbf{L}, \mathbf{R}\}^*$ such that $\nu \triangleleft \{X \mapsto S\} \models \varphi$.

A formula φ is *valid*, written $\models \varphi$, if and only if all assignments of its free variables are its models, and *satisfiable* if it has a model. Without loss of generality, we assume that each variable in a formula either has only free occurrences or is quantified exactly once; we denote the set of (free and quantified) variables occurring in a formula φ as $\text{Vars}(\varphi)$.

2.3 Representing Models as Trees.

We fix a formula φ with variables $\text{Vars}(\varphi) = \mathbb{X}$. A *symbol* ξ over \mathbb{X} is a (total) function $\xi : \mathbb{X} \rightarrow \{0, 1\}$, e.g., $\xi = \{X \mapsto 0, Y \mapsto 1\}$ is a symbol over $\mathbb{X} = \{X, Y\}$. We use $\Sigma_{\mathbb{X}}$ to denote the set of all symbols over \mathbb{X} and $\vec{0}$ to denote the symbol mapping all variables in \mathbb{X} to 0, i.e., $\vec{0} = \{X \mapsto 0 \mid X \in \mathbb{X}\}$.

A finite assignment $\nu : \mathbb{X} \rightarrow 2^{\{\mathbf{L}, \mathbf{R}\}^*}$ of φ 's variables can be encoded as a finite tree τ_ν of symbols over \mathbb{X} where every position $p \in \{\mathbf{L}, \mathbf{R}\}^*$ satisfies the following conditions: (a) if $p \in \nu(X)$, then $\tau_\nu(p)$ contains $\{X \mapsto 1\}$, and (b) if $p \notin \nu(X)$, then either $\tau_\nu(p)$ contains $\{X \mapsto 0\}$ or $\tau_\nu(p') = \perp$ for some prefix p' of p (note that the occurrences of \perp in τ are limited since τ still needs to be a tree). Observe that ν can have multiple

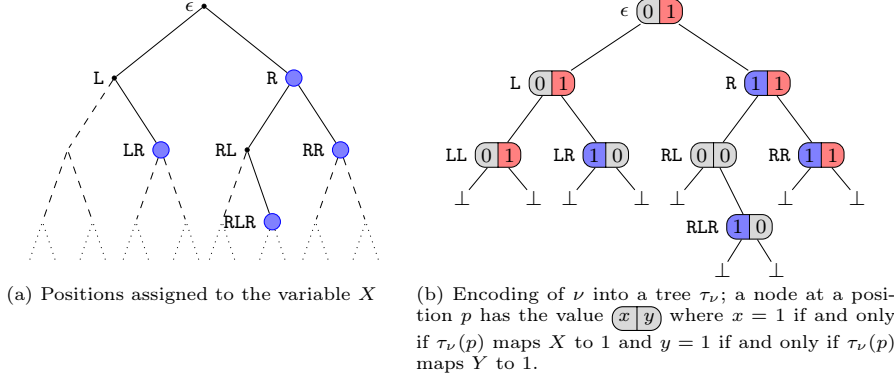


Fig. 2: An example of an assignment ν to a pair of variables $\{X, Y\}$ such that $\nu(X) = \{LR, R, RLR, RR\}$ and $\nu(Y) = \{\epsilon, L, LL, R, RR\}$ and its encoding into a tree

encodings: the unique minimum one τ_ν^{min} and (infinitely many) extensions of τ_ν^{min} with $\vec{0}$ -only trees. The *language* of φ is defined as the set of all encodings of its models $\mathcal{L}(\varphi) = \{\tau_\nu \in \Sigma_{\mathbb{X}}^* \mid \nu \models \varphi \text{ and } \tau_\nu \text{ is an encoding of } \nu\}$. See Fig. 2 for an example of an assignment and its encoding.

Let ξ be a symbol over \mathbb{X} . For a set of variables $\mathbb{Y} \subseteq \mathbb{X}$, we define the *projection* of ξ with respect to \mathbb{Y} as the set of symbols $\pi_{\mathbb{Y}}(\xi) = \{\xi' \in \Sigma_{\mathbb{X}} \mid \xi|_{\mathbb{X} \setminus \mathbb{Y}} \subseteq \xi'\}$. Intuitively, the projection removes the original assignments of variables from \mathbb{Y} and allows them to be substituted by any possible value. We define $\pi_{\mathbb{Y}}(\perp) = \perp$ and write π_Y if \mathbb{Y} is the singleton set $\{Y\}$. As an example, for $\mathbb{X} = \{X, Y\}$ the projection of $\vec{0}$ with respect to $\{X\}$ is given as $\pi_X(\vec{0}) = \{\{X \mapsto 0, Y \mapsto 0\}, \{X \mapsto 1, Y \mapsto 0\}\}$.¹ The definition of projection can be extended to trees τ over $\Sigma_{\mathbb{X}}$ so that $\pi_{\mathbb{Y}}(\tau)$ is the set of trees $\{\tau' \in \Sigma_{\mathbb{X}}^* \mid \forall p \in \text{dom}(\tau) : \text{if } \tau(p) = \perp, \text{ then } \tau'(p) = \perp, \text{ else } \tau'(p) \in \pi_{\mathbb{Y}}(\tau(p))\}$ and subsequently to languages L so that $\pi_{\mathbb{Y}}(L) = \bigcup \{\pi_{\mathbb{Y}}(\tau) \mid \tau \in L\}$.

2.4 The Classical Decision Procedure for WS2S.

The classical decision procedure for the WS2S logic goes through a direct construction of a TA \mathcal{A}_φ having the same language as a given formula φ . Let us briefly recall the automata constructions used (cf. [9]). Given a complete TA $\mathcal{A} = (Q, \delta, I, R)$, the *completion* assumes that \mathcal{A} is deterministic and returns $\mathcal{A}^G = (Q, \delta, I, Q \setminus R)$, the projection returns $\pi_X(\mathcal{A}) = (Q, \delta^{\pi_X}, I, R)$ with $\delta_a^{\pi_X}(q, r) = \delta_{\pi_X(a)}(q, r)$, and the *subset construction* returns the deterministic and complete automaton $\mathcal{A}^D = (2^Q, \delta^D, \{I\}, R^D)$ where $\delta_a^D(S, S') = \bigcup_{q \in S, q' \in S'} \delta_a(q, q')$ and $R^D = \{S \subseteq Q \mid S \cap R \neq \emptyset\}$. The binary operators $\circ \in \{\cup, \cap\}$ are implemented through a *product construction*, which—given the TA \mathcal{A} and another complete TA $\mathcal{A}' = (Q', \delta', I', R')$ —returns the automaton $\mathcal{A} \circ \mathcal{A}' = (Q \times Q', \Delta^\times, I^\times, R^\circ)$ where $\Delta_a^\times((q, r), (q', r')) = \Delta_a(q, q') \times \Delta_a'(r, r')$, $I^\times = I \times I'$, and for $(q, r) \in Q \times Q'$, $(q, r) \in R^\cap \Leftrightarrow q \in R \wedge r \in R'$ and $(q, r) \in R^\cup \Leftrightarrow q \in R \vee r \in R'$.

¹ Note that our definition of projection differs from the usual one, which would create a symbol over the alphabet $\mathbb{X} \setminus \mathbb{Y}$; in the example, it would produce a single symbol $\{Y \mapsto 0\}$ over the alphabet of symbols over $\{Y\}$.

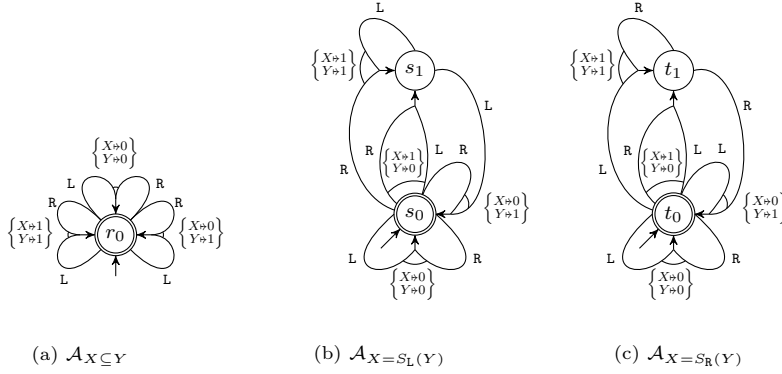


Fig. 3: Tree automata for atomic WS2S formulae. Transitions are represented using multiple-source hyper-edges. For instance, the transition $(s_0, s_1) \cdot \{X \mapsto 1, Y \mapsto 1\} \rightarrow s_1$ in $\mathcal{A}_{X=S_L(Y)}$ is represented by the hyper-edge with sources s_0 and s_1 over the symbol $\{X \mapsto 1, Y \mapsto 1\}$ that joins just before entering s_1 . The L and R labels on the “legs” of the hyper-edge going to s_0 and s_1 denote the position in the left-hand side of the transition (L and R stand for “left” and “right”).

Testing non-emptiness of \mathcal{A} can be implemented through the equivalence $\mathcal{L}(\mathcal{A}) \neq \emptyset$ if and only if $\text{reach}_\delta(I) \cap R \neq \emptyset$ where the set $\text{reach}_\delta(S)$ of states *reachable* from a set $S \subseteq Q$ through δ -transitions is computed as the least fixpoint

$$\text{reach}_\delta(S) = \mu Z. S \cup \bigcup_{q,r \in Z} \delta(q, r). \quad (1)$$

The same fixpoint computation is used to compute the derivative with respect to a^\star for some $a \in \Sigma$ as $\mathcal{A} - a^\star = (Q, \delta, \text{reach}_{\delta_a}(I), R)$: the new leaf states are all those reachable from I through a -transitions.

The classical WSkS decision procedure uses the above operations to construct the automaton \mathcal{A}_φ inductively to the structure of φ as follows: (i) If φ is an atomic formula, then \mathcal{A}_φ is a pre-defined *base* TA over Σ_X (we show those TAs in Fig. 3). (ii) If $\varphi = \varphi_1 \wedge \varphi_2$, then $\mathcal{A}_\varphi = \mathcal{A}_{\varphi_1} \cap \mathcal{A}_{\varphi_2}$. (iii) If $\varphi = \varphi_1 \vee \varphi_2$, then $\mathcal{A}_\varphi = \mathcal{A}_{\varphi_1} \cup \mathcal{A}_{\varphi_2}$. (iv) If $\varphi = \neg\psi$, then $\mathcal{A}_\varphi = \mathcal{A}_\psi^c$. (v) Finally, if $\varphi = \exists X. \psi$, then $\mathcal{A}_\varphi = (\pi_X(\mathcal{A}_\psi))^D - \vec{0}^\star$.

Points (i) to (iv) are self-explanatory. In point (v), the projection implements the quantification by forgetting the values of the X component of all symbols. Since this yields non-determinism, projection is followed by determinisation by the subset construction. Further, the projection can produce some new trees that contain $\vec{0}$ -only labelled sub-trees, which need not be present in some smaller encodings of the same model. Consider, for example, a formula ψ having the language $\mathcal{L}(\psi)$ given by the tree τ_ν in Fig. 2b and all its $\vec{0}$ -extensions. To obtain $\mathcal{L}(\exists X. \psi)$, it is not sufficient to make the projection $\pi_X(\mathcal{L}(\psi))$ because the projected language does not contain the minimum encoding τ_ν^{\min} of $\nu : Y \mapsto \{\epsilon, L, LL, R, RR\}$, but only those encodings ν' such that $\nu'(\text{RLR}) = \{Y \mapsto 0\}$. Therefore, the $\vec{0}$ -derivative is needed to saturate the language with *all* encodings of the encoded models (if some of these encodings were missing, the inductive construction could produce a wrong result, for instance, if the language were subsequently complemented). As mentioned above, on the level of automata, the $\vec{0}$ derivative can be achieved by replacing the set of leaf states I of \mathcal{A}_φ by $\text{reach}_{\Delta_{\vec{0}}}(I)$ where Δ is the transition function of \mathcal{A}_φ . See [9] for more details.

3 Automata Terms

Our algorithm for deciding WS2S may be seen as an alternative implementation of the classical procedure from Section 2.4. The main innovation is the data structure of *automata terms*, which implicitly represent the automata constructed by the automata operations. Unlike the classical procedure—which proceeds by a bottom-up traversal on the formula structure, building an automaton for each sub-formula before proceeding upwards—automata terms allow for constructing parts of automata at higher levels from parts of automata on the lower levels even though the construction of the lower level automata has not yet finished. This allows one to test the language emptiness on the fly and use techniques of state space pruning, which will be discussed later in Section 4.

3.1 Syntax of Automata Terms.

Terms are created according to the grammar

$$\begin{aligned}
 A &::= S \mid D && (\text{automata term}) \\
 S &::= \{t, \dots, t\} && (\text{set term}) \\
 D &::= S - \vec{0}^* && (\text{derivative term}) \\
 t &::= q \mid t + t \mid t \& t \mid \bar{t} \mid \pi_X(t) \mid S \mid D && (\text{state term})
 \end{aligned}$$

starting from states $q \in Q_i$, denoted as *atomic states*, of a given finite set of *base automata* $\mathcal{B}_i = (Q_i, \delta_i, I_i, R_i)$ with pairwise disjoint sets of states. For simplicity, we assume that the base automata are complete, and we denote by $\mathcal{B} = (Q^{\mathcal{B}}, \delta^{\mathcal{B}}, I^{\mathcal{B}}, R^{\mathcal{B}})$ their component-wise union. *Automata terms* A specify the set of leaf states of an automaton. *Set terms* S list a finite number of the leaf states explicitly, while *derivative terms* D specify them symbolically as states reachable from a set of states S via $\vec{0}$'s. The states themselves are represented by *state terms* t . (Notice that set terms S and derivative terms D can be both automata terms and state terms.) Intuitively, the structure of state terms records the automata constructions used to create the top-level automaton from states of the base automata. Non-leaf state terms, the state terms' transition function, and root state terms are then defined inductively from base automata as described below in detail. We will normally use t, u to denote terms of all types (unless the type of the term needs to be emphasized).

Example 1 Consider a formula $\varphi \equiv \neg \exists X. \text{Sing}(X) \wedge X = \{\epsilon\}$ and its corresponding automata term $t_\varphi = \left\{ \overline{\{\pi_X(\{q_0\} \& \{p_0\})\}} - \vec{0}^* \right\}$ (we will show how t_φ was obtained from φ later). For the sake of presentation, we will consider the base automata given in Fig. 4 for the predicates $\text{Sing}(X)$ and $X = \{\epsilon\}$. The term t_φ above denotes the TA $\left((\pi_X(\mathcal{A}_{\text{Sing}(X)} \cap \mathcal{A}_{X=\{\epsilon\}}))^{\mathcal{D}} - \vec{0}^* \right)^{\mathcal{C}}$ constructed using the automata operations of intersection, projection, subset construction, derivative, and complement. \square

3.2 Semantics of Terms.

We will define the denotation of an automata term t as the automaton $\mathcal{A}_t = (Q, \Delta, I, R)$. For a set automata term $t = S$, we define $I = S$, $Q = \text{reach}_\Delta(S)$ (i.e., Q is the set of

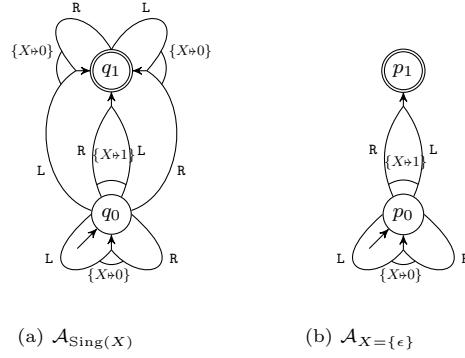


Fig. 4: Tree automata for the predicates used in Example 1

$$\begin{array}{ll}
\mathcal{R}(t+u) \Leftrightarrow \mathcal{R}(t) \vee \mathcal{R}(u) & (2) \quad \Delta_a(t+u, t'+u') = \Delta_a(t, t') [+] \Delta_a(u, u') & (8) \\
\mathcal{R}(t \& u) \Leftrightarrow \mathcal{R}(t) \wedge \mathcal{R}(u) & (3) \quad \Delta_a(t \& u, t' \& u') = \Delta_a(t, t') [\&] \Delta_a(u, u') & (9) \\
\mathcal{R}(\pi_X(t)) \Leftrightarrow \mathcal{R}(t) & (4) \quad \Delta_a(\pi_X(t), \pi_X(t')) = \{\pi_X(u) \mid u \in \Delta_{\pi_X(a)}(t, t')\} & (10) \\
\mathcal{R}(\bar{t}) \Leftrightarrow \neg \mathcal{R}(t) & (5) \quad \Delta_a(\bar{t}, t') = \{\bar{u} \mid u \in \Delta_a(t, t')\} & (11) \\
\mathcal{R}(S) \Leftrightarrow \exists t \in S. \mathcal{R}(t) & (6) \quad \Delta_a(S, S') = \left\{ \bigcup_{t \in S, t' \in S'} \Delta_a(t, t') \right\} & (12) \\
\mathcal{R}(q) \Leftrightarrow q \in R^{\mathcal{B}} & (7) \quad \Delta_a(q, r) = \delta_a^{\mathcal{B}}(q, r) & (13)
\end{array}$$

(a) Root term states

(b) Transitions among compatible state terms

Fig. 5: Semantics of terms

state terms reachable from the leaf state terms), and Δ and R are defined inductively to the structure of t . Particularly, R contains the terms of Q that satisfy the predicate \mathcal{R} defined in Fig. 5a, and Δ is defined in Fig. 5b, with the addition that whenever the rules in Fig. 5b do not apply, then we let $\Delta_a(t, t') = \{\emptyset\}$. The \emptyset here is used as a universal sink state in order to maintain Δ complete, which is needed for automata terms representing complements to yield the expected language. In Figs. 5a and 5b, the terms t, t', u, u' are arbitrary terms, S, S' are set terms, and $q, r \in Q^{\mathcal{B}}$.

The transitions of Δ for terms of the type $+$, $\&$, π_X , $\bar{\cdot}$, and S are built from the transition function of their sub-terms analogously to how the automata operations of the product union, product intersection, projection, complement, and subset construction, respectively, build the transition function from the transition functions of their arguments (cf. Section 2). The only difference is that the state terms stay *annotated* with the particular operation by which they were made (the annotation of the set state terms are the set brackets). The root states are also defined analogously as in the classical constructions.

Finally, we complete the definition of the term semantics by adding the definition of semantics for the derivative term $S - \vec{0}^*$. This term is a symbolic representation of the set term that contains all state terms upward-reachable from S in \mathcal{A}_S over $\vec{0}$.

Formally, we first define the so-called *saturation* of \mathcal{A}_S as

$$(S - \vec{0}^*)^s = \text{reach}_{\Delta_{\vec{0}}}(S) \quad (14)$$

(with $\text{reach}_{\Delta_{\vec{0}}}(S)$ defined as the fixpoint (1)), and we complete the definition of Δ and \mathcal{R} in Figs. 5a and 5b with three new rules to be used with a derivative term D :

$$\Delta_a(D, u) = \Delta_a(D^s, u) \quad (15) \quad \mathcal{R}(D) \Leftrightarrow \mathcal{R}(D^s) \quad (17)$$

$$\Delta_a(u, D) = \Delta_a(u, D^s) \quad (16)$$

The automaton \mathcal{A}_D then equals \mathcal{A}_{D^s} , i.e., the semantics of a derivative term is defined by its saturation.

Example 2 Let us consider a derivative term $t = \{\pi_X(\{q_0\} \& \{p_0\})\} - \vec{0}^*$, which occurs within the nested automata term t_φ of Example 1. The set term representing all terms reachable upward from t is then the term

$$t^s = \{\pi_X(\{q_0\} \& \{p_0\}), \pi_X(\{q_1\} \& \{p_1\}), \pi_X(\{q_s\} \& \{p_s\}), \\ \pi_X(\{q_1\} \& \{p_s\}), \pi_X(\{q_0\} \& \{p_s\})\}.$$

The semantics of t is then the automaton \mathcal{A}_t with the set of states given by t^s . \square

3.3 Properties of Terms.

In this section, we establish properties of automata terms that we will use later when establishing the correctness of our decision procedure. An implication of the definitions in the previous section, essential for termination of our algorithm in Section 4, is that the automata represented by terms indeed have finitely many states. This is a direct consequence of the following lemma.

Lemma 1 *The size of $\text{reach}_\Delta(t)$ is finite for any automata term t .*

Proof (idea) First, we define the *depth* of a term t , denoted as $d(t)$, inductively as follows: (i) $d(q) = 1$ for $q \in Q^B$, (ii) $d(t_1 \circ t_2) = 1 + \max(d(t_1), d(t_2))$ for $\circ \in \{\&, +\}$, (iii) $d(\diamond t_1) = 1 + d(t_1)$ for $\diamond \in \{\pi_X, \cdot\}$, (iv) $d(S) = 1 + \max_{t \in S}(d(t))$, and (v) $d(S - \Gamma^*) = 1 + d(S)$. Then, since the number of reachable states in base automata is finite, for a given n there is a finite number of terms of depth at most n . By induction on the depth of terms, we can show that for a pair of terms t_1 and t_2 , it holds that for each $t \in \Delta_a(t_1, t_2)$ we have $d(t) \leq \max(d(t_1), d(t_2))$. Therefore, for an automata term S it holds that $\text{reach}_\Delta(S)$ is finite. \square

Let us denote by $\mathcal{L}(t)$ the language $\mathcal{L}(\mathcal{A}_t)$ of the automaton induced by a term t . In the following, we often use the notions of a term expansion and an expanded term. An *expanded term* is a term that does not contain a derivative term as a subterm. *Term expansion* is then defined recursively as follows: (i) $t^e = t$ if t is expanded and (ii) $t^e = (t[u/u^s])^e$ where u is a derivative term of the form $S - \Gamma^*$ for an expanded term S . Intuitively, the term expansion saturates derivative subterms in a bottom-up manner. Note that the expansion of any automata term A is a set term, i.e., $A^e = \{t_1, \dots, t_n\}$.

Lemma 2 *Given an automata term t and its expanded term t^e , it holds that*

- (i) t^e is of a finite size and
- (ii) $\mathcal{L}(t^e) = \mathcal{L}(t)$.

Proof (idea) (i): This can be easily seen from the fact that term expansion is performed by a bottom-up traversal on the structure of t while substituting derivative terms with their saturations. From the definition of saturation in (14) and Lemma 1, it follows that each such saturation is finite.

(ii): First, note that saturation preserves language, i.e., it holds that

$$\mathcal{L}\left((S - \vec{0}^*)\right) = \mathcal{L}\left((S - \vec{0}^*)^s\right). \quad (18)$$

The previous fact follows from the definition of derivative automaton in Section 2.4. In particular, given $\mathcal{A}_S = (Q, \Delta, S, R)$, we have that

$$\mathcal{A}_S - \vec{0}^* = (Q, \Delta, \text{reach}_{\Delta_{\vec{0}}}(S), R), \quad (19)$$

which matches the definition of saturation in (14). The lemma follows from the fact that the expansion substitutes terms for saturated terms with equal languages. \square

Lemma 3 below shows that languages of terms can be defined from the languages of their sub-terms if the sub-terms are set terms of derivative terms. The terms on the left-hand sides are implicit representations of the automata operations of the respective language operators on the right-hand sides. The main reason why the lemma cannot be extended to all types of sub-terms and yield an inductive definition of term languages is that it is not meaningful to talk about the bottom-up language of an isolated state term that is neither a set term nor a derivative term (which both are also automata terms). This is also one of the main differences from [13], where every term has its own language, which makes the reasoning and the correctness proofs in the current paper significantly more involved.

Lemma 3 *For automata terms A_1, A_2 and a set term S , the following equalities hold:*

$$\begin{aligned} \mathcal{L}(\{A_1\}) &= \mathcal{L}(A_1) & (a) & \quad \mathcal{L}(\{\overline{A_1}\}) = \overline{\mathcal{L}(A_1)} & (d) \\ \mathcal{L}(\{A_1 + A_2\}) &= \mathcal{L}(A_1) \cup \mathcal{L}(A_2) & (b) & \quad \mathcal{L}(\{\pi_X(A_1)\}) = \pi_X(\mathcal{L}(A_1)) & (e) \\ \mathcal{L}(\{A_1 \& A_2\}) &= \mathcal{L}(A_1) \cap \mathcal{L}(A_2) & (c) & \quad \mathcal{L}(S - \vec{0}^*) = \mathcal{L}(S) - \vec{0}^* & (f) \end{aligned}$$

Proof (a): We prove the following more general form of (a):

$$\mathcal{L}(\{A_1, \dots, A_n\}) = \mathcal{L}\left(\bigcup_{1 \leq i \leq n} A_i^e\right). \quad (20)$$

(Note that A_1, \dots, A_n are automata terms—i.e., either set terms or derivative terms—so their expanded terms will be set terms.) Intuitively, in this proof we show that determinisation does not change the language of a term. Let us use $\mathcal{A}_{\bigcup_{1 \leq i \leq n} A_i^e}$ to denote the TA represented by the term $\bigcup_{1 \leq i \leq n} A_i^e$.

(\subseteq) Let τ be a tree. It holds that $\tau \in \mathcal{L}(\{A_1, \dots, A_n\})$ if and only if $\tau \in \mathcal{L}(\{A_1^e, \dots, A_n^e\})$, i.e., if there is an accepting run ρ on τ in $\mathcal{A}_{\{A_1^e, \dots, A_n^e\}}$. Note that ρ

maps all leaves of τ to the terms from $\{A_1^e, \dots, A_n^e\}$, i.e., each leaf of τ is labelled by some A_i^e , which is a *set* of terms of a lower level (such a set term can be seen as a *macrostate*—i.e., a set of states—from determinisation of TAs). Moreover, for all non-leaf positions $w \in \text{dom}(\tau) \setminus \text{leaf}(\tau)$, let $\rho(w) = U$, $\rho(w.L) = U_L$, and $\rho(w.R) = U_R$. Then, from (12), we have that if $u \in U$, then there exist $u_L \in U_L$ and $u_R \in U_R$ such that $u \in \Delta_{\tau(w)}(u_L, u_R)$. Let us define an auxiliary function $\mu(w, u) = (u_L, u_R)$ that we will use later. Since ρ is accepting, there is a term $r \in \rho(\epsilon)$ such that $\mathcal{R}(r)$.

We will now use ρ to construct a run ρ' of $\mathcal{A}_{\bigcup A_i^e}$ on τ . The run ρ' will now map positions to a single term as follows: For the root position, we set $\rho'(\epsilon) = r$. Then, given $w \in \text{dom}(\tau) \setminus \text{leaf}(\tau)$, the labels of children of w are defined as $\rho'(w.L) = u_L$ and $\rho'(w.R) = u_R$ where $(u_L, u_R) = \mu(w, \rho'(w))$. As a consequence, we have that $\forall w \in \text{leaf}(\tau) : \rho'(w) \in \bigcup_{1 \leq i \leq n} A_i^e$. Then, for each $w \in \text{dom}(\tau)$, it holds that $\rho'(w) \in \text{reach}_\Delta(\bigcup_{1 \leq i \leq n} A_i^e)$ where Δ is the transition function of $\mathcal{A}_{\bigcup A_i^e}$. Therefore, ρ' is a run of $\mathcal{A}_{\bigcup A_i^e}$ on τ and is accepting, so $\tau \in \mathcal{L}\left(\bigcup_{1 \leq i \leq n} A_i^e\right)$.

(\supseteq) Consider a tree $\tau \in \mathcal{L}\left(\bigcup_{1 \leq i \leq n} A_i^e\right)$. Then there is an accepting run ρ on τ in $\mathcal{A}_{\bigcup A_i^e}$. We can then use ρ to construct the run ρ' on $\text{dom}(\tau)$ defined as follows: For $u \in \text{leaf}(\tau)$, if $\rho(u) \in A_i^e$, we set $\rho'(u) = A_i^e$. For $w \in \text{dom}(\tau) \setminus \text{leaf}(\tau)$, we set $\rho'(w) = r$ such that $\{r\} = \Delta_{\tau(w)}(\rho'(w.L), \rho'(w.R))$ (we know that $\Delta_{\tau(w)}(\rho'(w.L), \rho'(w.R))$ is a singleton set due to (12)). For the constructed run ρ' , it now holds that $\forall w \in \text{dom}(\tau) : \rho(w) \in \rho'(w)$, therefore ρ' is an accepting run on τ in $\mathcal{A}_{\{A_1^e, \dots, A_n^e\}}$, i.e., $\tau \in \mathcal{L}(\{A_1, \dots, A_n\})$.

(b): (\subseteq) Let $\tau \in \mathcal{L}(\{A_1 + A_2\})$. Then there is an accepting run ρ on τ in $\mathcal{A}_{\{A_1^e + A_2^e\}}$. Since ρ is accepting, we can define mappings ρ_1, ρ_2 on $\text{dom}(\tau)$ such that for all $w \in \text{dom}(\tau)$ we have $\rho_1(w) = l(\rho(w))$ and $\rho_2(w) = r(\rho(w))$ where $l(S_1 + S_2) = S_1$ and $r(S_1 + S_2) = S_2$. The mappings ρ_1 and ρ_2 are runs of $\mathcal{A}_{\{A_1^e\}}$ and $\mathcal{A}_{\{A_2^e\}}$ on τ respectively. Moreover, since $\mathcal{R}(\rho(\epsilon))$, we have that $\mathcal{R}(\rho_1(\epsilon)) \vee \mathcal{R}(\rho_2(\epsilon))$. To conclude, $\tau \in \mathcal{L}\left(\mathcal{A}_{\{A_1^e\}}\right)$ or $\tau \in \mathcal{L}\left(\mathcal{A}_{\{A_2^e\}}\right)$, so $\tau \in \mathcal{L}(\{A_1\}) \cup \mathcal{L}(\{A_2\})$ and from (a) we get $\tau \in \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$.

(\supseteq) Consider $\tau \in \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$. From (a) we get $\tau \in \mathcal{L}(\{A_1\}) \cup \mathcal{L}(\{A_2\})$. Then there are runs ρ_1 in $\mathcal{A}_{\{A_1^e\}}$ and ρ_2 in $\mathcal{A}_{\{A_2^e\}}$ on τ such that at least one of them is accepting. We can define a mapping ρ on $\text{dom}(\tau)$ such that $\forall w \in \text{dom}(\tau) : \rho(w) = \rho_1(w) + \rho_2(w)$, which is an accepting run on τ in $\mathcal{A}_{\{A_1^e + A_2^e\}}$. Therefore $\tau \in \mathcal{L}(\{A_1 + A_2\})$.

(c): Dual to (b).

(d): Let τ be a tree. We will consider runs ρ and $\bar{\rho}$ of $\mathcal{A}_{\{A_1^e\}}$ and $\mathcal{A}_{\{\overline{A_1^e}\}}$ on τ respectively. First, note that both runs exist, which is guaranteed by the presence of the universal sink state \emptyset , cf. Section 3.2. Second, note that the two runs are unique, since there is a single leaf state and the transition function is deterministic by (12). Further, from (11), it holds that $\forall w \in \text{dom}(\tau) : \bar{\rho}(w) = \overline{\rho(w)}$. From the definition of \mathcal{R} we have $\mathcal{R}(\bar{\rho}(\epsilon)) \Leftrightarrow \neg \mathcal{R}(\rho(\epsilon))$, therefore, ρ is not accepting in $\mathcal{A}_{\{A_1^e\}}$ if and only if $\bar{\rho}$ is accepting in $\mathcal{A}_{\{\overline{A_1^e}\}}$. As a consequence, $\tau \in L(\{\overline{A_1^e}\})$ if and only if $\tau \notin L(\{A_1^e\})$. From (a), we know that $L(\{A_1^e\}) = L(A_1^e)$.

(e): (\subseteq) Let $\tau \in \mathcal{L}(\{\pi_X(A_1)\})$ and ρ be an accepting run of $\mathcal{A}_{\{\pi_X(A_1^e)\}}$ on τ . From the definition of the transition function in (10) and (4), we get that there is an accepting

run ρ' on some τ' in $\mathcal{A}_{\{A_1^\dagger\}}$ where $\tau \in \pi_X(\tau')$ and $\forall w \in \text{dom}(\tau) : \rho(w) = \pi_X(\rho'(w))$. Therefore, $\tau \in \pi_X(\mathcal{L}(\{A_1^\dagger\})) = \pi_X(\mathcal{L}(A_1))$.

(\supseteq) Let $\tau \in \pi_X(\mathcal{L}(A_1))$. From the definition of projection, there is $\tau' \in \mathcal{L}(A_1)$ such that $\tau \in \pi_X(\tau')$. According to (a), there is an accepting run ρ on τ' in $\mathcal{A}_{\{A_1^\dagger\}}$. Then there is also an accepting run ρ' on τ in $\mathcal{A}_{\{\pi_X(A_1^\dagger)\}}$ where $\forall w \in \text{dom}(\tau) : \rho'(w) = \pi_X(\rho(w))$.

(f): We prove the following more general equality: $\mathcal{L}(S) - \Gamma^* = \mathcal{L}(S - \Gamma^*)$, for a set of symbols Γ (note that S is a set term). In the following text, given a set term U , we define $U \ominus \Gamma = U^e \cup \bigcup \{\Delta_\Gamma(t_1, t_2) \mid t_1, t_2 \in U^e\}$. Note that $\text{reach}_\Delta(U^e) = \text{reach}_\Delta(U \ominus \Gamma)$. Further, we use $\Gamma^{\leq n}$ to denote the set of trees over Γ of height at most n , i.e., $\Gamma^{\leq n} = \{t \in \Gamma^* \mid \forall w \in \text{dom}(t) : |w| \leq n\}$. We first prove the following two claims.

Claim 1 Let U be a set term. Then $\mathcal{L}(U \ominus \Gamma) = \mathcal{L}(U) - \Gamma^{\leq 1}$.

Proof (\subseteq) Let $\tau \in \mathcal{L}(U \ominus \Gamma)$ and ρ be an accepting run of $\mathcal{A}_{U \ominus \Gamma}$ on τ . The run ρ maps leaves of τ to the leaf states in $U \ominus \Gamma$. Moreover, for each $w \in \text{leaf}(\tau)$ such that $\rho(w) \notin U^e$ (i.e., ρ maps w to a newly added leaf state) there exist $t_L^w, t_R^w \in U^e$ such that $\rho(w) \in \Delta_\Gamma(t_L^w, t_R^w)$. We can therefore extend ρ to the run ρ' defined such that $\rho'_{|\text{dom}(\tau)} = \rho$ and for all $w \in \text{leaf}(\tau)$ such that $\rho(w) \notin U^e$, we define $\rho'(w.L) = t_L^w$ and $\rho'(w.R) = t_R^w$. The run ρ' is accepting in \mathcal{A}_{U^e} on a tree $\tau' \in \mathcal{L}(U)$ such that $\tau \in \tau' - \Gamma^{\leq 1}$, and so $\tau \in \mathcal{L}(U) - \Gamma^{\leq 1}$.

(\supseteq) Let $\tau \in \mathcal{L}(U) - \Gamma^{\leq 1}$ and $\tau' \in \mathcal{L}(U)$ be a tree such that $\tau \in \tau' - \Gamma^{\leq 1}$. Hence there is an accepting run ρ' of \mathcal{A}_{U^e} on τ' . Consider the set $\Theta = \{w \in \text{leaf}(\tau) \mid \rho'(w) \notin U^e\}$ of positions mapped by ρ' to newly added states. Since $\tau \in \tau' - \Gamma^{\leq 1}$, it holds that $\forall w \in \Theta : \rho'(w.L) \in U^e \wedge \rho'(w.R) \in U^e \wedge \tau'(w) \in \Gamma$. Therefore, $\rho = \rho'_{|\text{dom}(\tau)}$ is an accepting run of $\mathcal{A}_{U \ominus \Gamma}$ on τ , i.e., $\tau \in \mathcal{L}(U \ominus \Gamma)$. ■

Claim 2 Let U be a set term, $U_0 = U$, and $U_{i+1} = U_i \ominus \Gamma$ for $i \geq 0$. Then $\mathcal{L}(U_m) = \mathcal{L}(U) - \Gamma^{\leq m}$.

Proof We prove the claim by induction on m .

- *Base case* $m = 0$: $\mathcal{L}(U_0) = \mathcal{L}(U) = \mathcal{L}(U) - \Gamma^{\leq 0}$.
- *Inductive case*: We assume that the claim holds for $0, \dots, m$. We prove that it holds also for $m + 1$. From Claim 1 we have

$$\mathcal{L}(U_{m+1}) = \mathcal{L}(U_m \ominus \Gamma) = \mathcal{L}(U_m) - \Gamma^{\leq 1}. \quad (21)$$

By the induction hypothesis we further have

$$\mathcal{L}(U_{m+1}) = (\mathcal{L}(U) - \Gamma^{\leq m}) - \Gamma^{\leq 1}. \quad (22)$$

Finally, from the definition of the derivative we obtain

$$(\mathcal{L}(U) - \Gamma^{\leq m}) - \Gamma^{\leq 1} = \mathcal{L}(U) - \Gamma^{\leq m+1}, \quad (23)$$

which concludes the proof. ■

We now prove the main part of the lemma. Consider the sequence of automata terms S_0, S_1, \dots where $S_0 = S^\epsilon$ and $S_{i+1} = S_i \ominus \Gamma$. From the monotonicity of \ominus and Lemma 1, there is an n such that $S_n \neq S_{n-1}$ and $S_n = S_{n+i}$ for all $i \geq 0$. From Claim 2 we have $\mathcal{L}(S_i) = \mathcal{L}(S) - \Gamma^{\leq i}$ and, consequently, $\mathcal{L}(S_n) = \mathcal{L}(S) - \Gamma^{\leq n}$. Because S_n is the fixpoint of the sequence of automata terms S_0, S_1, \dots , it holds that $\mathcal{L}(S_n) = \mathcal{L}(S) - \Gamma^*$. Finally, we have $S_n = \text{reach}_{\Delta_\Gamma}(S^\epsilon) = S - \Gamma^*$ (by (14)), so we conclude that $\mathcal{L}(S) - \Gamma^* = \mathcal{L}(S - \Gamma^*)$. \square

Lemma 3 shows fundamental properties of terms. Based on it we further focus on flattening of terms, whose properties are described by the following lemma.

Lemma 4 *For sets of terms S and S' such that $S \neq \emptyset$ and $S' \neq \emptyset$, we have:*

$$\mathcal{L}(\{S + S'\}) = \mathcal{L}(\{S[+] S'\}), \quad (a)$$

$$\mathcal{L}(\{S \& S'\}) = \mathcal{L}(\{S[\&] S'\}), \quad (b)$$

$$\mathcal{L}(\{\pi_X(S)\}) = \mathcal{L}(\{\pi_X(t) \mid t \in S\}). \quad (c)$$

Proof (a): (\subseteq) Let $\tau \in \mathcal{L}(\{S + S'\})$. From Lemma 3b we have $\mathcal{L}(\{S + S'\}) = \mathcal{L}(S) \cup \mathcal{L}(S')$. Hence there are runs ρ_1 in \mathcal{A}_{S^ϵ} and ρ_2 in $\mathcal{A}_{S'^\epsilon}$ on τ and, moreover, at least one of them is accepting (both runs exist since the transition function Δ is complete). Then, we can construct a mapping ρ from τ defined such that for all $w \in \text{dom}(\tau)$, we set $\rho(w) = \rho_1(w) + \rho_2(w)$. Note that ρ is a run of $\mathcal{A}_{\{t_1^\epsilon + t_2^\epsilon \mid t_1 \in S, t_2 \in S'\}}$ on τ , i.e., it maps leaves of $\text{dom}(\tau)$ to terms of the form $t_1^\epsilon + t_2^\epsilon$ for $t_1 \in S$ and $t_2 \in S'$. Moreover, ρ is accepting since at least one of the runs ρ_1 and ρ_2 is accepting. Therefore, $\tau \in \mathcal{L}(\{t_1 + t_2 \mid t_1 \in S, t_2 \in S'\})$. From the definition of the augmented product, it follows that $\tau \in \mathcal{L}(S[+] S')$ and, finally, from Lemma 3a, we have $\tau \in \mathcal{L}(\{S[+] S'\})$.

(\supseteq) Let $\tau \in \mathcal{L}(\{S[+] S'\})$. From Lemma 3a, we get $\tau \in \mathcal{L}(S[+] S')$, and from the definition of the augmented product, we obtain that $\tau \in \mathcal{L}(\{t_1 + t_2 \mid t_1 \in S, t_2 \in S'\})$. Therefore, there is an accepting run ρ on τ in $\mathcal{A}_{\{t_1^\epsilon + t_2^\epsilon \mid t_1 \in S, t_2 \in S'\}}$. Furthermore, let us consider the run ρ' of $\mathcal{A}_{\{S + S'\}}$ on τ (note that, due to (12) and the completeness of the transition function, there is exactly one). By induction on the structure of τ , we can easily show that for all $w \in \text{dom}(\tau)$, if $\rho(w) = t_1 + t_2$, then $\rho'(w) = S_1 + S_2$ such that $t_1 \in S_1$ and $t_2 \in S_2$ (the property clearly holds at leaves and is also preserved by the transition function). Let $\rho(\epsilon) = t_1^\epsilon + t_2^\epsilon$ and $\rho'(\epsilon) = S_1^\epsilon + S_2^\epsilon$. Since $\mathcal{R}(t_1^\epsilon + t_2^\epsilon)$, it also holds that $\mathcal{R}(S_1^\epsilon + S_2^\epsilon)$. Therefore, ρ' is accepting, so $\tau \in \mathcal{L}(\{S + S'\})$.

(b): Dual to (a).

(c): From Lemma 3e we have that $\mathcal{L}(\{\pi_X(S)\}) = \pi_X(\mathcal{L}(S))$. Therefore, it is sufficient to prove the following identity: $\pi_X(\mathcal{L}(S)) = \mathcal{L}(\{\pi_X(t) \mid t \in S\})$.

(\subseteq) Let $\tau \in \pi_X(\mathcal{L}(S))$. Then, there is a tree $\tau' \in \mathcal{L}(S)$ such that $\tau \in \pi_X(\tau')$. Let ρ be an accepting run of \mathcal{A}_{S^ϵ} on τ' . We will construct a run ρ' of $\mathcal{A}_{\{\pi_X(t) \mid t \in S^\epsilon\}}$ on τ' such that for all $w \in \text{dom}(\tau)$, we set $\rho'(w) = \pi_X(\rho(w))$. It follows that $\tau \in \mathcal{L}(\{\pi_X(t) \mid t \in S\})$.

(\supseteq) Let $\tau \in \mathcal{L}(\{\pi_X(t) \mid t \in S\})$ and ρ be an accepting run of $\mathcal{A}_{\{\pi_X(t) \mid t \in S^\epsilon\}}$ on τ . We will now construct a mapping ρ' from $\text{dom}(\tau)$ such that for all $w \in \text{dom}(\tau)$, we set $\rho'(w) = t$ where $\rho(w) = \pi_X(t)$. It follows that ρ' is an accepting run of \mathcal{A}_{S^ϵ} on τ' , and so $\tau \in \pi_X(\mathcal{L}(S))$. \square

3.4 Terms of Formulae.

Our algorithm in Section 4 will translate a WS2S formula φ into the automata term $t_\varphi = \{\langle\varphi\rangle\}$ representing a deterministic automaton with its only leaf state represented by the state term $\langle\varphi\rangle$. The base automata of t_φ include the automaton \mathcal{A}_{φ_0} for each atomic predicate φ_0 used in φ . The state term $\langle\varphi\rangle$ is then defined inductively to the structure of φ as follows:

$$\langle\varphi_0\rangle = I_{\varphi_0} \quad (24)$$

$$\langle\varphi \wedge \psi\rangle = \langle\varphi\rangle \& \langle\psi\rangle \quad (25)$$

$$\langle\varphi \vee \psi\rangle = \langle\varphi\rangle + \langle\psi\rangle \quad (26)$$

$$\langle\neg\varphi\rangle = \overline{\langle\varphi\rangle} \quad (27)$$

$$\langle\exists X. \varphi\rangle = \{\pi_X(\langle\varphi\rangle)\} - \vec{0}^* \quad (28)$$

In the definition, φ_0 is an atomic predicate, I_{φ_0} is the set of leaf states of \mathcal{A}_{φ_0} , and φ and ψ denote arbitrary WS2S formulae. We note that the translation rules may create sub-terms of the form $\{\{t\}\}$, i.e., with nested set brackets. Since $\{\cdot\}$ semantically means determinisation by subset construction, such double determinisation terms can be always simplified to $\{t\}$ (cf. Lemma 3a). See Example 1 for a formula φ and its corresponding term t_φ . Theorem 1 establishes the correctness of the formula-to-term translation.

Theorem 1 *Let φ be a WS2S formula. Then $\mathcal{L}(\varphi) = \mathcal{L}(t_\varphi)$.*

Proof To simplify the proof, we restrict the definition of terms to *deterministic terms* U constructed using the following grammar:

$$U ::= \{u, \dots, u\} \mid \{\pi_X(u), \dots, \pi_X(u)\} \quad (29)$$

$$u ::= q \mid u + u \mid u \& u \mid \bar{u} \mid U \mid U - \Gamma^* \quad (30)$$

where q is a state of an automaton. It is easy to see that deterministic terms form a proper subset of all terms constructed using the definition in Section 3.1 (e.g., the term $\pi_X(t_1) \& \pi_X(t_2)$ is not deterministic). They are, however, sufficient to capture the terms that emerge from the translation presented above. Note that for two expanded deterministic terms t_1 and t_2 we have $|\Delta_a(t_1, t_2)| = 1$. Further note that for a WS2S formula φ , $\langle\varphi\rangle$ is a deterministic term.

Now, we prove $\mathcal{L}(\varphi) = \mathcal{L}(\{\langle\varphi\rangle\})$ by induction on the structure of φ . In the proof, we use properties of the classical decision procedure from Section 2.4.

– $\varphi = \varphi_0$ where φ_0 is an atomic formula: Let I_{φ_0} be the set of leaf states of \mathcal{A}_{φ_0} .

$$\begin{aligned} \mathcal{L}(\{\langle\varphi_0\rangle\}) &= \mathcal{L}(\{I_{\varphi_0}\}) && \text{[(24)]} \\ &= \mathcal{L}(I_{\varphi_0}) && \text{[Lemma 3a]} \\ &= \mathcal{L}(\mathcal{A}_{\varphi_0}) && \text{[term semantics]} \\ &= \mathcal{L}(\varphi_0). && \text{[property of automata for atoms]} \end{aligned}$$

- $\varphi = \psi_1 \wedge \psi_2$: We use the following equational reasoning:

$$\begin{aligned}
\mathcal{L}(\{\langle \psi_1 \wedge \psi_2 \rangle\}) &= \mathcal{L}(\{\langle \psi_1 \rangle \& \langle \psi_2 \rangle\}) && \text{[(25)]} \\
&= \mathcal{L}(\{\{\langle \psi_1 \rangle \& \langle \psi_2 \rangle\}\}) && \text{[Lemma 3a]} \\
&= \mathcal{L}(\{\{\langle \psi_1 \rangle\} \& \{\langle \psi_2 \rangle\}\}) && \text{[Lemma 4b]} \\
&= \mathcal{L}(\{\langle \psi_1 \rangle\}) \cap \mathcal{L}(\{\langle \psi_2 \rangle\}). && \text{[Lemma 3c]} \\
&= \mathcal{L}(\psi_1) \cap \mathcal{L}(\psi_2) && \text{[induction hypothesis]} \\
&= \mathcal{L}(\varphi). && \text{[classical procedure]}
\end{aligned}$$

- $\varphi = \psi_1 \vee \psi_2$: We use the following equational reasoning:

$$\begin{aligned}
\mathcal{L}(\{\langle \psi_1 \vee \psi_2 \rangle\}) &= \mathcal{L}(\{\langle \psi_1 \rangle + \langle \psi_2 \rangle\}) && \text{[(26)]} \\
&= \mathcal{L}(\{\{\langle \psi_1 \rangle + \langle \psi_2 \rangle\}\}) && \text{[Lemma 3a]} \\
&= \mathcal{L}(\{\{\langle \psi_1 \rangle\} + \{\langle \psi_2 \rangle\}\}) && \text{[Lemma 4a]} \\
&= \mathcal{L}(\{\langle \psi_1 \rangle\}) \cup \mathcal{L}(\{\langle \psi_2 \rangle\}). && \text{[Lemma 3b]} \\
&= \mathcal{L}(\psi_1) \cup \mathcal{L}(\psi_2) && \text{[induction hypothesis]} \\
&= \mathcal{L}(\varphi). && \text{[classical procedure]}
\end{aligned}$$

- $\varphi = \neg\psi$: First, we prove the following claim:

Claim 3 Let t be a deterministic term, then $\mathcal{L}(\{\overline{\{t\}}\}) = \mathcal{L}(\{\bar{t}\})$.

Proof First, consider two expanded deterministic terms t_1 and t_2 . Since t_1 and t_2 are deterministic, from (12) we have $\Delta_a(t_1, t_2) = \{t'\}$ for some deterministic term t' and any symbol a . Therefore (from (11)), $\Delta_a(\overline{t_1}, \overline{t_2}) = \{\bar{t}'\}$ and $\Delta_a(\overline{\{t_1\}}, \overline{\{t_2\}}) = \{\overline{\{t'\}}\}$. Hence, there is an accepting run ρ on a tree τ in $\mathcal{A}_{\{\overline{\{t\}}\}}$ if and only if there is an accepting run ρ' on τ in $\mathcal{A}_{\{\bar{t}\}}$ where for all $w \in \text{dom}(\tau)$ it holds that $\rho(w) = \bar{s} \Leftrightarrow \rho'(w) = \{s\}$. ■

We proceed to the main part of the proof.

$$\begin{aligned}
\mathcal{L}(\{\langle \neg\psi \rangle\}) &= \mathcal{L}(\{\overline{\langle \psi \rangle}\}) && \text{[(27)]} \\
&= \mathcal{L}(\{\overline{\{\langle \psi \rangle\}}\}) && \text{[Claim 3]} \\
&= \overline{\mathcal{L}(\{\langle \psi \rangle\})} && \text{[Lemma 3d]} \\
&= \overline{\mathcal{L}(\psi)} && \text{[induction hypothesis]} \\
&= \mathcal{L}(\varphi). && \text{[classical procedure]}
\end{aligned}$$

- $\varphi = \exists X. \psi$: We start by proving the following claim:

Claim 4 Let t be a deterministic term, then $\mathcal{L}(\{\pi_X(\{t\})\}) = \mathcal{L}(\{\pi_X(t)\})$.

Proof First, consider two expanded deterministic terms t_1 and t_2 . Since t_1 and t_2 are both deterministic, we have $\Delta_a(t_1, t_2) = \{t_a\}$ for some deterministic term t_a and any symbol a . Therefore, according to (10), $\Delta_a(\pi_X(t_1), \pi_X(t_2)) = \{\pi_X(t_b) \mid b \in \pi_X(a)\}$ and $\Delta_a(\pi_X(\{t_1\}), \pi_X(\{t_2\})) = \{\pi_X(\{t_b\}) \mid b \in \pi_X(a)\}$. Hence, there is an accepting run ρ on a tree τ in $\mathcal{A}_{\{\pi_X(\{t\})\}}$ if and only if there is an accepting

run ρ' on τ in $\mathcal{A}_{\{\pi_X(t)\}}$, where for all $w \in \text{dom}(\tau)$ it holds that $\rho(w) = \pi_X(s) \Leftrightarrow \rho'(w) = \pi_X(\{s\})$. ■

We proceed to the main part of the proof.

$$\begin{aligned}
\mathcal{L}(\{\langle \exists X. \psi \rangle\}) &= \mathcal{L}(\{\pi_X(\langle \psi \rangle)\} - \vec{0}^*) && \text{[(28)]} \\
&= \mathcal{L}(\{\pi_X(\langle \psi \rangle)\}) - \vec{0}^* && \text{[Lemma 3f]} \\
&= \mathcal{L}(\{\pi_X(\{\langle \psi \rangle\})\}) - \vec{0}^* && \text{[Claim 4]} \\
&= \pi_X(\mathcal{L}(\{\langle \psi \rangle\})) - \vec{0}^* && \text{[Lemma 3e]} \\
&= \pi_X(\mathcal{L}(\psi)) - \vec{0}^* && \text{[induction hypothesis]} \\
&= \mathcal{L}(\varphi). && \text{[classical procedure]} \quad \square
\end{aligned}$$

4 An Efficient Decision Procedure

The development in Section 3 already implies a naive automata term-based satisfiability check. Namely, by Theorem 1, we know that a formula φ is satisfiable if and only if $\mathcal{L}(\mathcal{A}_{t_\varphi}) \neq \emptyset$. After translating φ into t_φ using rules (24)–(28), we may use the definitions of the transition function and root states of $\mathcal{A}_{t_\varphi} = (Q, \Delta, I, F)$ in Section 3 to decide the language emptiness through evaluating the root state test $\mathcal{R}(\text{reach}_\Delta(I))$. The equalities and equivalences (8)–(17) can be implemented as recursive functions. We will further refer to this algorithm as the *simple recursion*. The evaluation of $\text{reach}_\Delta(I)$ induces nested evaluations of the fixpoint (14): the one on the top level of the language emptiness test and another one for every expansion of a derivative sub-term. The termination of these fixpoint computations is guaranteed due to Lemma 1.

Such a naive implementation is, however, inefficient and has only disadvantages in comparison to the classical decision procedure. In this section, we will discuss how it can be optimized. Besides an essential *memoization* needed to implement the recursion efficiently, we will show that the automata term representation is amenable to optimizations that cannot be used in the classical construction. These are techniques of state space pruning: the fact that the emptiness can be tested on the fly during the automata construction allows one to avoid exploration of state space irrelevant to the test. The pruning is done through the techniques of *lazy evaluation* and *subsumption*. We will also discuss optimizations of the transition function of Section 3 through *product flattening* and *nondeterministic union*, which are analogous to standard implementations of automata intersection and union.

4.1 Memoization

The simple recursion repeats the fixpoint computations that saturate derivative terms from scratch at every call of the transition function or root test. This is easily countered through *memoization*, known, e.g., from compilers of functional languages, which caches results of function calls in order to avoid their re-evaluation. Namely, after saturating a derivative sub-term $t = S - \vec{0}^*$ of t_φ for the first time, we simply *replace* t in t_φ by the saturation $t^s = \text{reach}_{\Delta_{\vec{0}}}(S)$. Since a derivative is a symbolic representation of its saturated version (cf. (14)), the replacement does not change the language of t_φ . Using memoization, every fixpoint computation is then carried out only once.

4.2 Lazy Evaluation

The *lazy* variant of the procedure uses *short-circuiting* to optimize connectives \wedge and \vee , and *early termination* to optimize fixpoint computation in derivative saturations. Namely, assume that we have a term $t_1 + t_2$ and that we test whether $\mathcal{R}(t_1 + t_2)$. Suppose that we establish that $\mathcal{R}(t_1)$; we can *short circuit* the evaluation and immediately return *true*, completely avoiding touching the potentially complex term t_2 . Similarly for a term of the form $t_1 \& t_2$, where we can short circuit the evaluation when one branch is *false*.

Furthermore, *early termination* is used to optimize fixpoint computations used to saturate derivatives within tests $\mathcal{R}(S - \vec{0}^*)$ (obtained from sub-formulae such as $\exists X. \psi$). Namely, instead of first unfolding the whole fixpoint into a set $\{t_1, \dots, t_n\}$ and only then testing whether $\mathcal{R}(t_i)$ is true for some t_i , the terms t_i can be tested as soon as they are computed, and the fixpoint computation can be stopped early, immediately when the test succeeds on one of them. Then, instead of replacing the derivative sub-term by its full saturation, we replace it by the partial result $\{t_1, \dots, t_i\} - \vec{0}^*$ for $i \leq n$. Finishing the evaluation of the fixpoint computation might later be required in order to compute a transition from the derivative. We note that this corresponds to the concept of *continuations* from functional programming, used to represent a paused computation that may be required to continue later.

Example 3 Let us now illustrate the lazy decision procedure on our running example formula $\varphi \equiv \neg \exists X. \text{Sing}(X) \wedge X = \{\epsilon\}$ and the corresponding automata term $t_\varphi = \{ \{ \pi_X(\{q_0\} \& \{p_0\}) \} - \vec{0}^* \}$ from Example 1. The task of the procedure is to compute the value of $\mathcal{R}(\text{reach}_\Delta(t_\varphi))$, i.e., whether there is a root state reachable from the leaf state $\langle \varphi \rangle$ of \mathcal{A}_{t_φ} . The fact that φ is ground allows us to slightly simplify the problem because any ground formula ψ is satisfiable if and only if $\perp \in \mathcal{L}(\psi)$, i.e., if and only if the leaf state $\langle \psi \rangle$ of \mathcal{A}_{t_ψ} is also a root. It is thus enough to test $\mathcal{R}(\langle \varphi \rangle)$ where $\langle \varphi \rangle = \{ \pi_X(\{q_0\} \& \{p_0\}) \} - \vec{0}^*$.

The computation proceeds as follows. First, we use (5) from Fig. 5a to propagate the root test towards the derivative, i.e., to obtain that $\mathcal{R}(\langle \varphi \rangle)$ if and only if $\neg \mathcal{R}(\{ \pi_X(\{q_0\} \& \{p_0\}) \} - \vec{0}^*)$. Since the \mathcal{R} -test cannot be directly evaluated on a derivative term, we need to start saturating it into a set term, evaluating \mathcal{R} on the fly, hoping for early termination. We begin with evaluating the \mathcal{R} -test on the initial element $t_0 = \pi_X(\{q_0\} \& \{p_0\})$ of the set. The test propagates through the projection π_X due to (4) and evaluates as *false* on the left conjunct (through, in order, (3), (6), and (7)) since the state q_0 is not a root state. As a trivial example of short circuiting, we can skip evaluating \mathcal{R} on the right conjunct $\{p_0\}$ and conclude that $\mathcal{R}(t_0)$ is *false*.

The fixpoint computation then continues with the first iteration, computing the $\vec{0}$ -successors of the set $\{t_0\}$. We will obtain the set $\Delta_{\vec{0}}(t_0, t_0) = \{t_0, t_1\}$ with $t_1 = \pi_X(\{q_1\} \& \{p_1\})$. The test $\mathcal{R}(t_1)$ now returns *true* because both q_1 and p_1 are root states. With that, the fixpoint computation may terminate early, with the \mathcal{R} -test on the derivative sub-term returning *true*. Memoization then replaces the derivative sub-term in $\langle \varphi \rangle$ by the partially evaluated version $\{t_0, t_1\} - \vec{0}^*$, and $\mathcal{R}(\langle \varphi \rangle)$ is evaluated as *false* due to (5). We therefore conclude that φ is unsatisfiable (and invalid since it is ground). \square

4.3 Subsumption

The next technique we use is based on pruning out parts of a search space that are *subsumed* by other parts. In particular, we generalize (in a similar way as we did for WS1S in our previous work [13]) the concept used in *antichain* algorithms for efficiently deciding language inclusion and universality of finite word and tree automata [11, 39, 5, 1]. Although the problems are in general computationally infeasible (they are **PSPACE**-complete for finite word automata and **EXPTIME**-complete for finite tree automata), antichain algorithms can solve them efficiently in many practical cases.

We apply the technique by keeping set terms in the form of antichains of *simulation-maximal* elements and prune out any other simulation-smaller elements. Intuitively, the notion of a term t being simulation-smaller than t' implies that trees that might be generated from the leaf states $T \cup \{t\}$ can be generated from $T \cup \{t'\}$ too, hence discarding t does not hurt. Formally, we introduce the following rewriting rule:

$$\{t_1, t_2, \dots, t_n\} \rightsquigarrow \{t_2, \dots, t_n\} \quad \text{for } t_1 \sqsubseteq t_2, \quad (31)$$

which may be used to simplify set sub-terms of automata terms. The rule (31) is applied after every iteration of the fixpoint computation on the current partial result. Hence the sequence of partial results is monotone, which, together with the finiteness of $\text{reach}_\Delta(t)$, guarantees termination. The *subsumption* relation \sqsubseteq used in the rule is defined as

$$S \sqsubseteq S' \quad \Leftrightarrow S \subseteq S' \vee S \sqsubseteq^{\forall\exists} S' \quad (32)$$

$$t \& u \sqsubseteq t' \& u' \Leftrightarrow t \sqsubseteq t' \wedge u \sqsubseteq u' \quad (33)$$

$$t + u \sqsubseteq t' + u' \Leftrightarrow t \sqsubseteq t' \wedge u \sqsubseteq u' \quad (34)$$

$$\bar{t} \sqsubseteq \bar{t'} \quad \Leftrightarrow t' \sqsubseteq t \quad (35)$$

$$\pi_X(t) \sqsubseteq \pi_X(t') \Leftrightarrow t \sqsubseteq t' \quad (36)$$

where $S \sqsubseteq^{\forall\exists} S'$ denotes $\forall t \in S \exists t' \in S'. t \sqsubseteq t'$. Intuitively, on base TAs, subsumption corresponds to inclusion of the set terms (the left disjunct of (32)). This clearly has the intended outcome: a larger set of states can always simulate a smaller set in accepting a tree. The rest of the definition is an inductive extension of the base case. It can be shown that \sqsubseteq for any automata term t is an upward simulation on \mathcal{A}_t in the sense of [1]. Consequently, rewriting sub-terms in an automata term according to the new rule (31) does not change its language.

4.4 Product Flattening

Product flattening is a technique that we use to reduce the size of fixpoint saturations that generate conjunctions and disjunctions of sets as their elements. Consider a term of the form $D = \{\pi_X(S_0 \& S'_0)\} - \bar{0}^*$ for a pair of sets of terms S_0 and S'_0 where the TAs \mathcal{A}_{S_0} and $\mathcal{A}_{S'_0}$ have sets of states Q and Q' , respectively. The saturation generates the set $\{\pi_X(S_0 \& S'_0), \dots, \pi_X(S_n \& S'_n)\}$ with $S_i \subseteq Q, S'_i \subseteq Q'$ for all $0 \leq i \leq n$. The size of this set is $2^{|Q|+|Q'|}$ in the worst case. In terms of the automata operations, this fixpoint expansion corresponds to first determinizing both \mathcal{A}_{S_0} and $\mathcal{A}_{S'_0}$ and only then using the product construction (cf. Section 2.4). The automata intersection, however, works for nondeterministic automata too—the determinization is not

needed. Implementing this standard product construction on terms would mean transforming the original fixpoint above into the following fixpoint with a *flattened product*: $D = \{\pi_X(S_0 \& S'_0)\} - \vec{0}^*$ where $\&$ is the augmented product for conjunction. This way, we can decrease the worst-case size of the fixpoint to $|Q| \cdot |Q'|$. A similar reasoning holds for terms of the form $\{\pi_X(S_0 + S'_0)\} - \vec{0}^*$. Formally, the technique can be implemented by the following pair of sub-term rewriting rules where S and S' are non-empty sets of terms:

$$S + S' \rightsquigarrow S [+] S', \quad (37) \quad S \& S' \rightsquigarrow S [\&] S'. \quad (38)$$

Observe that for terms obtained from WS2S formulae using the translation from Section 3, the rules are not helpful in their given form. Consider, for instance, the term $\{\pi_X(\{r\} \& \{q\})\} - \vec{0}^*$ obtained from a formula $\exists X. \varphi \wedge \psi$ with φ and ψ being atoms. The term would be, using rule (38), rewritten into the term $\{\pi_X(\{r \& q\})\} - \vec{0}^*$. Then, during a subsequent fixpoint computation, we might obtain a fixpoint of the following form: $\{\pi_X(\{r \& q\}), \pi_X(\{r \& q, r_1 \& q_1\}), \pi_X(\{r_1 \& q_1, r_2 \& q_2\})\}$, where the occurrences of the projection π_X disallow one to perform the desired union of the inner sets, and so the application of rule (38) did not help. We therefore need to equip our procedure with a rewriting rule that can be used to push the projection inside a set term S :

$$\pi_X(S) \rightsquigarrow \{\pi_X(t) \mid t \in S\}. \quad (39)$$

In the example above, using rule (39) we would now obtain the term $\{\pi_X(r \& q)\} - \vec{0}^*$ (note that we rewrote $\{\{\cdot\}\}$ to $\{\cdot\}$ as mentioned in Section 3) and the fixpoint $\{\pi_X(r \& q), \pi_X(r_1 \& q_1), \pi_X(r_2 \& q_2)\}$. The correctness of the rules is guaranteed by Lemma 4.

We, however, still have to note that there is a danger related with the rules (37)–(39). Namely, if they are applied to some terms in a partially evaluated fixpoint but not to all, the form of these terms might get different (cf. $\pi_X(\{r \& q\})$ and $\pi_X(r \& q)$), and it will not be possible to combine them as source states of TA transitions when computing Δ_a , leading thus to an incorrect result. We resolve the situation in such a way that we apply the rules as a pre-processing step only before we start evaluating the top-level fixpoint, which ensures that all terms will subsequently be generated in a compatible form.

4.5 Nondeterministic Union

Optimization of the product term saturations from the previous section can be pushed one step further for terms of the form $\{\pi_X(S + S')\} - \vec{0}^*$. The idea is to use the *nondeterministic TA union* to implement the union operation instead of the product construction. The TA union is implemented as the component-wise union of the two TAs. Its size is hence linear to the size of the input instead of quadratic as in the case of the product (i.e., $|Q| + |Q'|$ instead of $|Q| \cdot |Q'|$). To work correctly, the nondeterministic union requires disjoint input sets of states (otherwise, the combination of the two transition functions could generate runs that are not possible in either of the input TAs). We implement the nondeterministic union through the following rewriting rule:

$$S + S' \rightsquigarrow S \cup S' \quad \text{for } S \not\bowtie S' \quad (40)$$

where S and S' are sets of terms (similarly to Section 4.4, in order to successfully reduce the fixpoint state space on terms obtained from WS2S formulae, we also need to apply

rule (39) to push projection inside set terms). The relation $\not\bowtie$ used in the rule is the *non-interference* of terms, which generalizes the state space disjointness requirement of the nondeterministic union of TAs. Its complement, the *interference* of terms \bowtie , is defined using the following equivalences:

$$S \bowtie S' \Leftrightarrow S = S' \vee \exists t \in S, t' \in S'. t \bowtie t' \quad (41)$$

$$t \& u \bowtie t' \& u' \Leftrightarrow t \bowtie t' \vee u \bowtie u' \quad (42)$$

$$t + u \bowtie t' + u' \Leftrightarrow t \bowtie t' \vee u \bowtie u' \quad (43)$$

$$\bar{t} \bowtie \bar{t}' \Leftrightarrow t \bowtie t' \quad (44)$$

$$\pi_X(t) \bowtie \pi_X(t') \Leftrightarrow t \bowtie t' \quad (45)$$

$$D \bowtie t \Leftrightarrow D^S \bowtie t \quad (46)$$

$$t \bowtie D \Leftrightarrow t \bowtie D^S \quad (47)$$

$$q \bowtie r \Leftrightarrow \exists 1 \leq k \leq n. q, r \in Q_k \quad (48)$$

For terms t and u that are not matched by any rule above, we define $t \not\bowtie u$ (for instance, $t_1 \& t_2 \not\bowtie u_1 + u_2$). Interference between terms tells us when we cannot perform the rewriting. Intuitively, this happens when we obtain a term $\{S + S'\}$ where S and S' contain states from the same base automaton \mathcal{B}_k with the set of states Q_k .

In order to avoid interference in the terms obtained from WS2S formulae, we can perform the following pre-processing step: When translating a WS2S formula φ into a term t_φ , we create a special version of a base TA for every occurrence of an atomic formula in φ . This way, we can never mix up terms that emerged from different subformulae to enable a transition that would otherwise stay disabled.

To use rule (40), it is necessary to modify treatment of the sink state \emptyset in the definition of Δ of Section 3. The technical difficulty we need to circumvent is that (unlike for finite word automata) the nondeterministic union of two (even complete) TAs is not complete.

This can cause situations such as the following: let $D = \{\pi_X(\{\bar{t}\} + \{\bar{r}\})\} - \bar{0}^*$ such that $\Delta_{\bar{0}}(t, t) = \{t\}$, $\Delta_{\bar{0}}(r, r) = \{r\}$, and $\mathcal{R}(t)$ and $\mathcal{R}(r)$ are both *true*, i.e., both t and r can accept any $\bar{0}$ -tree, which also means that the union of their complements should not accept any $\bar{0}$ -tree. Indeed, the saturation of D is the set term $D^S = \text{reach}_{\Delta_{\bar{0}}}(\{\pi_X(\{\bar{t}\} + \{\bar{r}\})\}) = \{\pi_X(\{\bar{t}\} + \{\bar{r}\})\}$ where it holds that $\neg \mathcal{R}(\pi_X(\{\bar{t}\} + \{\bar{r}\}))$, i.e., it does not accept any $\bar{0}$ -tree. On the other hand, if we use the new rule (40) together with rule (39), we obtain the term $\{\pi_X(\bar{t}), \pi_X(\bar{r})\} - \bar{0}^*$. When computing its saturation, we will obtain a new element $\Delta_{\bar{0}}(\pi_X(\bar{t}), \pi_X(\bar{r})) = \pi_X(\bar{\emptyset})$. The term $\pi_X(\bar{\emptyset})$ was constructed using the implicit rule of Section 3 that sends the otherwise undefined successors of a pair of terms to $\{\emptyset\}$. Note that $\mathcal{R}(\pi_X(\bar{\emptyset}))$ is *true*, yielding that the fixpoint approximation $\{\pi_X(\bar{t}), \pi_X(\bar{r}), \pi_X(\bar{\emptyset})\}$ is a root state, so a $\bar{0}$ -tree is accepted. Therefore, the application of the new rule (40) changed the language.

Although the previous situation cannot happen with terms obtained from WS2S formulae using the translation rules from Section 3, in order to formulate a correctness claim for any terms constructed using our grammar, we remedy the issue by modifying the definition of implicit transitions of Δ to $\{\emptyset\}$ from Section 3. We give the modified transition function $\Delta^\#$ in Fig. 6.

Note that in the previous example, when using the modified transition function $\Delta^\#$ for computing the saturation of the term $\{\pi_X(\bar{t}), \pi_X(\bar{r})\} - \bar{0}^*$, we would from $t \not\bowtie r$

$$\Delta_a^\#(t, t') = \begin{cases} \Delta_a^\bullet(t, t') & \text{if } t \bowtie t' \\ \{\emptyset\} & \text{otherwise} \end{cases} \quad (49)$$

$$\Delta_a^\bullet(t + u, t' + u') = \Delta_a^\#(t, t') [+] \Delta_a^\#(u, u') \quad (50)$$

$$\Delta_a^\bullet(t \& u, t' \& u') = \Delta_a^\#(t, t') [\&] \Delta_a^\#(u, u') \quad (51)$$

$$\Delta_a^\bullet(\pi_X(t), \pi_X(t')) = \left\{ \pi_X(u) \mid u \in \Delta_{\pi_X(a)}^\#(t, t') \right\} \quad (52)$$

$$\Delta_a^\bullet(\bar{t}, \bar{t}') = \left\{ \bar{u} \mid u \in \Delta_a^\#(t, t') \right\} \quad (53)$$

$$\Delta_a^\bullet(S, S') = \left\{ \bigcup_{t \in S, t' \in S'} \Delta_a^\#(t, t') \right\} \quad (54)$$

$$\Delta_a^\bullet(q, r) = \delta_a^B(q, r) \quad (55)$$

Fig. 6: Modified transition function

deduce that $\pi_X(\bar{t}) \not\bowtie \pi_X(\bar{r})$. As a consequence, $\Delta_0^\#(\pi_X(\bar{t}), \pi_X(\bar{r})) = \{\emptyset\}$, which is not accepting.

We will denote the semantics of a term t obtained using $\Delta^\#$ instead of Δ as $\mathcal{L}^\#(t)$. First, we show that the properties of terms from Section 3 under the original semantics hold also for the modified semantics.

Lemma 5 *For automata terms A_1, A_2 and a set term S , the following equalities hold:*

$$\begin{aligned} \mathcal{L}^\#(\{A_1\}) &= \mathcal{L}^\#(A_1) & (a) \quad \mathcal{L}^\#(\{\overline{A_1}\}) &= \overline{\mathcal{L}^\#(A_1)} & (d) \\ \mathcal{L}^\#(\{A_1 + A_2\}) &= \mathcal{L}^\#(A_1) \cup \mathcal{L}^\#(A_2) & (b) \quad \mathcal{L}^\#(\{\pi_X(A_1)\}) &= \pi_X(\mathcal{L}^\#(A_1)) & (e) \\ \mathcal{L}^\#(\{A_1 \& A_2\}) &= \mathcal{L}^\#(A_1) \cap \mathcal{L}^\#(A_2) & (c) \quad \mathcal{L}^\#(S - \vec{0}^*) &= \mathcal{L}^\#(S) - \vec{0}^* & (f) \end{aligned}$$

Proof In the following proofs we abuse notation and denote by \mathcal{A}_S the automaton of the term S with the altered transition function $\Delta^\#$.

(a): We prove the following more general form of (a):

$$\mathcal{L}^\#(\{A_1, \dots, A_n\}) = \mathcal{L}^\# \left(\bigcup_{1 \leq i \leq n} A_i^\epsilon \right) \quad (g)$$

(Again, note that all expanded terms are set terms.) Intuitively, in this proof we show that determinisation does not change the modified language of a term. Let us use $\mathcal{A}_{\bigcup A_i^\epsilon}$ to denote the TA represented by the term $\bigcup_{1 \leq i \leq n} A_i^\epsilon$. Recall that we are using the modified semantics with the altered term transition function $\Delta^\#$.

(\subseteq) Let τ be a tree. It holds that $\tau \in \mathcal{L}^\#(\{A_1, \dots, A_n\})$ if and only if $\tau \in \mathcal{L}^\#(\{A_1^\epsilon, \dots, A_n^\epsilon\})$, i.e., if there is an accepting run ρ on τ in $\mathcal{A}_{\{A_1^\epsilon, \dots, A_n^\epsilon\}}$. Note that ρ maps all leaves of τ to the terms from $\{A_1^\epsilon, \dots, A_n^\epsilon\}$, i.e., each leaf of τ is labelled by some A_i^ϵ , which is a *set* of terms of a lower level (such a set term can be seen as a *macrostate*—i.e., a set of states—from determinisation of TAs). Since ρ is accepting, there is a term $r \in \rho(\epsilon)$ such that $\mathcal{R}(r)$. Note that because $\mathcal{R}(r)$, it follows that $r \neq \emptyset$.

We will now use ρ to construct a run ρ' of $\mathcal{A}_{\bigcup A_i^\varepsilon}$ on τ . The run ρ' will now map every position of τ to a single term. For the root position, we set $\rho'(\epsilon) = r$. We proceed by induction as follows: For all non-leaf positions $w \in \text{dom}(\tau) \setminus \text{leaf}(\tau)$ such that $\rho'(w) = u$, assume that in the original run it holds that $\rho(w.L) = U_L$ and $\rho(w.R) = U_R$. Then, let $u_L \in U_L$ and $u_R \in U_R$ be terms such that $u \in \Delta^\sharp(u_L, u_R)$ (the presence of such terms is guaranteed by (54)). The following inductive invariant holds: If $u \neq \emptyset$, then $u_L \neq \emptyset$ and $u_R \neq \emptyset$ (the invariant follows from (54), the fact that $r \neq \emptyset$, and (41)). We set $\rho'(w.L) = u_L$ and $\rho'(w.R) = u_R$.

As a consequence, we have that $\forall w \in \text{leaf}(\tau) : \rho'(w) \in \bigcup_{1 \leq i \leq n} A_i^\varepsilon$. Then, for each $w \in \text{dom}(\tau)$, it holds that $\rho'(w) \in \text{reach}_{\Delta^\sharp}(\bigcup_{1 \leq i \leq n} A_i^\varepsilon)$ where Δ^\sharp is the (modified) transition function of $\mathcal{A}_{\bigcup A_i^\varepsilon}$. This follows from the definition of modified transition function for set terms (54). Therefore, ρ' is a run of $\mathcal{A}_{\bigcup A_i^\varepsilon}$ on τ and is accepting, so $\tau \in \mathcal{L}^\sharp(\bigcup_{1 \leq i \leq n} A_i^\varepsilon)$.

(\supseteq) Consider a tree $\tau \in \mathcal{L}^\sharp(\bigcup_{1 \leq i \leq n} A_i^\varepsilon)$. Then there is an accepting run ρ on τ in $\mathcal{A}_{\bigcup A_i^\varepsilon}$. We can then use ρ to construct the run ρ' on $\text{dom}(\tau)$ defined as follows: For $u \in \text{leaf}(\tau)$, if $\rho(u) \in A_i^\varepsilon$, we set $\rho'(u) = A_i^\varepsilon$. For $w \in \text{dom}(\tau) \setminus \text{leaf}(\tau)$, we set $\rho'(w) = r$ such that $\{r\} = \Delta_{\tau(w)}^\sharp(\rho'(w.L), \rho'(w.R))$ (we know that $\Delta_{\tau(w)}^\sharp(\rho'(w.L), \rho'(w.R))$ is a singleton set due to (54)). For the constructed run ρ' , it now holds that $\forall w \in \text{dom}(\tau) : \rho(w) \in \rho'(w)$, therefore ρ' is an accepting run on τ in $\mathcal{A}_{\{A_1^\varepsilon, \dots, A_n^\varepsilon\}}$, i.e., $\tau \in \mathcal{L}^\sharp(\{A_1, \dots, A_n\})$.

(b)–(e): The proof is identical to the proof of corresponding variant in Lemma 3 (with altered term transition function).

(f): The proof is similar to the proof of Lemma 3f with one exception. In particular, in the proof of (the modified version of) Claim 1, we need to make use of the fact that interference is preserved along transition relation, which is formalized in the following claim.

Claim 5 For two terms t_1, t_2 such that $t_1 \bowtie t_2$, symbol a , and for each $t \in \Delta_a^\sharp(t_1, t_2)$ it holds that $t \bowtie t_1$ and $t \bowtie t_2$.

Proof By induction on the structure of terms:

- *Base case:* Let t_1 and t_2 be states of some base automata. From $t_1 \bowtie t_2$ and (48), we can deduce that t_1 and t_2 are both states of some base automaton \mathcal{B}_k , i.e., $t_1, t_2 \in Q_k$. Then it also holds that $\Delta_a^\sharp(t_1, t_2) \subseteq Q_k$, so for every $t \in \Delta_a^\sharp(t_1, t_2)$, we have that $t \bowtie t_1$ and $t \bowtie t_2$.

Let us now continue with inductive cases.

- Let $t_1 = u_1 \& v_1$ and $t_2 = u_2 \& v_2$. From (42), it follows that either $u_1 \bowtie u_2$ or $v_1 \bowtie v_2$.

$$\begin{aligned}
 \Delta_a^\sharp(t_1, t_2) &= \Delta_a^\sharp(u_1 \& v_1, u_2 \& v_2) \\
 &= \Delta_a^\bullet(u_1 \& v_1, u_2 \& v_2) && \text{[(49)]} \\
 &= \Delta_a^\sharp(u_1, u_2) [\&] \Delta_a^\sharp(v_1, v_2) && \text{[(51)]} \\
 &= \{u \& v \mid u \in \Delta_a^\sharp(u_1, u_2) \wedge v \in \Delta_a^\sharp(v_1, v_2)\} && \text{[def. of [\&]]}
 \end{aligned}$$

Therefore, for all $t = u \& v \in \Delta_a^\sharp(t_1, t_2)$:

- if $u_1 \bowtie u_2$, then $u \bowtie u_1$ and $u \bowtie u_2$, so, from (42), it also holds that $t \bowtie t_1$ and $t \bowtie t_2$; and
- if $v_1 \bowtie v_2$, then $v \bowtie v_1$ and $v \bowtie v_2$, so, from (42), it also holds that $t \bowtie t_1$ and $t \bowtie t_2$.
- The proofs for other inductive cases are similar. ■

The other parts of the proof are similar. □

Lemma 6 *For sets of terms S and S' such that $S \neq \emptyset$ and $S' \neq \emptyset$, we have:*

$$\mathcal{L}^\#(\{S + S'\}) = \mathcal{L}^\#(\{S[+]S'\}), \quad (a)$$

$$\mathcal{L}^\#(\{S \& S'\}) = \mathcal{L}^\#(\{S[\&]S'\}), \quad (b)$$

$$\mathcal{L}^\#(\{\pi_X(S)\}) = \mathcal{L}^\#(\{\pi_X(t) \mid t \in S\}). \quad (c)$$

Proof (a): (\subseteq) Let $\tau \in \mathcal{L}^\#(\{S + S'\})$. From Lemma 5b we have $\mathcal{L}^\#(\{S + S'\}) = \mathcal{L}^\#(S) \cup \mathcal{L}^\#(S')$. Hence there are runs ρ_1 in \mathcal{A}_{S^ϵ} and ρ_2 in $\mathcal{A}_{S'^\epsilon}$ on τ such that for all $w \in \text{dom}(\tau)$, $\rho_1(w) \neq \emptyset \wedge \rho_2(w) \neq \emptyset$, and, moreover, at least one of them is accepting. Note that both runs exist since the transition function $\Delta^\#$ is complete (for a pair of terms t_1 and t_2 , (i) if $t_1 \not\bowtie t_2$, then trivially $\Delta^\#(t_1, t_2) = \{\emptyset\} \neq \emptyset$ and (ii) if $t_1 \bowtie t_2$, then, from the definition of modified transition function we have $\Delta^\#(t_1, t_2) = \Delta^\bullet(t_1, t_2) \neq \emptyset$). Then, we can construct a mapping ρ from τ defined such that for all $w \in \text{dom}(\tau)$, we set $\rho(w) = \rho_1(w) + \rho_2(w)$. Note that ρ is a run of $\mathcal{A}_{\{t_1^\epsilon + t_2^\epsilon \mid t_1 \in S, t_2 \in S'\}}$ on τ , i.e., it maps leaves of $\text{dom}(\tau)$ to terms of the form $t_1^\epsilon + t_2^\epsilon$ for $t_1 \in S$ and $t_2 \in S'$. Moreover, ρ is accepting since at least one of the runs ρ_1 and ρ_2 is accepting. Therefore, $\tau \in \mathcal{L}^\#(\{t_1 + t_2 \mid t_1 \in S, t_2 \in S'\})$. From the definition of the augmented product, it follows that $\tau \in \mathcal{L}^\#(S[+]S')$ and, finally, from Lemma 5a, we have $\tau \in \mathcal{L}^\#(\{S[+]S'\})$.

(\supseteq) Let $\tau \in \mathcal{L}^\#(\{S[+]S'\})$. From Lemma 5a, we get $\tau \in \mathcal{L}^\#(S[+]S')$, and from the definition of the augmented product, we obtain $\tau \in \mathcal{L}^\#(\{t_1 + t_2 \mid t_1 \in S, t_2 \in S'\})$. Therefore, there is an accepting run ρ on τ in $\mathcal{A}_{\{t_1^\epsilon + t_2^\epsilon \mid t_1 \in S, t_2 \in S'\}}$. Furthermore, let us consider the run ρ' of $\mathcal{A}_{\{S + S'\}}$ on τ (note that, due to (12), the definition of interference, and the completeness of the transition function, there is exactly one). By induction on the structure of τ , we can easily show that for all $w \in \text{dom}(\tau)$, if $\rho(w) = t_1 + t_2$, then $\rho'(w) = S_1 + S_2$ such that $t_1 \in S_1$ and $t_2 \in S_2$ (the property clearly holds at leaves and is also preserved by the transition function). Let $\rho(\epsilon) = t_1^\epsilon + t_2^\epsilon$ and $\rho'(\epsilon) = S_1^\epsilon + S_2^\epsilon$. Since $\mathcal{R}(t_1^\epsilon + t_2^\epsilon)$, it also holds that $\mathcal{R}(S_1^\epsilon + S_2^\epsilon)$. Therefore, ρ' is accepting, so $\tau \in \mathcal{L}^\#(\{S + S'\})$.

(b): Dual to (a).

(c): Identical to the proof of Lemma 4c (with the altered transition function). □

The following theorem shows that formula-to-term translation is correct even for the modified semantics.

Theorem 2 *Let φ be a WS2S formula. Then, $\mathcal{L}^\#(t_\varphi) = \mathcal{L}(\varphi)$.*

Proof In the proof we use the notion of expanded terms. By $t^{\epsilon, \Delta}$ we denote that a term t is expanded using term transition function Δ from Section 3.2. In the first step we prove $\mathcal{L}^\#(t_\psi) = \mathcal{L}(t_\psi)$ by showing that $\langle \psi \rangle^{\epsilon, \Delta} = \langle \psi \rangle^{\epsilon, \Delta^\#}$ for each subformula ψ of φ by induction on the structure of φ :

- $\varphi = \varphi_0$ where φ_0 is an atomic formula: Let I_{φ_0} be the set of leaf states and Q_{φ_0} set of states of a unique \mathcal{A}_{φ_0} . For each $q_1, q_2 \in Q_{\varphi_0}$ we have $q_1 \bowtie q_2$. Since I_{φ_0} is already expanded, $\langle \varphi_0 \rangle^{e, \Delta} = \langle \varphi_0 \rangle^{e, \Delta^\#}$.
- $\varphi = \psi_1 \wedge \psi_2$: We use the following equational reasoning.

$$\begin{aligned}
\langle \varphi \rangle^{e, \Delta} &= \langle \psi_1 \wedge \psi_2 \rangle^{e, \Delta} = (\langle \psi_1 \rangle \& \langle \psi_2 \rangle)^{e, \Delta} && \text{[(25)]} \\
&= \langle \psi_1 \rangle^{e, \Delta} \& \langle \psi_2 \rangle^{e, \Delta} && \text{[expansion propagation]} \\
&= \langle \psi_1 \rangle^{e, \Delta^\#} \& \langle \psi_2 \rangle^{e, \Delta^\#} && \text{[induction hypothesis]} \\
&= (\langle \psi_1 \rangle \& \langle \psi_2 \rangle)^{e, \Delta^\#} && \text{[expansion propagation]} \\
&= \langle \varphi \rangle^{e, \Delta^\#} && \text{[(25)]}
\end{aligned}$$

- $\varphi = \psi_1 \vee \psi_2$: Dual to $\psi_1 \wedge \psi_2$.
- $\varphi = \neg \psi$: We use the following equational reasoning.

$$\begin{aligned}
\langle \varphi \rangle^{e, \Delta} &= (\langle \neg \psi \rangle)^{e, \Delta} && \text{[(27)]} \\
&= \overline{\langle \psi \rangle}^{e, \Delta} && \text{[expansion propagation]} \\
&= \overline{\langle \psi \rangle}^{e, \Delta^\#} && \text{[induction hypothesis]} \\
&= (\langle \neg \psi \rangle)^{e, \Delta^\#} && \text{[expansion propagation]} \\
&= \langle \varphi \rangle^{e, \Delta^\#} && \text{[(27)]}
\end{aligned}$$

- $\varphi = \exists X. \psi$: We use the following equational reasoning.

$$\begin{aligned}
\langle \exists X. \psi \rangle^{e, \Delta} &= (\{\pi_X(\langle \psi \rangle)\} - \vec{0}^*)^{e, \Delta} && \text{[(28)]} \\
&= (reach_{\Delta_{\vec{0}}}(\{\pi_X(\langle \psi \rangle)\}))^{e, \Delta} && \text{[(14)]} \\
&= reach_{\Delta_{\vec{0}}}(\{\pi_X(\langle \psi \rangle^{e, \Delta})\}) && \text{[expansion propagation]} \\
&= reach_{\Delta_{\vec{0}}}(\{\pi_X(\langle \psi \rangle^{e, \Delta^\#})\}) && \text{[induction hypothesis]}
\end{aligned}$$

From the inductive construction of $\langle \varphi \rangle$ let us now observe that for every $t_1, t_2 \in reach_{\Delta_{\vec{0}}}(\{\pi_X(\langle \psi \rangle^{e, \Delta^\#})\})$ we have $t_1 \bowtie t_2$. This follows from the definition of interference and from the fact that for every set term S occurring in $\langle \psi \rangle$ and every $t_1, t_2 \in S$ it holds that $t_1 \bowtie t_2$. Based on the previous, we have

$$\begin{aligned}
\langle \exists X. \psi \rangle^{e, \Delta} &= reach_{\Delta_{\vec{0}}}(\{\pi_X(\langle \psi \rangle^{e, \Delta^\#})\}) \\
&= reach_{\Delta_{\vec{0}}^\#}(\{\pi_X(\langle \psi \rangle^{e, \Delta^\#})\}) && \text{[previous reasoning]} \\
&= \langle \exists X. \psi \rangle^{e, \Delta^\#} && \text{[expansion prop. and (28)]}
\end{aligned}$$

Since $t_\varphi^{e, \Delta} = t_\varphi^{e, \Delta^\#}$ and the fact that for each $a \in \Sigma$ and $t_1, t_2 \in t_\varphi^{e, \Delta}$: $\Delta_a(t_1, t_2) = \Delta_a^\#(t_1, t_2)$, we have $\mathcal{L}(t_\varphi) = \mathcal{L}^\#(t_\varphi)$. Finally, from Theorem 1 we have $\mathcal{L}(t_\varphi) = \mathcal{L}(\varphi)$, which concludes the proof. \square

Based on Lemma 5, Lemma 6, and Theorem 2 we can show correctness of the non-deterministic union rule (40):

Lemma 7 *Let S and S' be sets of terms such that $S \not\bowtie S'$. Then*

$$\mathcal{L}^\#(\{S + S'\}) = \mathcal{L}^\#(S \cup S').$$

Proof (\subseteq) From Lemma 5b, we have $\mathcal{L}^\#(\{S + S'\}) = \mathcal{L}^\#(S) \cup \mathcal{L}^\#(S')$. Let $\tau \in \mathcal{L}^\#(S) \cup \mathcal{L}^\#(S')$ and ρ be an accepting run on τ of either \mathcal{A}_{S^e} or $\mathcal{A}_{S'^e}$. Therefore, ρ is an accepting run on τ also in $\mathcal{A}_{S^e \cup S'^e}$.

(\supseteq) Let $\tau \in \mathcal{L}^\#(S \cup S')$. For each $t_1 \in S^e$ and $t_2 \in S'^e$ it holds that $t_1 \not\bowtie t_2$, so we have that if $t \in \Delta_a^\#(t_1, t_2)$, then $t = \emptyset$. Therefore, if ρ is an accepting run of $\mathcal{A}_{S^e \cup S'^e}$ on τ , then ρ is an accepting run on τ in either \mathcal{A}_{S^e} or $\mathcal{A}_{S'^e}$. Without loss of generality, suppose that ρ is an accepting run on τ of \mathcal{A}_{S^e} and let ρ' be the run of $\mathcal{A}_{\{S + S'\}}$ on τ (note that $\mathcal{A}_{\{S + S'\}}$ is deterministic and complete, so ρ' is unique). By induction on the structure of τ , we can easily show that for all $w \in \text{dom}(\tau)$, if $\rho(w) = t_1$, then $\rho'(w) = S_1 + S_2$ such that $t_1 \in S_1$ (the property clearly holds at leaves and is also preserved by the modified transition function). Let $\rho(\epsilon) = t_1^\epsilon$ and $\rho'(\epsilon) = S_1^\epsilon + S_2^\epsilon$. Since $\mathcal{R}(t_1^\epsilon)$, it also holds that $\mathcal{R}(S_1^\epsilon + S_2^\epsilon)$. Therefore, ρ' is accepting, so $\tau \in \mathcal{L}^\#(\{S + S'\})$. \square

Note that although the optimization presented in this section can improve the worst-case number of reached terms, its use comes with a cost. In order to guarantee that rule (40) can be performed, we need to use a different base automaton for each atomic formula. A different base automaton can be obtained, e.g., by instantiating the automaton for a given formula every time with different names of states. The use of different base automata makes it, however, less likely that memoization avoids evaluating some function call (even though a similar one might have already been evaluated), which may significantly impact the overall performance of the decision procedure.

5 Experimental Evaluation

We have implemented the above introduced techniques (with the exception of Section 4.5 for the reasons described therein) in a prototype Haskell tool.² The base automata, hard-coded into the tool, were the TAs for the basic predicates from Section 2, together with automata for predicates $\text{Sing}(X)$ and $X = \{p\}$ for a variable X and a fixed tree position p . As an additional optimisation, our tool uses the so-called *antiprenexing* (proposed already in [13]), which pushes quantifiers down the formula tree using the standard logical equivalences. Intuitively, antiprenexing reduces the complexity of elements within fixpoints by removing irrelevant parts outside the fixpoint.

We have performed experiments with our tool on various formulae and compared its performance with that of MONA. We applied MONA both on the original form of the considered formulae as well as on their versions obtained by antiprenexing (which is built into our tool and which—as we realised—can significantly help MONA too). Our preliminary implementation of product flattening (cf. Section 4.4) is restricted to parts below the lowest fixpoint, and our experiments showed that it does not work well when applied on this level, where the complexity is not too high, so we turned it off for the experiments. We ran all experiments on a 64-bit Linux Debian workstation

² The implementation is available at <https://github.com/vhavlena/lazy-wsks>.

Table 1: Experimental results over the following parametric families of formulae:

1. $\varphi_n^{pt} \equiv \forall Z_1, Z_2. \exists X_1, \dots, X_n. \text{edge}(Z_1, X_1) \wedge \bigwedge_{i=1}^n \text{edge}(X_i, X_{i+1}) \wedge \text{edge}(X_n, Z_2)$ where $\text{edge}(X, Y) \equiv \text{edge}_L(X, Y) \vee \text{edge}_R(X, Y)$ and $\text{edge}_{L/R}(X, Y) \equiv \exists Z. Z = S_{L/R}(X) \wedge Z \subseteq Y$
2. $\varphi_n^{cnst} \equiv \exists X. X = \{(\text{LR})^4\} \wedge X = \{(\text{LR})^n\}$
3. $\varphi_n^{sub} = \forall X_1, \dots, X_n \exists X. \bigwedge_{i=1}^{n-1} X_i \subseteq X \Rightarrow (X_{i+1} = S_L(X) \vee X_{i+1} = S_R(X))$

φ	n	running time (sec)			# of subterms/states		
		<i>Lazy</i>	MONA	MONA+AP	<i>Lazy</i>	MONA	MONA+AP
φ_n^{pt}	1	0.02	0.16	0.15	149	216	216
	2	0.50	—	—	937	—	—
	3	0.83	—	—	2,487	—	—
	4	34.95	—	—	8,391	—	—
	5	60.94	—	—	23,827	—	—
φ_n^{cnst}	80	14.60	40.07	40.05	1,146	27,913	27,913
	90	21.03	64.26	64.20	1,286	32,308	32,308
	100	28.57	98.42	98.91	1,426	36,258	36,258
	110	38.10	—	—	1,566	—	—
	120	49.82	—	—	1,706	—	—
φ_n^{sub}	3	0.01	0.00	0.00	140	92	92
	4	0.04	34.39	34.47	386	170	170
	5	0.24	—	—	981	—	—
	6	2.01	—	—	2,376	—	—

with the Intel(R) Core(TM) i7-2600 CPU running at 3.40 GHz with 16 GiB of RAM. The timeout was set to 100 s.

We first considered various WS2S formulae on which MONA was successfully applied previously in the literature. On them, our tool is quite slower than MONA, which is not much surprising given the amount of optimisations built into MONA (for instance, for the benchmarks from [25], MONA on average took 0.1 s, while we timeouted). Next, we identified several parametric families of formulae (adapted from [13]), such as, e.g., $\varphi_n^{horn} \equiv \exists X. \forall X_1. \exists X_2, \dots, X_n. ((X_1 \subseteq X \wedge X_1 \neq X_2) \Rightarrow X_2 \subseteq X) \wedge \dots \wedge ((X_{n-1} \subseteq X \wedge X_{n-1} \neq X_n) \Rightarrow X_n \subseteq X)$, where our approach finished within 10 ms, while the time of MONA was increasing when increasing the parameter n , going up to 32 s for $n = 14$ and timeouting for $k \geq 15$. It turned out that MONA could, however, easily handle these formulae after antiprenexing, again (slightly) outperforming our tool. Finally, we also identified several parametric families of formulae that MONA could handle only very badly or not at all, even with antiprenexing, while our tool can handle them much better. These formulae are mentioned in the caption of Table 1, which give detailed results of the experiments.

Particularly, the columns under “**running time (sec)**” give the running times (in seconds) of our tool (denoted *Lazy*), MONA, and MONA with antiprenexing (MONA+AP). The columns under “**# of subterms/states**” characterize the size of the generated terms and automata. Namely, for our approach, we give the number of nodes in the final term tree (with the leaves being states of the base TAs). For MONA, we give the sum of the numbers of states of all the minimal deterministic TAs constructed by MONA when evaluating the formula. The “—” sign means a timeout or that the tool ran out of memory.

The formulae considered in Table 1 speak about various paths in trees. We were originally inspired by formulae kindly provided by Josh Berdine, which arose from attempts to translate separation logic formulae to WS2S (and use MONA to discharge them), which are beyond the capabilities of MONA (even with antiprenexing). We were also unable to handle them with our tool, but our experimental results on the tree path formulae indicate (despite the prototypical implementation) that our techniques can help one to handle some complex graph formulae that are out of the capabilities of MONA. Thus, they provide a new line of attack on deciding hard WS2S formulae, complementary to the heuristics used in MONA. Improving the techniques and combining them with the classical approach of MONA is a challenging subject for our future work.

6 Related Work

The seminal works [7, 30] on the automata-logic connection were the milestones leading to what we call here the classical tree automata-based decision procedure for WS k S [35]. Its non-elementary worst-case complexity was proved in [33], and the work [16] presents the first implementation, restricted to WS1S, with the ambition to use heuristics to counter the high complexity. The authors of [9] provide an excellent survey of the classical results and literature related to WS k S and tree automata.

The tool MONA [12] implements the classical decision procedures for both WS1S and WS2S. It is still the standard tool of choice for deciding WS1S/WS k S formulae due to its all-around most robust performance. The efficiency of MONA stems from many optimizations, both higher-level (such as automata minimization, the encoding of first-order variables used in models, or the use of multi-terminal BDDs to encode the transition function of the automaton) as well as lower-level (e.g. optimizations of hash tables, etc.) [23, 21]. The MSO(Str) logic, a dialect of WS1S, can also be decided by a similar automata-based decision procedure, implemented within, e.g., JMOSEL [36] or the symbolic finite automata framework of [10]. In particular, JMOSEL implements several optimizations (such as second-order value numbering [27]) that allow it to outperform MONA on some benchmarks (MONA also provides an MSO(Str) interface on top of the WS1S decision procedure).

The original inspiration for our work are the antichain techniques for checking universality and inclusion of finite automata [11, 39, 5, 1] and language emptiness of alternating automata [11], which use symbolic computation together with subsumption to prune large state spaces arising from subset construction. This paper is a continuation of our work on WS1S, which started by [14], where we discussed a basic idea of generalizing the antichain techniques to a WS1S decision procedure. In [13], we then presented a complete WS1S decision procedure based on these ideas that is capable to rival MONA on already interesting benchmarks. The work in [37] presents a decision procedure that, although phrased differently, is in essence fairly similar to that of [13]. One additional feature of [37] over [13] is that it can employ laziness even when generating base automata. This feature can have a significant effect for formulae with large integer constants, such as $x = 1,000,000,000 \wedge x = 1,000$. While the formula is clearly unsatisfiable, MONA constructs the base automata, which might already be too large to fit in the memory.

This paper generalizes [13] to WS2S. It is not merely a straightforward generalization of the word concepts to trees. A nontrivial transition was needed from language terms of [13], with their semantics being defined straightforwardly from the seman-

tics of sub-terms, to tree automata terms, with the semantics defined as a language of an automaton with transitions defined inductively to the structure of the term. This change makes the reasoning and correctness proof considerably more complex, though the algorithm itself stays technically quite simple. Due to our implementation in Haskell, we can, similarly to [37], avoid constructing large base automata and only construct those parts necessary to establishing the status of input formulae.

Finally, Ganzow and Kaiser [15] developed a new decision procedure for the weak monadic second-order logic on inductive structures within their tool Toss. Their approach completely avoids automata; instead, it is based on the Shelah’s composition method. The paper reports that the Toss tool could outperform MONA on two families of WS1S formulae, one derived from Presburger arithmetics and one formula of the form that we mention in our experiments as problematic for MONA but solvable easily by MONA with antiprenexing.

Acknowledgements We thank the anonymous reviewers, both of the conference and the journal version of the paper, for their careful reading of the drafts, the spotted bugs, and the helpful comments on how to improve the exposition in this paper. This work was supported by the Czech Science Foundation project 19-24397S, the FIT BUT internal project FIT-S-20-6427, and The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science—LQ1602.

References

1. Abdulla, P.A., Chen, Y.F., Holík, L., Mayr, R., Vojnar, T.: When Simulation Meets Antichains (on Checking Language Inclusion of NFAs). In: TACAS’10, *LNCS*, vol. 6015, pp. 158–174. Springer (2010)
2. Basin, D., Klarlund, N.: Automata Based Symbolic Reasoning in Hardware Verification. In: CAV’98, *LNCS*, pp. 349–361. Springer (1998)
3. Baukus, K., Bensalem, S., Lakhnech, Y., Stahl, K.: Abstracting WS1S Systems to Verify Parameterized Networks. In: TACAS’00, *LNCS*, vol. 1785, pp. 188–203. Springer (2000)
4. Bodeveix, J., Filali, M.: FMona: A Tool for Expressing Validation Techniques over Infinite State Systems. In: TACAS’00, *LNCS*, vol. 1785, pp. 204–219. Springer (2000)
5. Bouajjani, A., Habermehl, P., Holík, L., Touili, T., Vojnar, T.: Antichain-Based Universality and Inclusion Testing over Nondeterministic Finite Tree Automata. In: CIAA’08, *LNCS*, vol. 5148, pp. 57–67. Springer (2008)
6. Bozga, M., Iosif, R., Sifakis, J.: Structural Invariants for Parametric Verification of Systems with Almost Linear Architectures. Tech. Rep. arXiv:1902.02696 (2019)
7. Büchi, J.R.: On a Decision Method in Restricted Second-Order Arithmetic. In: International Congress on Logic, Methodology, and Philosophy of Science, pp. 1–11. Stanford University Press (1962)
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. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2008)
10. D’Antoni, L., Veanes, M.: Minimization of Symbolic Automata. In: POPL’14., pp. 541–554 (2014)
11. Doyen, L., Raskin, J.F.: Antichain Algorithms for Finite Automata. In: TACAS’10, *LNCS*, vol. 6015, pp. 2–22. Springer (2010)
12. Elgaard, J., Klarlund, N., Möller, A.: MONA 1.x: New Techniques for WS1S and WS2S. In: CAV’98, *LNCS*, vol. 1427, pp. 516–520. BRICS, Department of Computer Science, Aarhus University, Springer (1998)
13. Fiedor, T., Holík, L., Janků, P., Lengál, O., Vojnar, T.: Lazy Automata Techniques for WS1S. In: TACAS’17, *LNCS*, vol. 10205, pp. 407–425. Springer (2017)
14. Fiedor, T., Holík, L., Lengál, O., Vojnar, T.: Nested Antichains for WS1S. In: TACAS’15, *LNCS*, vol. 9035. Springer (2015)

15. Ganzow, T., Kaiser, L.: New Algorithm for Weak Monadic Second-Order Logic on Inductive Structures. In: CSL'10, *LNCS*, vol. 6247, pp. 366–380. Springer (2010)
16. Glenn, J., Gasarch, W.: Implementing WS1S via Finite Automata. In: Workshop on Implementing Automata, *LNCS*, vol. 1260, pp. 50–63. Springer (1996)
17. Habermehl, P., Holík, L., Rogalewicz, A., Šimáček, J., Vojnar, T.: Forest Automata for Verification of Heap Manipulation. *Formal Methods in System Design* **41**(1), 83–106 (2012)
18. Hamza, J., Jobstmann, B., Kuncak, V.: Synthesis for Regular Specifications over Unbounded Domains. In: FMCAD'10, pp. 101–109. IEEE Computer Science (2010)
19. Havlena, V., Holík, L., Lengál, O., Vojnar, T.: Automata terms in a lazy WS1S decision procedure. In: Proc. of CADE-27, *LNCS*, vol. 11716, pp. 300–318. Springer (2019)
20. Hune, T., Sandholm, A.: A Case Study on Using Automata in Control Synthesis. In: FASE'00, *LNCS*, vol. 1783, pp. 349–362. Springer (2000)
21. Klarlund, N.: A Theory of Restrictions for Logics and Automata. In: CAV'99, *LNCS*, vol. 1633, pp. 406–417. Springer (1999)
22. Klarlund, N., Møller, A.: MONA Version 1.4 User Manual. BRICS, Department of Computer Science, Aarhus University (2001). Notes Series NS-01-1. Available from <http://www.brics.dk/mona/>. Revision of BRICS NS-98-3
23. Klarlund, N., Møller, A., Schwartzbach, M.I.: MONA Implementation Secrets. *International Journal of Foundations of Computer Science* **13**(4), 571–586 (2002)
24. Klarlund, N., Nielsen, M., Sunesen, K.: A Case Study in Automated Verification Based on Trace Abstractions. In: Formal System Specification, The RPC-Memory Specification Case Study, *LNCS*, vol. 1169. Springer (1996)
25. Madhusudan, P., Parlato, G., Qiu, X.: Decidable Logics Combining Heap Structures and Data. In: POPL'11, pp. 611–622. ACM (2011)
26. Madhusudan, P., Qiu, X.: Efficient Decision Procedures for Heaps Using STRAND. In: SAS'11, *LNCS*, vol. 6887, pp. 43–59. Springer (2011)
27. Margaria, T., Steffen, B., Topnik, C.: Second-Order Value Numbering. In: GraMoT'10, *ECEASST*, vol. 30, pp. 1–15. EASST (2010)
28. Møller, A., Schwartzbach, M.: The Pointer Assertion Logic Engine. In: PLDI'01. ACM Press (2001). Also in SIGPLAN Notices 36(5), 2001.
29. Morawietz, F., Cornell, T.: The Logic-Automaton Connection in Linguistics. In: LACL'97, *LNAI*, vol. 1582. Springer (1997)
30. Rabin, M.O.: Decidability of Second Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society* **141**, 1–35 (1969)
31. Sandholm, A., Schwartzbach, M.I.: Distributed Safety Controllers for Web Services. In: FASE'98, pp. 270–284. Springer (1998)
32. Smith, M.A., Klarlund, N.: Verification of a Sliding Window Protocol Using IOA and MONA. In: FORTE/PSTV'00, *IFIP*, vol. 183, pp. 19–34. Kluwer (2000)
33. Stockmeyer, L.J., Meyer, A.R.: Word Problems Requiring Exponential Time (Preliminary Report). In: Fifth Annual ACM Symposium on Theory of Computing, STOC'73, pp. 1–9. ACM, New York, NY, USA (1973)
34. Tateishi, T., Pistoia, M., Tripp, O.: Path- and Index-Sensitive String Analysis Based on Monadic Second-Order Logic. *ACM Trans. Comput. Log.* **22**(4), 33:1–33:33 (2013)
35. Thatcher, J.W., Wright, J.B.: Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic. *Mathematical systems theory* **2**(1), 57–81 (1968)
36. Topnik, C., Wilhelm, E., Margaria, T., Steffen, B.: jMosel: A Stand-Alone Tool and jABC Plugin for M2L(Str). In: SPIN'06, *LNCS*, vol. 3925, pp. 293–298. Springer (2006)
37. Traytel, D.: A Coalgebraic Decision Procedure for WS1S. In: 24th EACSL Annual Conference on Computer Science Logic (CSL'15), *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 41, pp. 487–503. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2015)
38. Wies, T., Muñoz, M., Kuncak, V.: An Efficient Decision Procedure for Imperative Tree Data Structures. In: CADE'11, *LNCS*, vol. 6803, pp. 476–491. Springer (2011)
39. Wulf, M.D., Doyen, L., Henzinger, T.A., Raskin, J.F.: Antichains: A New Algorithm for Checking Universality of Finite Automata. In: CAV'06, *LNCS*, vol. 4144, pp. 17–30. Springer (2006)
40. Wulf, M.D., Doyen, L., Maquet, N., Raskin, J.F.: Antichains: Alternative Algorithms for LTL Satisfiability and Model-Checking. In: TACAS'08, *LNCS*, vol. 4693. Springer (2008)

-
41. Wulf, M.D., Doyen, L., Raskin, J.F.: A Lattice Theory for Solving Games of Imperfect Information. In: HSCC'06, *LNCS*, vol. 3927. Springer (2006)
 42. Zee, K., Kuncak, V., Rinard, M.C.: Full Functional Verification of Linked Data Structures. In: POPL'08, pp. 349–361. ACM (2008)
 43. Zhou, M., He, F., Wang, B., Gu, M., Sun, J.: Array Theory of Bounded Elements and its Applications. *J. Autom. Reasoning* **52**(4), 379–405 (2014)