EPiC Series in Computer Science
Volume 35, 2015, Pages 59–66
LPAR-20. 20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning - Short Presentations

Symbolic WS1S

Loris D'Antoni[1] and Margus Veanes[2]

[1] University of Wisconsin-Madison
loris@cs.wisc.edu

[2] Microsoft Research
margus@microsoft.com

Abstract

We extend weak monadic second-order logic of one successor (WS1S) to symbolic alphabets by allowing character predicates to range over decidable first order theories and not just finite alphabets. We call this extension symbolic WS1S (s-WS1S). We then propose two decision procedures for such a logic: 1) we use symbolic automata to extend the classic reduction from WS1S to finite automata to our symbolic logic setting; 2) we show that every s-WS1S formula can be reduced to a WS1S formula that preserves satisfiability, at the price of an exponential blow-up.

1 Introduction

Logics that can reason about sequences of events or strings are ubiquitous in many fields of computer science including program verification, string processing, and program monitoring [5, 6]. These logics are typically described as temporal logics, as they can describe events appearing in defined order. Examples of such logics include linear temporal logic (LTL) [3] and weak monadic second-order logic of one successor (WS1S) [1], often just called MSO. Temporal logics are equipped with operators that can describe the order between events appearing in a given sequence. For example, an LTL formula can specify that in a particular string the letter a should always be followed by a b. Despite the increase in expressiveness provided by these operators both these logics have desirable decidable properties, the most important being decidable satisfiability.

Although widely used in many practical contexts, classic temporal logics can only describe sequences of events that are drawn from a finite domain. In particular, the predicates used to test individual characters always have to be of finite cardinality and disjoint. For example, the following property of a list of integers is not naturally describable in existing decidable temporal logics without first performing some abstractions on the alphabet of the list being read: $P =$*every odd number is eventually followed by a number greater than 4*. In this paper we propose symbolic WS1S (s-WS1S), an extension of WS1S that can describe such a property while retaining decidable satisfiability checking. s-WS1S formulas are parametric in an underlying alphabet theory (in the case of P linear integer arithmetic). The underlying theory tells us what is the alphabet of our sequences (in the case of P the integers) and what base predicates can appear in our formulas (in the case of P any unary formula in linear integer arithmetic).

A.Fehnker, A.McIver, G.Sutcliffe and A.Voronkov (eds.), LPAR-20 (EPiC Series in Computer Science, vol. 35), pp. 59–66

In order to retain decidability the underlying theory is required to form a decidable Boolean algebra, i.e., checking satisfiability is decidable and predicates are closed under Boolean operations. Despite this requirement s-WS1S is strictly more expressive than WS1S. We propose two decision procedures for s-WS1S.

Our first decision procedure is based on the following automata-based reduction for WS1S: every WS1S formula φ can be compiled into a finite automaton that accepts the same set of sequences accepted by φ [6]. The formula φ is then satisfiable if the equivalent automaton is not empty. Similarly, we show that every s-WS1S formula can be compiled into an equivalent Symbolic Finite Automaton (s-FA), an extension of finite automata that can capture strings over arbitrary and potentially infinite alphabets [2]. Since checking emptiness of s-FAs is decidable, the reduction also provides a decision procedure for s-WS1S.

Our second decision procedures is based on a reduction to the non-symbolic version of WS1S. This reduction is based on the following idea: although the set of predicates and alphabet symbols in a given theory can be infinite, the set of concrete predicates appearing in a concrete formula is finite. Moreover, the set of Boolean combinations of such predicates is also finite and each such combination describes an equivalence class for the set of symbols appearing in our alphabet. Concretely, every s-WS1S φ formula can be compiled into a WS1S formula that preserves satisfiability in which the alphabet symbols are the set of possible Boolean combinations of all the alphabet predicates appearing in φ. Since there can be exponentially many combinations, this reduction can cause an exponential blow-up.

Contributions. In summary our contributions are:

1. The logic s-WS1S, which extends WS1S to describe sequences over arbitrary and potentially infinite alphabets.
2. A symbolic decision procedure for s-WS1S based on a reduction to symbolic finite automata.
3. A decision procedure for s-WS1S based on an exponential reduction to the non-symbolic version of WS1S.

2 Symbolic Monadic Second Order Logic

We first formally define the notion of effective Boolean algebra. Next, we define symbolic weak monadic second-order logic of one successor (s-WS1S) and its semantics. Last, we define symbolic finite automata (s-FA), which will be used in our algorithms.

2.1 Effective Boolean Algebras

An *effective Boolean algebra* $\mathcal{A}$ has components $(\mathfrak{D}, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$. $\mathfrak{D}$ is a recursively enumerable set of *domain elements*. Ψ is an recursively enumerable set of *predicates* closed under the Boolean connectives and $\bot, \top \in \Psi$. The *denotation function* $[\![\_]\!] : \Psi \to 2^{\mathfrak{D}}$ is such that, $[\![\bot]\!] = \emptyset$, $[\![\top]\!] = \mathfrak{D}$, for all $\varphi, \psi \in \Psi$, $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$, $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$, and $[\![\neg\varphi]\!] = \mathfrak{D} \setminus [\![\varphi]\!]$. For $\varphi \in \Psi$, we write $\mathit{IsSat}(()\varphi)$ when $[\![\varphi]\!] \neq \emptyset$ and say that φ is *satisfiable*. $\mathcal{A}$ is *decidable* if $\mathit{IsSat}()$ is decidable.

The intuition is that such an algebra is represented programmatically as an API with corresponding methods implementing the Boolean operations and the denotation function. We are primarily going to use the following two effective Boolean algebras in the examples, but the techniques in the paper are fully generic.

$\mathbf{2^{bv}}^k$ is the powerset algebra whose domain is the finite set $\text{BV}k$, for some $k > 0$, consisting of all nonnegative integers smaller than 2^k, or equivalently, all k-bit bit-vectors. A predicate is represented by a BDD of depth k.[1] The Boolean operations correspond directly to the BDD operations, $\bot$ is the BDD representing the empty set. The denotation $[\![\beta]\!]$ of a BDD β is the set of all integers n such that a binary representation of n corresponds to a solution of β.

SMT^σ is the decision procedure for a theory over some sort σ, say integers, such as the theory of integer linear arithmetic. This algebra can be implemented through an interface to an SMT solver. Ψ contains in this case the set of all formulas $\varphi(x)$ in that theory with one fixed free integer variable x. For example, a formula $(x \bmod k) = 0$, say div_k, denotes the set of all numbers divisible by k. Then $div_2 \wedge div_3$ denotes the set of numbers divisible by six.

2.2 s-WS1S

The symbolic weak monadic second-order logic of one successor (s-WS1S) operating on words over an effective Boolean algebra $\mathcal{A}$ is defined by the following grammar:

$$\varphi := \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \mid \exists X.\varphi \mid S(x,y) \mid x = y \mid x < y \mid P_\psi(y) \mid x \in X$$

where $\psi \in \Psi_\mathcal{A}$ is a predicate over the alphabet theory. First order variables (denoted by lower case letters) range over positions of a string, while second order variables (denoted by upper case letters) range over sets of positions in the string. For example, given the predicate $\psi(r) = r > 0$ over the theory of linear integer arithmetic, the formula

$$\exists x_1.\exists x_2.P_\psi(x_1) \wedge P_\psi(x_2) \wedge x_1 < x_2$$

is true for all the strings $a_1 \dots a_n \in \mathbb{N}^*$ for which there exists two positions $i, j \in [1..n]$ such that the symbols a_i and a_j are both numbers greater than 0 and position i appears before position j (i.e. $i < j$).

The operator that differentiates s-WS1S from WS1S is the unary predicate $P_\psi(x)$. While in WS1S predicates are drawn from a finite signature Σ in our case they can be any predicate in a decidable Boolean algebra. Moreover, while in WS1S any position x in the string satisfies exactly one predicate in Σ (i.e. each character is an element of Σ), in our case each position can satisfy infinitely many predicate. For example the character $r = 6$ satisfies $\psi(r)$ above but also $even(r)$. In particular, since $\psi_1, \psi_2 \in \Psi_\mathcal{A}$ then $\psi_1 \wedge_\mathcal{A} \psi_2 \in \Psi_\mathcal{A}$, it follows that if P_{ψ_1} and P_{ψ_2} are valid predicates in s-WS1S then so is $P_{\psi_1 \wedge_\mathcal{A} \psi_2}$.

We now formally define the semantics of s-WS1S. Given a word $w = a_1 \dots a_n \in \mathfrak{D}^*$, positions $\bar{\imath} = i_1, \dots, i_j \in [1..n]$, position sets $\bar{I} = I_1, \dots, I_k \subseteq [1..n]$, and an s-WS1S formula $\varphi(\bar{x}, \bar{X})$ with j free first-order variables $\bar{x} = x_1, \dots, x_j$ and k free second-order variables $\bar{X} = X_1, \dots, X_k$ we define the semantics using judgements of the form

$$w, \bar{\imath}, \bar{I} \vDash \varphi(\bar{x}, \bar{X}).$$

We define the judgements inductively[2]:

- $w, \bar{\imath}, \bar{I} \vDash \neg\varphi_1(\bar{x}, \bar{X})$ iff $w, \bar{\imath}, \bar{I} \nvDash \varphi_1(\bar{x}, \bar{X})$;

[1] The variable order of the BDD is the reverse bit order of the binary representation of a number, in particular, the most significant bit has the lowest ordinal.

[2] We assume without loss of generality that all quantified variables have distinct names.

- $w, \bar{\imath}, \bar{I} \vDash \varphi_1(\bar{x}, \bar{X}) \wedge \varphi_2(\bar{x}, \bar{X})$ iff $w, \bar{\imath}, \bar{I} \vDash \varphi_1(\bar{x}, \bar{X})$ and $w, \bar{\imath}, \bar{I} \vDash \varphi_2(\bar{x}, \bar{X})$;
- $w, \bar{\imath}, \bar{I} \vDash \exists x.\varphi_1(x \cdot \bar{x}, \bar{X})$ iff there exists $i \in [1..n]$ such that $w, i \cdot \bar{\imath}, \bar{I} \vDash \varphi(x \cdot \bar{x}, \bar{X})$;
- $w, \bar{\imath}, \bar{I} \vDash \exists X.\varphi_1(\bar{x}, X \cdot \bar{X})$ iff there exists $I \subseteq [1..n]$ such that $w, \bar{\imath}, I \cdot \bar{I} \vDash \varphi(\bar{x}, X \cdot \bar{X})$;
- $w, \bar{\imath}, \bar{I} \vDash S(x_l, x_m)$ iff $i_m = i_l + 1$;
- $w, \bar{\imath}, \bar{I} \vDash x_l = x_m$ iff $i_l = i_m$;
- $w, \bar{\imath}, \bar{I} \vDash x_l < x_m$ iff $i_l < i_m$;
- $w, \bar{\imath}, \bar{I} \vDash P_\psi(x_l)$ iff $a_{i_l} \in [\![\psi]\!]_{\mathcal{A}}$;
- $w, \bar{\imath}, \bar{I} \vDash x_l \in X_m$ iff $i_l \in I_m$.

Given a formula φ with no free variables, a word $w \in \mathfrak{D}^*$ is a model of φ iff $w \vDash \varphi$. The language of φ is the following subset of $\mathfrak{D}^*$, $\mathscr{L}(\varphi) = \{w \mid w \vDash \varphi\}$.

2.3 Symbolic Finite Automata

A symbolic finite automaton is a finite automaton over a symbolic alphabet, where edge labels are replaced by predicates. In order to preserve the classical closure operations (intersection, complement, etc.), the predicates must form an effective Boolean algebra.

Definition 1. A *symbolic finite automaton* (*SFA*) M is a tuple $(\mathcal{A}, Q, q^0, F, \Delta)$ where $\mathcal{A}$ is an effective Boolean algebra, called the *alphabet*, Q is a finite set of *states*, $q^0 \in Q$ is the *initial state*, $F \subseteq Q$ is the set of *final states*, and $\Delta \subseteq Q \times \Psi_{\mathcal{A}} \times Q$ is a finite set of *moves* or *transitions*. ⊠

Elements of $\mathfrak{D}_{\mathcal{A}}$ are called *characters* and finite sequences of characters, elements of $\mathfrak{D}_{\mathcal{A}}^*$, are called *words*; ϵ denotes the empty word. A move $\rho = (p, \varphi, q) \in \Delta$ is also denoted by $p \xrightarrow{\varphi}_M q$ (or $p \xrightarrow{\varphi} q$ when M is clear), where p is the *source* state, denoted *Source*(ρ), q is the *target* state, denoted *Target*(ρ), and φ is the *guard* or *predicate* of the move, denoted *Guard*(ρ). A move is *feasible* if its guard is satisfiable. Given a character $a \in \mathfrak{D}_{\mathcal{A}}$, an *a-move* of M is a move $p \xrightarrow{\varphi} q$ such that $a \in [\![\varphi]\!]$, also denoted $p \xrightarrow{a}_M q$ (or $p \xrightarrow{a} q$ when M is clear). In the following let $M = (\mathcal{A}, Q, q^0, F, \Delta)$ be an SFA.

Definition 2. A word $w = a_1 a_2 \cdots a_k \in \mathfrak{D}_{\mathcal{A}}^*$, is *accepted at state p of M*, denoted $w \in \mathscr{L}_p(M)$, if there exist $p_{i-1} \xrightarrow{a_i}_M p_i$ for $1 \leq i \leq k$, such that $p_0 = p$, and $p_k \in F$. The *language accepted by M* is $\mathscr{L}(M) \stackrel{\text{def}}{=} \mathscr{L}_{q^0}(M)$. ⊠

3 Deciding s-WS1S using the Cartesian Product Algebra

The classic decision procedure for WS1S relies on the fact that any formula $\varphi(X_1, \ldots, X_k)$ with k free second-order variables[3] and over a finite alphabet Σ can be compiled into a deterministic finite automaton A that accepts strings over the alphabet $\Sigma \times \{0,1\}^k$, such that every string $w = a_1 \ldots a_n$ accepted by A has the following property: if for each variable X_i we define the set $S_i = \{j \mid \text{the i-th bit of } a_j \text{ is } 1\}$, we then have that

$$w_\Sigma, S_1, \ldots, S_k \vDash \varphi(X_1, \ldots, X_k)$$

where $w_\Sigma \in \Sigma^*$ is the projection of w on its first component.

[3] We only consider second order variable for simplicity but without loss of generality.

Our first decision procedure uses the same idea and transforms a symbolic WS1S formula over the alphabet σ into a symbolic finite automaton over the alphabet $\sigma \times \{0,1\}^k$. As this reduction requires a change of alphabet, we define the concept of a *Cartesian Product Algebra* of two effective Boolean algebras to combine *position*-predicates (over $\{0,1\}^k$) with *character*-predicates (over σ). Given two effective Boolean algebras, $\mathcal{A}_i = (\mathfrak{D}_i, \Psi_i, [\![\_]\!]_i, \bot_i, \top_i, \vee_i, \wedge_i, \neg_i)$ for $i \in \{1,2\}$, their Cartesian Product Algebra $\mathcal{A}_1 \times \mathcal{A}_2$ has the following components:

$$(\mathfrak{D}_1 \times \mathfrak{D}_2, \Psi\ , [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$$

where, for each $\psi \in \Psi$ there exists an effective *sum of products* decomposition $\{(\psi_1^i, \psi_2^i)\}_{i=1}^k \subseteq \Psi_1 \times \Psi_2$, for some finite $k \geq 1$, such that

$$[\![\psi]\!] = \bigcup_{i=1}^{k} [\![\psi_1^i]\!]_1 \times [\![\psi_2^i]\!]_2$$

and, conversely, every such sum of products has an equivalent representation in Ψ . For example, we can let $\bot$ have the effecive decomposition $\{(\top_1, \bot_2)\}$ and we can let $\top$ have the effective decomposition $\{(\top_1, \top_2)\}$. Observe that a sum of products decomposition of $\neg\psi$ is obtained from any sum of products decomposition of ψ using deMorgan's laws.

For an efficient implementation of predicates in Ψ we use *Binary Decision Trees* or *BDTs*. A *BDT for* $\mathcal{A}_1 \times \mathcal{A}_2$ is either

- a *leaf node* $\langle\psi\rangle$ with *label* $\psi \in \Psi_1$, or
- a *nonleaf node* $B = \langle\varphi, B_t, B_f\rangle$ with $\varphi \in \Psi_2$ as its *label* and
  - B_t is a *BDT for* $\mathcal{A}_1 \times \mathcal{A}_2$ called the *true case of* B, and
  - B_f is a *BDT for* $\mathcal{A}_1 \times \mathcal{A}_2$ called the *false case of* B.

The *path condition* to the root of a BDT B is $\top_2$ and if the path condition to a node $\langle\varphi, B_t, B_f\rangle$ in B is π then the path condition to node B_t in B is $\pi \wedge_2 \varphi$ and the path condition to node B_f in B is $\pi \wedge_2 \neg_2\varphi$. In other words, a BDT for $\mathcal{A}_1 \times \mathcal{A}_2$ is a Shannon expansion whose leaf labels or *terminals* are predicates of the algebra $\mathcal{A}_1$ and whose nonleaf node labels or *nonterminals* are predicates of the algebra $\mathcal{A}_2$. The pair (ψ, π) where π is the path condition to a leaf node $\langle\psi\rangle$ of a BDT B is called a *branch* of B. The *sum of products decomposition* of a BDT is the set of all of its branches.

As an implementation side note, subtrees of a BDT are maximally shared, thus the implementation of a BDT is in reality a *Binary Decision Graph* or a *BDG* and in its sum of products decomposition, path conditions to shared nodes have been joined to *summaries* or disjunctions of path conditions.

We say that a BDT B is *well-formed* if for all branches (ψ, π) of B:

- π is satisfiable, and
- either both ψ and $\neg_1\psi$ are satisfiable or else $\psi \in \{\bot_1, \top_1\}$.

For example, the BDT $\langle\top_2, \langle\bot_1\rangle, \langle\top_1\rangle\rangle$ is not well-formed because in the branch $(\top_1, \neg_2\top_2)$ the path condition $\neg_2\top_2$ is unsatisfiable. Observe that both $\langle\top_1\rangle$ and $\langle\bot_1\rangle$ are trivially well-formed. If $\bot_1 \wedge_1 \bot_1$ is not identical to the the predicate $\bot_1$ in Ψ_1 then $\langle\bot_1 \wedge_1 \bot_1\rangle$ is not well-formed.

The two key operations over BDTs are negation and conjunction. Both operations are similar to corresponding BDD operations. Negation is defined as follows.

$$\neg\langle\psi\rangle = \langle\neg_1\psi\rangle, \quad \neg\langle\varphi, B_t, B_f\rangle = \langle\varphi, \neg B_t, \neg B_f\rangle$$

Trivially, negation preserves well-formdness, provided that $\neg_1\bot_1 = \top_1$ and $\neg_1\top_1 = \bot_1$. Conjunction is defined as follows where we assume implicit satisfiability checking and simplification, so that if $\psi \wedge_i \phi$ is unsatisfiable then $\psi \wedge_i \phi = \bot_i$, analogously that valid formulas are simplified to $\top_i$ in both algebras.

$$B \wedge C = B \overset{\top_2}{\wedge} C$$

Where $\overset{\pi}{\wedge}$ carries the path condition π and is defined as follows.

$$\begin{aligned}
\langle \varphi, B_t, B_f\rangle \overset{\pi}{\wedge} C &= \langle \varphi, B_t \overset{\pi\wedge_2\varphi}{\wedge} C, B_f \overset{\pi\wedge_2\neg_2\varphi}{\wedge} C\rangle \\
\langle \psi\rangle \overset{\pi}{\wedge} \langle \varphi, C_t, C_f\rangle &= \begin{cases} \langle \psi\rangle \overset{\pi}{\wedge} C_t, & \text{if unsat}(\pi \wedge_2 \neg_2\varphi); \\ \langle \psi\rangle \overset{\pi}{\wedge} C_f, & \text{else if unsat}(\pi \wedge_2 \varphi); \\ \langle \varphi, \langle \psi\rangle \overset{\pi\wedge_2\varphi}{\wedge} C_t, \langle \psi\rangle \overset{\pi\wedge_2\neg_2\varphi}{\wedge} C_f\rangle, & \text{otherwise.} \end{cases} \\
\langle \psi\rangle \overset{\pi}{\wedge} \langle \phi\rangle &= \langle \psi \wedge_1 \phi\rangle
\end{aligned}$$

There are several optimizations as well as caching of prior results in the concrete implementation that make the implementation feasible but have been omitted here for brevity. If both B and C are well-formed then it follows from the construction that $B \wedge C$ is also well-formed.

The above choice of representation is taylored for our concrete application here. It maintains a semi-canonical form that allows simple checking of satisfiability and validity of predicates in Ψ. In the BDT representation of predicates, we refer to the algebra $\mathcal{A}_2$ as the *nonleaf (or nonterminal) algebra* and $\mathcal{A}_1$ as the *leaf (or terminal) algebra*.

In our current translation from s-WS1S formulas, position-predicates ψ are represented by bitvector predicates in the terminal algebra that is implemented by a BDD solver. The number of bits required in a position predicate corresponds to the maximum number of variables used in the given WS1S formula. The character predicates are predicates over any given character domain, such as Unicode characters, or integers, and are represented by predicates in the nonterminal algebra. We have experimented with both Z3 and a BDD solver as the nonterminal algebra.

An example of an s-WS1S formula involving both kinds of predicates is the following. To be concrete assume that we use regex character classes for nonleaf predicates, such as *digit*, say character-predicate `\d`, *word character*, say `\w`, or *white space*, say `\s`, and the character domain is 16-bit bit-vectors (i.e., unsigned integers less than 2^{16}). A possible s-WS1S statement is: there are at least two positions labeled by word characters and at least one position labeled by a digit; formally:

$$\exists x\, y\, (x < y \wedge P_{\backslash \mathtt{w}}(x) \wedge P_{\backslash \mathtt{w}}(y)) \wedge \exists x(P_{\backslash \mathtt{d}}(x))$$

Observe that a digit is also a word letter, i.e., $\backslash\mathtt{d} \wedge_2 \backslash\mathtt{w}$ is satisfiable in the nonterminal (character) algebra. So for example, a shortest string that satisfies the formula is a string of two word letters where at least one of which is also a digit.

4 Reducing to Monadic Second Order Logic

Our second decision procedures is based on a reduction to the non-symbolic version of WS1S. This reduction is based on the following idea: although the set of predicates and alphabet symbols in a given theory can be infinite, the set of concrete predicates appearing in a concrete formula is finite. Moreover, the set of Boolean combinations (also called minterms) of such

predicates is also finite and each such combination describes an equivalence class for the set of symbols appearing in our alphabet.

More formally, a *minterm* is a minimal satisfiable Boolean combinations of all guards that occur in the s-WS1S formula ψ. For example given two predicates $\Phi = \{\varphi_1, \varphi_2\}$, the set of minterms for Φ is the set

$$M = \{\varphi_1 \wedge \varphi_2, \neg\varphi_1 \wedge \varphi_2, \varphi_1 \wedge \neg\varphi_2, \neg\varphi_1 \wedge \neg\varphi_2\}$$

Unlike for the original set of predicates Φ, all the predicates in Φ are disjoint. Moreover, symbols belonging to the same minterm are *indistinguishable* with respect to the original s-WS1S formula. More formally, for any two symbols $a, b \in \mathfrak{D}$ in the same minterm $\varphi \in M$, and two strings $v, w \in \mathfrak{D}^*$,

$$vaw \in L(\psi) \Leftrightarrow vbw \in L(\psi).$$

Using this idea we reduce the s-WS1S formula ψ to an WS1S formula ψ' over the alphabet M. The reduction is very simple: every predicate $\varphi(x)$ appearing in ψ is replaced by the predicate $\varphi_1(x) \vee \ldots \varphi_m(x)$ where $\{\varphi_1, \ldots, \varphi_m\}$ is the set of minterms in M in which the predicate φ appears positively (i.e. it is not negated). This reduction can in general cause an exponential blow-up. Notice that now our alphabet is M and the new WS1S formula accepts string in M^* and not $\mathfrak{D}^*$.

We will now argue that the $s-WS1S$ formula ψ is satisfiable if and only if the WS1S formula ψ' is satisfiable. If a string $a_1 \ldots a_n$ is accepted by ψ, let $\varphi_1 \ldots \varphi_n$ be the sequence of minterms of the formulas in ψ such that $a_i \in [\![\varphi_i]\!]$ for each $i \leq n$. Since minterms are disjoint, such a sequence is unique. We now need to show that the string $\varphi_1 \ldots \varphi_n \in M^*$ is accepted by ψ'. Whenever $w, x_1, \ldots, x_n \vDash p(x)$ we have to show that in WS1S $w, x_1, \ldots, x_n \vDash \varphi_{i_1}(x) \vee \ldots \vee \varphi_{i_j}(x)$ where $m = \{\varphi_{i_1}, \ldots, \varphi_{i_j}\}$ are the minterms in which p appears positively. From the definition of minterm, it is easy to show that $\varphi_{i_1} \vee \ldots \vee \varphi_{i_j}$ is equivalent to p. Therefore we have if that some symbol $a \in \mathfrak{D}$ is a model of $p(x)$ then its minterm appears in m. The other direction is similar.

5 Future Work

This paper shows some preliminary results on s-WS1S, a symbolic extension WS1S. In particular we propose two decision procedures for the logic s-WS1S. We also have a preliminary implementation of the two decision procedures for s-WS1S, but a comprehensive evaluation of their performance is part of our future work. Moreover, many questions remain unanswered:

- Are there better ways to implement the Cartesian product algebra?
- What is the exact complexity of each operation in such an algebra?
- Can we leverage existing heuristics and techniques for solving WS1S in our symbolic decision procedure [4]?

References

[1] J. Buchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

[2] L. D'Antoni and M. Veanes. Minimization of symbolic automata. In *POPL'14*. ACM, 2014.

[3] G. De Giacomo and M. Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 2013.

[4] J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *TACAS '95*, volume 1019 of *LNCS*. Springer, 1995.

[5] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004.

[6] W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455. Springer, 1996.