# Relativizations for the logic-automata connection[*]

Nils Klarlund
Bell Labs, Lucent Technologies

*Dedicated to the memory of Bob Paige*
*and his contributions to automata algorithms*

September 21, 2004

**Abstract**

BDDs and their algorithms implement a decision procedure for Quantified Propositional Logic. BDDs are a kind of acyclic automata. But unrestricted automata (recognizing unbounded strings of bit vectors) can be used to decide monadic second-order logics, which are more expressive. Prime examples are WS1S, a number-theoretic logic, or the string-based logical notation of introductory texts. One problem is that it is not clear which one is to be preferred in practice. For example, it is not known whether these two logics are computationally equivalent to within a linear factor, that is, whether a formula $\phi$ of one logic can be transformed to a formula $\phi'$ of the other such that $\phi'$ is true if and only if $\phi$ is and such that $\phi'$ is decided in time linear in that of the time for $\phi$.

Another problem is that first-order variables in either version are given automata-theoretic semantics according to relativizations, which are syntactic means of restricting the domain of quantification of a variable. Such relativizations lead to technical arbitrations that may involve normalizing each subformula in an asymmetric manner or may introduce spurious state space explosions.

In this paper, we investigate these problems through studies of congruences on strings. This algebraic framework is adapted to language-theoretic relativizations, where regular languages are intersected with restrictions. The restrictions are also regular languages. We introduce ternary and sexpartite characterizations of relativized regular languages. From properties of the resulting congruences, we are able to carry out detailed state space analyses that allow us to address the two problems.

We report briefly on practical experiments that support our results. We conclude that WS1S with first-order variables can be robustly implemented in a way that efficiently subsumes string-based notations.

[*] Some of the material in this paper appeared in *Computer Aided Verification, CAV '99*, LNCS 1633, 1999, under the title "A theory of restrictions for logics and automata." This work was carried out while the author was with AT&T Labs–Research; it was also supported in part by grant CCR-0341658 from the National Science Foundation.

# 1 Motivation

The relationship between automata and logic has been very successfully exploited through Binary Decision Diagrams[4]. This technique allows formulas of propositional logic to be decided through the use of automata representations for sets of strings of bounded length. But, a more general *logic-automata* connection exists: Büchi[5], Elgot[6], and Trakhtenbrot[17] argued fourty years ago that a logical notation, now called the Weak Second-order theory of 1 Successor or WS1S, would be a more natural alternative to what already was known as regular expressions. WS1S has an extremely simple syntax and semantics: it is a variation of predicate logic with first-order variables that denote natural numbers and second-order variables that denote finite sets of natural numbers; it has a single function symbol, which denotes the successor function, and has usual binary operators such as $\leq, =, \in$ and $\supseteq$.

Büchi, Elgot, and Trakhtenbrot showed that a decision procedure exists for this logic. The idea is to view interpretations as finite strings over bit vectors and then to show by explicit constructions of automata that the set of satisfying interpretations for any subformula is a regular language. In this way, an automaton becomes an object that represents the logical semantics of a subformula, and it makes sense to talk about automata-theoretic semantics, which characterizes the computational approach to the logic. As with Binary Decision Diagrams, the idea behind the decision procedure is to construct inductively a deterministic automaton for each subformula. This method, which we shall review in detail, handles each connective in the logic through an automata-theoretic operation, such as product or subset construction. To *decide* a sentence is then under this view the process of building the automaton inductively.

A main motivation of this article is to make the decision procedure feasible in practice. Since 1994, we have explored the practicality of the logic-automata connection in the *Mona* project, first described in [7]. The Mona tool has been used for a variety of tasks, for example in linguistics[18], pointer verification[13], protocol verification[14], and hardware verification[2]. Among other implementation challenges[12], we discovered spurious state space explosions in intermediate automata. Sometimes, we would be prevented from solving even trivial problems due to phenomena that we will uncover and overcome in the present article.

## 1.1 The two approaches to a logic of one successor

The problems we encountered are in part linked to the existence of two formulations of monadic second-order logic of one successor. Using the same syntax, the two approaches depend on different structures and different semantics.

The logic WS1S, the first approach, is a very natural notation. For example, a sentence in WS1S is either true or false and a formula with $k$ free first-order variables defines a relation that is simply a subset of $\mathbb{N}^k$, where $\mathbb{N}$ is the set of non-negative integers. Thus, we call this approach to the logic-automata connection *number-theoretic*. The automaton for a sentence is a simple one: it has one state (if minimized)! If it is accepting, the formula is true; if rejecting, the formula is false.

The second approach, the one emphasized in presentations of the logic-automata connection (such as in [15, 16]) is more complicated to explain, at least when it comes to the semantics, which is tied to a parameterized, finite, unbounded domain represented by a number $n \geq 0$. This number defines a set $\{0, \ldots, n-1\}$ of *positions*. All second-order terms are interpreted as subsets of $\{0, \ldots, n-1\}$ and all first-order terms are interpreted as numbers in $\{0, \ldots, n-1\}$. Under this view, the truth status of a sentence depends on $n$. The successor function denoted by the term $p+1$ is interpreted as ordinary addition except if $p$ is $n-1$; in that case, we may choose—arbitrarily—to define the meaning of $p+1$ to be $p$.

For example, a sentence may be defined that is true if and only if $n$ is even; intuitively, the sentence expresses:

- there a set $P$ that is maximal (not properly contained in any other set);
- there is a set $P'$ (which is a subset of $P$ since $P$ is maximal) of the even positions in $P$ (which can be expressed: $0 \in P'$ and for any $p$ either $p+1 = p$ or $p+1 \in P' \Leftrightarrow p \in P'$); and
- the maximal element of $P$ is in $P'$.

More generally, a formula with $k$ free first-order variables defines a parameterized family of finite relations, where the $n$th relation is a subset of $\{0, \ldots, n-1\}^k$.

We call this second approach *string-theoretic*, because the semantics can be explained advantageously in terms of strings. The ideas of the automata-theoretic decision procedure of WS1S still apply; in fact, the algorithm becomes simpler, which is the reason why this approach is often preferred in introductions to the logic-automata connection. The string-theoretic approach is also intrinsically appealing for certain applications, for example in the description and verification of parameterized hardware[2]. Among other names, these logics have been called MSO(S)[16], SOM[+][15], and M2L(Str)[7, 9]. They vary slightly, but we will identify them as M2L(Str) in this paper.

## 1.2 Comparison and the translation problem

There are two reasons for preferring the number-theoretic approach. First, its logical semantics is simpler as just argued—it can be explained to people with little mathematical background. Second, WS1S appears to be the stronger logic in the following sense: there is apparently no known polynomial time reduction $f$ from sentences interpreted under the WS1S view to sentences interpreted under the M2L(Str) view such that $f(\phi)$ is true for all $n$ if and only if $\phi$ is true. In contrast, there is a rather obvious, linear translation in the other direction: given a sentence $\phi$ under the string-view, we turn it into a formula $\phi'$ with one free variable \$ such that $\phi$ holds for $n$ if and only if $\phi'$ holds under the interpretation $\$ \mapsto n$. Specifically, $\phi'$ is obtained from $\phi$ by restricting quantified variables to the domain $\{0, \ldots, \$-1\}$. Syntactically, the restriction to $\{0, \ldots, \$-1\}$ for a variable can be expressed as a formula that is conjoined to the formula of the existential quantifier introducing it. Such syntactic constructs are well-known in logic; they are called *relativizations*, see[3]. We call the formula $\phi'$ a *WS1S representation* of the M2L(Str) formula $\phi$.

A main focus of the present work is to show how relativizations can be guaranteed to work in practice. In fact, even though the syntactic translation is linear, there is no guarantee that the computations involved in calculating the automaton for $\phi'$ (under the WS1S view) are not asymptotically more involved than those involved in $\phi$ (under the M2L(Str) view).

Indeed, during early experiments with this procedure, our problem was that seemingly innocuous formulas would yield enormous automata after the conversion into WS1S. So, we say that an *efficient translation algorithm* is one that in linear time transforms any sentence $\phi$ in M2L(Str) to a representation $\phi'$ such that $\phi'$ is decided in time that is linear in the time to decide $\phi$. Let us call the question of finding such an algorithm the *translation problem.*

## 1.3 Handling of first-order variables

Another computational problem we encountered with monadic second-order logics stems from the way that first-order variables and terms are handled. Through formula rewritings, they are transformed into second-order variables. The second-order variables are subjected to relativizations that restrict them to singleton sets. Consequently, automata corresponding to subformulas are not simply determined by the logical semantics, but also according to how relativizations are formulated. So, to make automata for formulas canonically determined, extra automata product operations are used to *conjunctively normalize* these intermediate automata as we shall see. (The canonicity of representations is essential to the success of automata-based methods such as BDDs—it guarantees that intermediate results are always pruned to their minimum size.) The *first-order semantics problem* is to find an automaton representation that is no bigger than the conjunctively normalized representation, while not requiring such explicit normalization steps. This is important in practice, since we want to minimize the amount of computational work.

We note that there are other ways of deciding WS1S, for example through Ehrenfeucht-Fraissé games[15] or through bounded-model techniques[1].

## 1.4 Contributions of this paper

In this paper, we propose solutions to the translation problem and the first-order semantics problem. We do so by studying relativizations in an algebraic framework. We proceed as follows.

We formulate a syntax for WS1S, where relativizations are made explicit, and we provide initially three different automata-theoretic semantics: (1) the *ad hoc* semantics that corresponds to the strategy we first used in the Mona implementation for first-order variables, (2) the *conjunctively normalized semantics*, where *all* the intermediate automata are conjoined with relativizations, and (3) the *ternary semantics*, which is based on valuations that identify the membership status (0 or 1) for each string in the restriction and assigns $\perp$ to each string not in the restriction. We explain why the ad hoc semantics is unsuitable, and why the conjunctively normalized semantics, in addition to being asymmetric, would slow down the decision procedure. We show that the ternary semantics makes most normalizations unnecessary, since they inherently propagate through the automata-theoretic constructions. Also, we indicate how the ternary semantics can be implemented based on the standard WS1S decision procedure.

To study the question of automata sizes, we give a detailed congruence-theoretic analysis of regular languages under relativizations, that is, under intersections with other regular sets that act as restrictions. We introduce a notion of a *thin* language, and we show that the relativizations occurring in the treatment of first-order variables and in the translation problem are thin. We prove that languages under thin relativizations make comparisons of the conjunctively normalized semantics and the ternary semantics easy: the latter is the same as the former except for some extra equivalence classes that we characterize. We show that if the automata of restrictions are bounded then the sizes of intermediate automata occurring under the ternary semantics are to within this additive bound the same as the sizes of automata of the conjunctively normalized semantics.

We strengthen this result by exhibiting congruences based on a *sexpartite semantics* that are no bigger than those of the conjunctively normalized semantics. The sexpartite valuations are based on the ternary ones, but for certain strings they do not yield an exact answer to what the ternary evaluation is. We are able to show that operations corresponding to logical connectives may be formulated directly on automata representing these congruences, under the further assumption of *crispness*, a language-theoretic property that the restrictions of the first-order semantics problem and the translation problem enjoy.

Our main result is that the resulting decision procedure, while requiring only few normalizations, involves intermediate automata that are at most the same in size (to within a linear factor) as the ones occurring under the conjunctively normalized semantics, but sometimes only logarithmic in size compared to the conjunctively normalized semantics. Thus, we have found a symmetric and efficient representation of formulas under restrictions.

We conclude that WS1S, and not a string-oriented logic, is the best interface to the logic-automata connection, since in practice the string-theoretic view is effectively subsumed by the number-theoretic view through the techniques developed in this article.

## 1.5 Organization

In Section 2, we review WS1S and its decision procedure. We provide further motivation for why normalizations are necessary, both in the case of first-order variables (Section 2.3) and in the case of the translation problem (Section 2.4). For the latter, we discuss an example in detail.

In Section 3, we formalize the classic semantics, the conjunctively normalized one, and the ternary one. We also discuss the relationship between a syntax that explicitly accommodates relativizations and the automata-theoretic semantics.

In Section 4, we develop an understanding of restrictions imposed on regular languages through ternary valuations. In particular, Theorem 1 relates sizes of automata under the conjunctively normalized semantics and under the ternary semantics.

In Section 5, we present state space engineering techniques that remove information from ternary valuations. The resulting sexpartite valuations are specialized ternary valuations. We discuss ways of calculating sexpartite valuations and how to reason about them.

In Section 6, we study the problem of calculating the sexpartite representation of intersections directly from the sexpartite representation of the sets involved.

In Section 7, we show how the sexpartite semantics of WS1S can be reformulated in ways that will support better algorithms; in particular, we try to avoid normalizations.

In Section 8, we present the algorithms that may be used in a decision procedure for WS1S based on techniques from the Sections 6 and 7. In particular, we formulate algorithms that allow us to formulate our second theorem: WS1S under restrictions of the kinds we are interested in can be decided in a way that is up to exponentially faster than using the conjunctive normalization technique.

In Section 9, we summarize and provide some hints about the practical performance of the Mona tool when equipped with techniques of this article.

# 2 WS1S: review and issues

We need to fix a syntax for WS1S. We follow the concrete, ASCII-based syntax of Mona, but we keep only a small set of primitives. *Nutshell WS1S* can be presented as follows. A formula $\phi$ is *composite* and of the form `~`$\phi'$ (negation), $\phi'$ `&` $\phi''$ (conjunction), or `ex2` $P^i : \phi'$ (existential second-order quantification), or it is *atomic* and of the form $P^i$ `sub` $P^j$ ($P^i$ is a subset of $P^j$), $P^i$ `<` $P^j$ (elements in $P^i$ are less than elements in $P^j$), $P^i$ `=`$P^j \setminus P^k$ ($P^i$ is the set difference of $P^j$ with $P^k$), or $P^i$ `=`$P^j$ `+1` ($P^i$ are the successors of elements in $P^j$). Here, we have assumed that variables are all second-order and named $P^i$, where $i \geq 1$. Other comparison operators, second-order terms with set-theoretic operators, and Boolean connectives can be introduced by trivial syntactic abbreviations, see [11, 16]. The treatment of first-order terms is discussed later.

## 2.1 Logical Semantics of WS1S

A decision procedure takes as input a formula $\phi_0$, called the main formula, whose truth status is to be investigated. Following standard practice, we sometimes regard the main formula as an abstract syntax or parse tree (with its root facing up—the usual convention). We define its *logical semantics* (or just *semantics*) inductively relative to a string $w$ over the alphabet $\Sigma = \mathbb{B}^k$, where $\mathbb{B} = \{0, 1\}$ and $k$ is the number of variables in $\phi_0$. We assume that $\phi_0$ is closed and that each variable is bound in at most one occurrence of an existential quantifier. Generally, we consider only formulas that are subformulas of $\phi_0$, since the semantics is formulated in terms of strings over a bounded alphabet—something that prevents us from giving semantics to all possible formulas given a value of $k$. We now regard a string $w = a_0 \cdots a_{\ell-1}$, where $\ell = |w|$ is the length of $w$, to be of the form:

$$\begin{array}{l} P^1 \\ \cdots \\ P^k \end{array} \qquad\qquad \begin{pmatrix} a_0^1 \\ \cdots \\ a_0^k \end{pmatrix} \cdots \begin{pmatrix} a_{\ell-1}^1 \\ \cdots \\ a_{\ell-1}^k \end{pmatrix}$$

where we have indicated (left) that if the string is viewed as a matrix, then row $i$ is called the $P^i$*-track*. Each letter $a$ is sometimes written in a transposed notation as the vector $(a^1, \ldots, a^k)^t$. The interpretation $w(P^i) \subseteq \mathbb{N}$ of $P^i$ defined by $w$ is the finite set $\{m \mid \text{the } m\text{th bit in the } P^i\text{-track is } 1\}$. Note that suffixing $w$ with a *null-extension*, a string of the form $\mathbf{0} \cdots \mathbf{0}$ with $\mathbf{0} = (0, \ldots, 0)^t$, does not change the interpretation of any variable.

The semantics of a formula $\phi$ can now be defined inductively relative to an interpretation $w$. We use the notation $w \vDash \phi$ (which is read: $w$ satisfies $\phi$) if the interpretation defined by $w$ makes $\phi$ true:

$$\begin{array}{lll}
w \vDash \texttt{\~{}}\phi' & \text{iff} & w \nvDash \phi' \\
w \vDash \phi' \ \texttt{\&} \ \phi'' & \text{iff} & w \vDash \phi' \text{ and } w \vDash \phi'' \\
w \vDash \texttt{ex2}\ P^i : \phi' & \text{iff} & \exists \text{ finite } M \subseteq \mathbb{N} : w[P^i \mapsto M] \vDash \phi' \\
w \vDash P^i \ \texttt{sub} \ P^j & \text{iff} & w(P^i) \subseteq w(P^j) \\
w \vDash P^i \ \texttt{<} \ P^j & \text{iff} & \forall h \in w(P^i) : \forall k \in w(P^j) : h < k \\
w \vDash P^i = P^j \backslash P^k & \text{iff} & w(P^i) = w(P^j) \backslash w(P^k) \\
w \vDash P^i = P^j \ \texttt{+1} & \text{iff} & w(P^i) = \{m + 1 \mid m \in w(P^j)\}
\end{array}$$

where we use the notation $w[P^i \mapsto M]$ for the minimal string $w'$, called the *witness string*, that is at least as long as $w$ and interprets all variables $P^j$, $j \neq i$, as $w$ does, but interprets $P^i$ as $M$. Note that the truth status of a subformula $\phi$ that contains an existential formula of the form $\texttt{ex2}\ P^i : \phi'$ does not depend on the $P^i$-track, since we assume that each $P^i$ is bound by at most one existential quantifier, and consequently there can be no free occurrence of $P^i$ in $\phi$. We could have chosen a model where these unnecessary tracks are removed—but that gives us the complication of working with different alphabets for each subformula.

To fully explain the decision procedure for WS1S, we need to look at the situation for quantification in more detail. This discussion will be rather technical; it is the reason why introductory texts concentrate on M2L(Str) for which existential quantification is straightforward. Looking at the witness string $w[P^i \mapsto M]$ again, we observe that it may be longer than $w$, since the greatest element in $M$ may be equal to or greater than the length of $w$. In this case, the extension consists of letters that are all zeros except for the $P^i$-track, which describes the elements of $M$ greater than $w - 1$.

To characterize such extensions more formally, we let $\Sigma_0^i$ be the set of all letters of the form $(0, \ldots, 0, X, 0, \ldots, 0)^t$ (where the $X$ means that the value of the $i$th component is either 0 or 1). An extension $z^i$ in ${\Sigma_0^i}^*$ is then called a *null-but-$i$ extension*. Thus, the witness string is of the form $w' \cdot z^i$, where $w'$ is the same string as $w$ except possibly for the $P^i$-track.

Generally, we may decompose any $w$ into parts $\tilde{w}$ and $z^i$ such that $z^i$ contains as many letters from the end of the string as possible that are all 0 outside the $P^i$-track; more precisely, we let $w = \tilde{w} \cdot z^i$, where $z^i$ is a maximal, null-but-$i$ extension of some prefix of $w$. Thus, $\tilde{w}$ is empty or at least one non-$P^i$-track in $\tilde{w}$ ends with a 1, i.e. the last letter in $\tilde{w}$ is not in $\Sigma_0^i$. We say that $\tilde{w}$ is *but-$i$-minimal*.

As defined, the witness string may not be shorter than $w$. For example, if $w$ is all zeros outside the $P^i$-track, with the $P^i$-track consisting of all ones and $M = \emptyset$, then the witness string is $\mathbf{0} \cdots \mathbf{0}$ of length $|w|$. We have introduced this requirement to avoid other unpleasant technicalities later.

For any formula $\phi$, we associate the *language* $L_\phi = \{w \mid w \vDash \phi\}$ of satisfying interpretations. The truth value of $\phi_0$ is independent of $w$, since by assumption it is a closed formula. Thus, $\phi_0$ is either true, when $L_{\phi_0} = \Sigma^*$, and we write $\models \phi_0$, or $\phi_0$ is false, when $L_{\phi_0} = \emptyset$, and we write $\not\models \phi_0$.

**Proposition 1** Words $w$ and $w'$ that interpret all variables in the same way satisfy the same set of formulas: if for all $i$, $1 \leq i \leq k$, $w(P^i) = w'(P^i)$, then for all $\phi$, $w \models \phi$ if and only if $w' \models \phi$. Thus, the interpretation is invariant under null-extensions.

**Proof** (Idea) By a simple induction on formulas $\phi$, we may show that for words $w$ and $w'$ such that $w'$ is a null-extension of $w$ the following holds: either both satisfy $\phi$ or both do not satisfy $\phi$. For existential quantification, we note that if $w'$ is a null-extension of $w$, then $w'[P^i \mapsto M]$ is a null-extension of $w[P^i \mapsto M]$. □

## 2.2 Automata-theoretic semantics

The automata-theoretic semantics constitutes a decision procedure that associates to each $\phi$ the deterministic, minimal automaton $A_\phi$ accepting the language $L_\phi$.

### 2.2.1 Automata preliminaries

We recall that an automaton $A = (\Sigma, Q, Q_0, \rightarrow, Q^F)$ consists of an alphabet $\Sigma$, which we here assume to be $\mathbb{B}^k$, a finite set of states $Q$, a set of initial states $Q_0$, a transition relation $\rightarrow \subseteq Q \times \Sigma \times Q$, and a set of final states $Q^F$. A *run* $(q_m)_{m \leq \ell}$ over a word $w = a_0 \cdots a_{\ell-1} \in \Sigma^*$ is a sequence of states $q_0, \ldots, q_\ell$ such that $q_0 \in Q_0$ and for all $m$, $0 \leq m < \ell$, $(q_m, a_m, q_{m+1}) \in \rightarrow$. The *length* of the run is $\ell$. The run is *accepting* if $q_\ell \in Q^F$, and the language accepted by $A$ is the set of $w$ that allows some accepting run. The automaton $A$ is deterministic if $Q_0$ is a singleton and if for all $q \in Q$ and $a \in \Sigma$ there is exactly one $q'$ such that $(q, a, q') \in \rightarrow$. Our automata are assumed deterministic if not otherwise indicated. For a deterministic automaton, any word $w$ allows exactly one run; if its length is $\ell$, then last state $q_\ell$ of this run is denoted $\textsc{last}\, A(w)$ and we say that $w$ *brings* $A$ to state $q_\ell$. Moreover, with $\mathbb{B} = \{0, 1\}$ (equipped with the usual truth functions $\wedge, \vee$, and $\neg$), we also let $A$ denote the characteristic function of $L$; in other words, $A : \Sigma^* \rightarrow \mathbb{B}$ is defined by $A(u) = (\textsc{last}\, A(w) \in Q^F)$.

The *size* $|A|$ of an automaton is its number of states.

### 2.2.2 Constructing $A_\phi$

For atomic formulas $\phi$, a small, deterministic, minimal automaton can be directly constructed that accepts the language $L_\phi$. For example, for the formula $P^1 \,\mathtt{sub}\, P^2$, a two-state automaton exists that accepts exactly the set of $w$ for which $w(P^1) \subseteq w(P^2)$; see Figure 1, where we have followed usual conventions: states are circles, the initial state is denoted by an arrow pointing to it, transitions are denoted by arrows marked with letters for which they apply, and the final states are designated by an inner circle.

Other atomic formulas are treated similarly, and for composite formulas we proceed by induction.

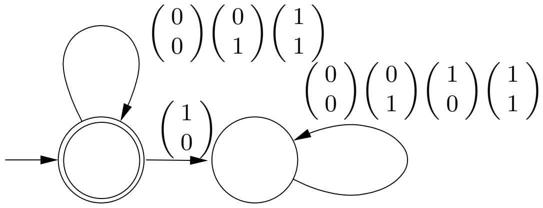

$\binom{0}{0}\binom{0}{1}\binom{1}{1}$
$\binom{1}{0}$
$\binom{0}{0}\binom{0}{1}\binom{1}{0}\binom{1}{1}$

Figure 1: Automaton that accepts satisfying interpretations for $P^1 \,\texttt{sub}\, P^2$.

For a formula $\phi$ of the form ~$\phi'$, the automaton $A_\phi$ is taken to be the complement of the automaton $A_{\phi'}$ calculated by induction. This automaton will be minimal by construction since it is obtained by simply reversing final and non-final states in the automaton $A_{\phi'}$. The case of conjunction is handled by an automata-theoretic product construction: given $A_{\phi'}$ accepting $L_{\phi'}$ and given $A_{\phi''}$ accepting $L_{\phi''}$, we construct the minimized product automaton of $A_{\phi'}$ and $A_{\phi''}$; this automaton accepts the language $L_{\phi'} \cap L_{\phi''}$.

The case of quantification is more complicated. Consider $\phi = \texttt{ex2}\ P^i : \phi'$. Define the projection operator $\textsc{proj}^i$ that transforms functions of type $\Sigma \to \mathbb{B}$ according to:

$$\textsc{proj}^i A(u) \quad = \quad \begin{cases} 1 & \text{if } \exists M : A(u[P^i \mapsto M]) = 1 \\ 0 & \text{if } \forall M : A(u[P^i \mapsto M]) = 0 \end{cases} \tag{1}$$

Then, we desire to make $A_\phi$ an automaton that recognizes $\textsc{proj}^i A_{\phi'}$. The quantification over unbounded (but finite) sets seems to contradict our use of finite strings. We describe how to calculate $\textsc{proj}^i$ in two steps. In the first step, knowledge about possible future extensions that only interpret the $P^i$-track is used to change the labeling of the $A_{\phi'}$ automaton. In the second step, a conventional projection construction is carried out.

From an automaton $A$, we construct the *futurization* of $A$, denoted $\textsc{fut}^i A$ according to null-but-$i$-extensions:

$$\textsc{fut}^i A(u) = \begin{cases} 1 & \text{if } \exists v \in {\Sigma_0^i}^* : A(u \cdot v) = 1 \\ 0 & \text{if } \forall v \in {\Sigma_0^i}^* : A(u \cdot v) \neq 1 \end{cases}$$

This description can clearly be implemented as a linear time automata-algorithm by changing the labeling of the states of $A$ appropriately.

Define the *bounded projection* operator $\textsc{bproj}^i$ such that

$$\textsc{bproj}^i A(u) \quad = \quad \begin{cases} 1 & \text{if } \exists M \text{ with } \max M < |u| : A(u[P^i \mapsto M]) = 1 \\ 0 & \text{if } \forall M \text{ with } \max M < |u| : A(u[P^i \mapsto M]) = 0 \end{cases}$$

This operator describes the traditional projection operation on the $P^i$ track. It can be formulated as an algorithm that yields a deterministic automaton via the subset construction. The algorithm may run in exponential time, since the resulting automaton may be exponentially bigger.

**Proposition 2**

$$\textsc{proj}^i A = \textsc{bproj}^i(\textsc{fut}^i A)$$

**Proof** We have

$$\begin{array}{l}\textsc{proj}^i A(u) = 1 \text{ iff} \\ \exists M : A(u[P^i \mapsto M]) = 1 \text{ iff} \\ \exists \hat{M}, z^i \text{ with } \max \hat{M} < |u| \text{ and } z^i \in {\Sigma_0^i}^* : A(u[P^i \mapsto \hat{M}] \cdot z^i) = 1 \text{ iff} \\ \textsc{bproj}^i(\textsc{fut}^i A)(u) = 1\end{array}$$

where the second biimplication is valid because $|u[P^i \mapsto M]| \geq |u|$ holds thanks to the definition of $u[P^i \mapsto M]$. □

## 2.3 Automata-theoretic semantics of first-order variables

Let us look at the first-order semantics problem. Adding first-order variables to nutshell WS1S can easily be done as follows: a first-order variable $p$ is regarded as a second-order term $P$ that is restricted to take on values that are singleton sets, whose sole element denotes the value of $p$, see [10, 15, 16]. This relativization is imposed syntactically by conjoining a singleton predicate `singleton(`$P$`)` to the formula where $P$ is quantified; the symbol $P$ is a meta-variable that stands for one of the $P^i$. The predicate `singleton(`$P$`)` is called the *restriction* of the relativization.

(The singleton predicate can be expressed as a formula in nutshell logic for any particular $P^i$ (the formula expresses that $P^i$ is nonempty and that any two elements in $P^i$ are the same). Thus, we view this predicate as a macro, a textual expansion mechanism, not an additional primitive predicate that must be interpreted.)

Unfortunately, relativization is not entirely robust. Interpretations $w$ not fulfilling `singleton(`$P$`)` take on "arbitrary" truth values for subformulas in the parse tree below the point of relativization. We explain this arbitrariness through an example that assumes that we have already added the ability to express both first-order and second-order constants to the nutshell language. The formula $\phi = p$`=0`, where $p$ is first-order, is then reasonably represented as $\{0\}$ `sub` $P$, since this formula has the right truth value relative to singleton interpretations of $P$. Similarly, we would reasonably translate the formula $\phi' = p < 1$ into $P$`<`$\{1\}$. Of course, $\phi$ and $\phi'$ are to be considered equivalent formulas, because there are interpreted over $\mathbb{N}$. But $\{0\}$ `sub` $P$ and $P$`<`$\{1\}$ are not equivalent as WS1S formulas, because they take on "arbitrary" truth values outside the restriction. For example, for $w = 00$ (which interprets $P$ as $\emptyset$, thus making no sense of $p$) makes $\{0\}$ `sub` $P$ false, but $\{0\}$ `sub` $P$ true.

This phenomenon leads to the following problem: automata corresponding to subformulas may have many states that describe spurious and irrelevant truth functions outside the restrictions. There might be so many, as we shall see, that the decision procedure breaks down in practice. Of course, the extra states corresponding to interpretations outside the restriction are eventually pruned thanks to the conjunction of the restriction in the relativized quantified formula. For example, if $p$ is removed as a free variable through first-order quantification in $\texttt{ex1}\ p : p\texttt{=0}$ and we represent $p\texttt{=0}$ by $\{0\}\ \texttt{sub}\ \ P$, then the classic relativization conjoins the restriction at the point of quantification so that we obtain $\texttt{ex2}\ P : \{0\}\ \texttt{sub}\ P\ \texttt{\&}\ \texttt{singleton(}P\texttt{)}$.

In order to fix the automata semantics of subformulas, we could arbitrate as follows: conjoin the restriction to every subformula $\phi$ in a procedure we call *normalization*. Then—after settling various technical choices of the normalization process—we would have a firm automata-theoretic explanation of the language $L(\phi)$ under what we call the *conjunctively normalized semantics*. (We define the various semantics formally in Section 3.)

The practical problem with this solution is that additional product and minimization calculations would be necessary: for each automaton $A$ representing a subformula $\phi$ and each free variable $P^i$, the automaton representing the singleton property for $P^i$ must be conjoined to $A$. Such extra calculational work slows down the decision procedure in the following sense: each automata-theoretic operation must be followed by a product and a minimization. (Under certain assumptions on restrictions, the need for minimization may disappear, see Section 4.) For example, complementation, which is normally very fast since it consists of flipping acceptance statuses of states, now would involve a product and a minimization operation.

In practice, the Mona implementation prior to the one implemented with the results of the present article used the *ad hoc semantics*: the restriction for variable $p$ is conjoined only to atomic formulas where $p$ occur and to the formula in the existential quantification introducing $p$. Note that this technique does not eliminate the problem of spurious behavior for intermediate formulas. For example, the conjunctively normalized atomic formula $p\texttt{=0}$ is not equivalent to the negation of the conjunctively normalized atomic formula $p\texttt{\~{}=0}$, where $\tilde{}=$ means "not equal."

## 2.4 Emulation of string semantics in WS1S

We turn to the translation problem of how to use restrictions to efficiently translate the string-theoretic version of monadic second-order logic into the number-theoretic version. A simple choice of syntax for M2L(Str) is to make it identical to nutshell WS1S syntax. The satisfaction relation is now denoted $\vDash_{string}$; it is the same as for WS1S except that quantification and successors are treated relative to $|w|$:

$$\begin{array}{lll} w \vDash_{string} \texttt{ex2}\ P^i : \phi' & \text{iff} & \exists M \subseteq \{0,\ldots,|w|-1\} : w[P^i \mapsto M] \vDash_{string} \phi' \\ w \vDash_{string} P^i = P^j\ \texttt{+1} & \text{iff} & w(P^i) = \{m+1 \mid m \in w(P^j) \text{ and } m+1 < |w|\} \end{array}$$

where the notation $w[P^i \mapsto M]$ again denotes the string $w$ altered so that the $P^i$ track describes $M$. Note that here $w[P^i \mapsto M]$ is exactly as long as $w$ since $M$ is a subset of $\{0,\ldots,|w|-1\}$. The interpretation of $\phi_0$ on a string of $w$ still does not depend on the individual tracks of $w$, but it *does* depend on the length of $w$. Thus we write $i \vDash_{string} \phi_0$ if $\phi_0$ holds for a string $w$ of length $i$.

To emulate $\vDash_{string}$ in $\vDash$, we must relativize all second-order terms to sets of numbers less than the last position in the string. Thus, we introduce a first-order variable \$ that simulates the entity $|w|$. (Of course, \$ really stands for some $P_i$ variable that is otherwise unused and that is relativized to act like a first-order variable.) A

\$-*restriction* for a variable expresses that the variable is a subset of $\{0, \ldots, \$ - 1\}$. Then, under the normalization strategy we conjoin \$-constraints for all free variables of each subformula. The result is a WS1S formula $\phi'$ with one free variable \$ such that $m \vDash_{string} \phi \;\Leftrightarrow\; w \vDash \phi'$, where the \$-track of $w$ interprets \$ as $m$. For example, the formula $\texttt{ex1}\ p : \texttt{ex1}\ q : p = q$ becomes in WS1S:

$$\begin{array}{l}
[\texttt{singleton}(\$)\ \texttt{\&} \\
\ \texttt{ex2}\ P : \texttt{ex2}\ Q : \\
\qquad [\texttt{singleton}(P)\ \texttt{\&}\ \texttt{singleton}(Q)\ \texttt{\&}\ \texttt{singleton}(\$) \\
\qquad\ \texttt{\&}\ P\ \texttt{<}\ \$\ \texttt{\&}\ Q\ \texttt{<}\$\ \texttt{\&}\ P\texttt{sub}\,Q\ \texttt{\&}\ Q\ \texttt{sub}\,P]]
\end{array}$$

as expressed in nutshell syntax, where each normalized subformula is enclosed in brackets. The M2L(Str) formulation is

$$\begin{array}{l}
[\texttt{ex2}\ P : \texttt{ex2}\ Q : \\
\qquad [\texttt{singleton}(P)\ \texttt{\&}\ \texttt{singleton}(Q)\ \texttt{\&} \\
\qquad P\texttt{sub}\,Q\ \texttt{\&}\ Q\ \texttt{sub}\,P]]
\end{array}$$

**Proposition 3** Under the translation outlined above, the minimized, canonical automata arising during the M2L(Str) decision procedure are essentially the same as the ones arising during the WS1S procedure except for at most two additional states.

**Proof** (Some insights) First, we must establish the relationship between the meaning of a formula $\phi$ in M2L(Str) and the meaning of $\overline{\phi}$ in WS1S, where $\overline{\phi}$ is obtained by conjoining the \$-restriction for each variable occurring free in $\phi$. If there are $n$ variables in $\phi$, then the \$-variable receives index $n+1$. Let $\overline{w}$ be a string interpreting these $n+1$ variables. An inductive argument shows that $\overline{w} \vDash \overline{\phi}$ if and only if (1) the $P^{n+1}$-track is interpreted as a singleton $\{\ell\}$ and (2) for each free variable $P^i$ in $\phi$ the $P^i$-track is interpreted as a set of numbers less than $\ell$.

Second, we can use this knowledge to construct the WS1S automaton for $\overline{\phi}$ from the M2L(Str) automaton $\phi$ by adding states $s_{\mathrm{accept}}$ (an accepting state) and $s_{\mathrm{reject}}$ (a rejecting state). The transition relation of the new automaton is the same as for the old one as long as the additional $P^{n+1}$-component is $0$. All old states are turned into rejecting states. When the \$-component is $1$, corresponding to the end of the string under the M2L(Str) representation, a transition is made to $s_{\mathrm{accept}}$ or $s_{\mathrm{reject}}$ according to the accept status of the state that would have been reached in the old automaton, provided that the $P^i$-component of all free variables is 0 (if the latter is not true, then a transition is made to $s_{\mathrm{reject}}$). From $s_{\mathrm{accept}}$, a transition is made to $s_{\mathrm{reject}}$ if any $1$ occurs in the \$-component (since \$ must be interpreted as a singleton set) or in the track corresponding to a free variable in $\phi$. The $s_{\mathrm{reject}}$ state is connected to itself on all letters.

Finally, it can be shown that during minimization of the new automaton, every pair of any old states are still not equivalent with respect to the canonical equivalence relation: the transitions to the two new states induce the same partition of the old states, regarded as part of the new automaton, as the one defined by accepting or rejecting states of the old automaton. □

Our practical experiments with running string-based examples translated into WS1S were based on the ad hoc strategy, where the restriction of variable is conjoined only to atomic formulas involving the variable and to the place where the variable is introduced by a quantifier.

The problem that bloated automata may occur thanks to the non-robustness of the ad hoc strategy is not just a theoretical one. We discovered the following problem that was serious enough to prevent benign formulas from being decided.

**Parity example** Consider the formula $\phi_{\mathrm{ODD}}$ = $\texttt{ex1}\,p: (p\,\texttt{in}\,P^1\ \oplus\cdots\oplus\ p\,\texttt{in}\,P^n)$ under the string-theoretic semantics, where $\oplus$ denotes addition modulo 2 (properly formulated in nutshell syntax). The formula holds if and only if there is a position contained in an odd number of the sets $P^i$. Translated into nutshell WS1S under the ad hoc strategy, the formula becomes:

$$
\begin{array}{ll}
\texttt{singleton(}\$\texttt{)}\ \texttt{\&} & \\
P^1\texttt{<}\ \$\ \texttt{\&} & \\
\cdots & \\
P^n\texttt{<}\ \$\ \texttt{\&} & \\
\texttt{ex2}\ P\texttt{:} & (\texttt{singleton(}\$\texttt{)}\ \texttt{\&} \\
& ((P\,\texttt{sub}\,P^1\ \texttt{\&}\ \texttt{singleton(}P\texttt{)}\ \texttt{\&}\ \texttt{singleton(}\$\texttt{)}\ \texttt{\&}\ P^1\texttt{<}\ \$) \\
& \ \oplus((\cdots) \\
& \quad \oplus\ (P\,\texttt{sub}\,P^n\ \texttt{\&}\ \texttt{singleton(}P\texttt{)}\ \texttt{\&}\ \texttt{singleton(}\$\texttt{)}\ \texttt{\&}\ P^n\texttt{<}\ \$)\ldots).
\end{array}
\tag{2}
$$

where $\oplus$ is a binary operator defined in terms of & and ~.

**Proposition 4** The parity formula $\phi_{\mathrm{ODD}}$ expressed as (2) produces intermediate automata whose size is doubly exponential in $n$ when constructed according to Section 2.2.2. But if restrictions are conjoined to all subformula (that is, also to each intermediate $\oplus$ formula), then all intermediate automata have at most 21 states.

**Proof** (Intuition) Instead of providing a genuine proof, we provide some intuition and experimental data. Initially, let us discuss the size of the minimal automaton for $\phi_{\mathrm{ODD}}$. The alphabet is $\mathbb{B}^{n+2}$, since the formula contains $n$ variables $P_i$ and the variables $\$$ and $p$, whose real names in nutshell logic are, say, $P_{n+1}$ and $P_{n+2}$. But since $p$ is quantified away, the $p$-track does not influence the way the minimal automaton works. So, we will regard the automaton as reading letters that are vectors of size $n+1$. The automaton must check for each letter whether the number of ones among the first $n$ tracks is odd . The automaton for $n = 2$ is shown in Figure 2 as generated by Mona and the Graphviz drawing program (the state labeled 0 is an artifact of Mona's representation of Boolean variables—it can be ignored). Intuitively, the automaton can be explained as follows, where states are named as in Figure 2. The initial state is 1, where the automaton can stay until it finds a letter with an odd number of ones among tracks $P_i$ with $i \leq n$. On such a letter it proceeds to state 3 recording this fact, and it stays there. So far, we have assumed that the end of the emulated string has not been reached, that is, a 1 in the $P_{n+1}$-track has not yet occurred. When this 1 occurs, the automaton proceeds to either a rejecting state $2$ from which it cannot escape or an accepting state $4$—according to which of the two previously mentioned states it was in. The automaton leaves accepting state $4$ if another occurrence of a $1$ happens in track $P_{n+1}$, since that violates the constraint on $\$$ as a first-order variable.

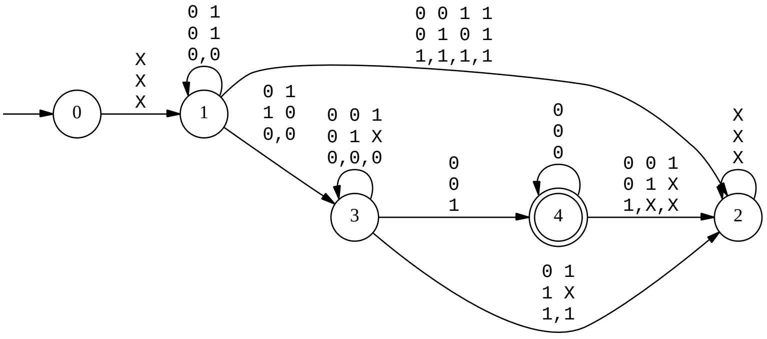


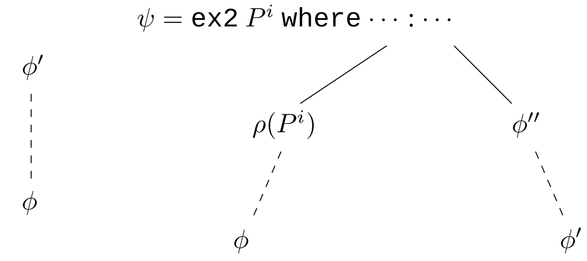
Figure 2: Automaton emulating string semantics for $n = 2$ of parity example.

Now consider the subformula inside the quantifier `ex2` $P$ in (2). It has $P$ as a free variable. The translation of this formula results in a minimal automaton whose size is exponential in $n$. Intuitively, this explosion stems from the need to record the status of whether position $m$ is in $P^i$, for each $1 \leq i \leq n$, where $m$ is the first position in $P_{n+1}$—if this information is not recorded, then it is impossible for the automaton to figure out the eventual truth value of the $i$th summand. In fact, when $m \in P^i$, the truth value of the $i$th summand still becomes false if the $P^i$-track contains any 1s further out than the number designated by $\$$, because of the last conjoint in each summand.

Moreover, it can be seen that the subset construction applied in connection with eliminating the $P$-variable will yield a further exponential blow-up. Intuitively, what happens is that the subset construction results in automaton that records the set of all vectors seen, since it must guess the value of $m$.

While we do not present a formal proof, the experimental behavior supports the intuition just given: for $n = 1, 2$ and, $3$, the size of the automaton corresponding to the subformula `ex2` $P$ is $21, 265$, and $65553$, approximately $2^{2^2}$, $2^{2^3}$, and $2^{2^4}$. It is not possible to calculate the automaton for $n = 5$.

Finally, we have experimentally found that for $n = 2, \ldots, 10$ the maximum number of states occurring in any intermediate automaton is 21 or less under a normalizing semantics (with no increase from $n = 4$ to 10). These experiments are done based on a ternary semantics (as in Section 3.1; for the usual binary semantics, the maximum number of states will possibly be less (according to results later this in article.) The automata still grow linearly in size of $n$ since the transitions, represented by BDDs in the Mona tool, become more complicated. We leave it to the reader to reflect on why the number of states is limited by a constant. □



Case (1) Case (2)

Figure 3: The two ways for $\phi \lhd \phi'$ to hold.

# 3 WS1S with relativizations

To give a deeper understanding of relativizations, we introduce *nutshell WS1S-R*, a variation on WS1S where relativizations are explicitly marked. Let $\rho$ be a formula that is the *restriction* of variable $P^i$. Existential quantification will now take the form

$$\texttt{ex2}\ P^i\ \texttt{where}\ \rho\colon \phi', \tag{3}$$

where $\phi'$ is the *traditional part* of the quantified formula. The restriction $\rho$ can be an arbitrary formula (as long as the main formula $\phi_0$ remains closed). In general, we denote by $\boldsymbol{\rho}(P^i)$ the formula $\rho$ introduced by the existential quantification of $P_i$. For uniformity, we assume that each $P^i$ is relativized in this way, possibly to the formula $P^i$=$P^i$, which is another way of saying `true`. Our goal in this section is to show how a ternary semantics allows restrictions to bubble up when needed—eliminating the need for normalization at every intermediate step.

To carry out inductive arguments, we define the partial ordering $\unlhd$ among subformulas as the reflexive closure of $\lhd$ defined as: $\phi \lhd \phi'$ if (1) $\phi$ is a proper subformula of $\phi'$ or (2) if there is a formula $\psi = \texttt{ex2}\ P^i\ \texttt{where}\ \boldsymbol{\rho}(P^i)\colon \phi''$ such that $\phi$ is a subformula of $\boldsymbol{\rho}(P^i)$ and $\phi'$ is a subformula of $\phi''$. This definition is illustrated by the parse trees in Figure 3, where the root of the subtree for Case (2) is the node for the formula $\psi$, which has two children, the left one for the restriction and the right one for the traditional part. The partial ordering $\lhd$ is well-founded: a post-order labeling of nodes with numbers $0, 1, \ldots$ produces an ordering, where all children of a node are assigned a number less than that of the parent (making the labeling consistent with case (1)) and where any node that is a left descendant of some node is assigned a number less than a right descendant (making the labeling consistent with case (2)).

When we formulate semantics, variable occurrences in the traditional part of formulas will be subjected to restrictions. More precisely, a variable occurrence of $P^i$ is *directly restrained* if it is inside the traditional part $\phi'$ of the formula (3) introducing $P^i$ (this formula exists as a subformula of the fixed main formula $\phi_0$, which is assumed to be closed). For a subformula $\phi$ of $\phi_0$, the set of *directly restrained variables* DRV($\phi$) is the set of free variables of $\phi$ that have a directly restrained occurrence in $\phi$.

**Proposition 5** For each $\phi$ and each $P \in \mathrm{DRV}(\phi)$, $\boldsymbol{\rho}(P) \lhd \phi$.

**Proof** Let $P^i$ be a variable that has a directly restrained occurrence in $\phi$. Then, $\phi$ is a subformula of $\phi'$, where $\texttt{ex2}\ P^i\ \texttt{where}\ \rho\colon \phi'$ is the formula introducing $P^i$ with $\rho = \boldsymbol{\rho}(P^i)$. Thus, $\rho = \boldsymbol{\rho}(P^i) \lhd \phi$ holds according to (2) of the definition of $\lhd$. □

The treatment of restrained variable occurrences require additional attention. Consider a variable $Q$ that is relativized to the restriction $P = Q$. When a subformula $\phi$ mentioning $Q$ is given meaning, we must include the requirement $P = Q$ in order to avoid later normalizations. But if $P$ itself is relativized to $P = \emptyset$ in some outer existential quantification $\psi$, then the restriction on $P$ itself must be included at some point. The situation is detailed here:

$$\psi = \texttt{ex2}\ P\ \texttt{where}\ P = \emptyset\colon \texttt{ex2}\ Q\ \texttt{where}\ P = Q\colon \underbrace{Q = Q}_{\phi} \tag{4}$$

In this case, we wish to enforce also implied restrictions such as $Q = \emptyset$.

To do so, we define for a formula $\phi$ the set $\mathrm{RV}(\phi)$ of *restrained variables* to be the restrained variables $\mathrm{DRV}(\phi)$ of $\phi$ together with variables $\mathrm{RV}(\boldsymbol{\rho}(P))$ for $P \in \mathrm{DRV}(\phi)$. This definition makes sense, because by virtue of Proposition 5, $\mathrm{RV}(\phi)$ can be defined inductively on formulas ordered according to $\lhd$. For example, in (4) the restrained variables of $\phi$ are $\mathrm{RV}(\phi) = \{P, Q\}$.

For any formula $\phi$, we define the induced restriction $\boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ to be the set of restrictions of restrained variables, that is, the set $\{\boldsymbol{\rho}(P^i) \mid P^i \in \mathrm{RV}(\phi)\}$. In the example, $\boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ is $\{P = \emptyset, P = Q\}$ and the conjunction of these formulas imply $Q = \emptyset$ as desired. By a union-wise extension of $\boldsymbol{\rho}^{\mathrm{RV}}$ to an operator taking a set of formulas as an argument, the expression $\boldsymbol{\rho}^{\mathrm{RV}}(\boldsymbol{\rho}^{\mathrm{RV}}(\phi))$ also makes sense.

**Proposition 6** (a) $\boldsymbol{\rho}(P) \lhd \phi$ holds for $P \in \mathrm{RV}(\phi)$.

(b) $\boldsymbol{\rho}^{\mathrm{RV}}(\phi) \supseteq \boldsymbol{\rho}^{\mathrm{RV}}(\boldsymbol{\rho}^{\mathrm{RV}}(\phi))$

**Proof**

(a) By Proposition 5, $\boldsymbol{\rho}(P) \lhd \phi$ holds for $P \in \mathrm{DRV}(\phi)$. Consider $P' \in \mathrm{RV}(\boldsymbol{\rho}(P))$ for some $P \in \mathrm{DRV}(\phi)$. We may by induction according to $\lhd$ assume that $\boldsymbol{\rho}(P') \lhd \boldsymbol{\rho}(P)$. Thus, by transitivity, $\boldsymbol{\rho}(P') \lhd \phi$ holds.

(b) We need only to prove that $\mathrm{RV}(\phi) \supseteq \mathrm{RV}(\boldsymbol{\rho}(P^i))$ holds for $P^i \in \mathrm{RV}(\phi)$. But this is a direct consequence of the definition of the restrained variables of a formula.

□

## 3.1 From binary to ternary semantics

**Classic semantics** There is an obvious way to define the semantics of Nutshell WS1S-R. We will state them using a meaning function $[\![\cdot]\!]^C$, which given a formula $\phi$ defines a value $[\![\phi]\!]^C \in \{0,1\}$. This notation anticipates multi-valued semantics:

$$\begin{array}{rcl}
[\![\texttt{\textasciitilde}\phi']\!]^C w &=& \neg[\![\phi']\!]^C w \\
[\![\phi' \mathbin{\texttt{\&}} \phi'']\!]^C w &=& [\![\phi']\!]^C w \ \wedge\ [\![\phi'']\!]^C w \\
[\![\texttt{ex2}\ P^i\ \texttt{where}\ \rho\colon \phi']\!]^C w &=& \begin{cases} 1 & \text{if } \exists M : [\![\phi']\!]^C w[P^i \mapsto M] = 1 \\ & \qquad \text{and } [\![\rho]\!]^C w[P^i \mapsto M] = 1 \\ 0 & \text{if } \forall M : [\![\phi']\!]^C w[P^i \mapsto M] = 0 \\ & \qquad \text{or } [\![\rho]\!]^C w[P^i \mapsto M] = 0 \end{cases} \\
[\![P^i\ \texttt{sub}\ P^j]\!]^C w &=& \begin{cases} 1 & \text{if } w \vDash P^i\ \texttt{sub}\ P^j \\ 0 & \text{if } w \nvDash P^i\ \texttt{sub}\ P^j \end{cases}
\end{array}$$

(Again, we have included only one kind of atomic formula; the others also follow their logical semantics.) Of course, we have the following correspondence between WS1S and WS1S-R.

**Proposition 7** Assume that for a formula $\phi$ in Nutshell WS1S-R, we denote by $\hat{\phi}$ the formula obtained by converting $\texttt{ex2}\ P^i\ \texttt{where}\ \rho\colon \phi'$ into $\texttt{ex2}\ P^i : \rho \mathbin{\texttt{\&}} \phi'$. Then, $w \vDash \hat{\phi}$ holds if and only if $[\![\phi]\!]^C w = 1$ holds.

Since the classic semantics behave like expected, we also write $w \vDash \phi$ for WS1S-R formulas that satisfy $[\![\phi]\!]^C w = 1$.

**Conjunctively normalized semantics** Recall from Section 2.3 that a conjunctive semantics solve the problem of spurious behavior of automata-based semantics. We define the semantics formally below, where we use the notation $[\![\boldsymbol{\rho}^{\mathrm{RV}}(P^i\ \texttt{sub}\ P^j)]\!]^N w = 1$ to denote that for each $\rho \in \boldsymbol{\rho}^{\mathrm{RV}}(P^i\ \texttt{sub}\ P^j)$ it holds that $[\![\rho]\!]^N w = 1$.

$$\begin{array}{rcl}
[\![\texttt{\textasciitilde}\phi']\!]^N w &=& \neg([\![\phi']\!]^N w) \ \wedge\ ([\![\boldsymbol{\rho}^{\mathrm{RV}}(\phi')]\!]^N w = 1) \\
[\![\phi' \mathbin{\texttt{\&}} \phi'']\!]^N w &=& [\![\phi']\!]^N w \ \wedge\ [\![\phi'']\!]^N w \\
[\![\texttt{ex2}\ P^i\ \texttt{where}\ \rho\colon \phi']\!]^N w &=& \begin{cases} 1 & \text{if } \exists M : [\![\phi']\!]^N w[P^i \mapsto M] = 1 \\ 0 & \text{if } \forall M : [\![\phi']\!]^N w[P^i \mapsto M] = 0 \end{cases} \\
[\![P^i\ \texttt{sub}\ P^j]\!]^N w &=& \begin{cases} 1 & \text{if } w \vDash P^i\ \texttt{sub}\ P^j \\ & \qquad \text{and } [\![\boldsymbol{\rho}^{\mathrm{RV}}(P^i\ \texttt{sub}\ P^j)]\!]^N w = 1 \\ 0 & \text{if } w \nvDash P^i\ \texttt{sub}\ P^j \\ & \qquad \text{or } [\![\boldsymbol{\rho}^{\mathrm{RV}}(P^i\ \texttt{sub}\ P^j)]\!]^N w = 0 \end{cases}
\end{array}$$

(The definition is a similar for the other atomic formulas.)

Following our goal of using as few normalization operations as possible, we apply the restrictions of free variables to atomic formulas and negations only. For conjunctions, the free variables are the union of the free variables in the contents; so, the explicit normalization can be omitted. For existential quantification, any variable free in the quantified formula `ex2` $P^i$ `where` $\rho : \phi'$ is also free in $\rho$ or in $\phi'$. One problem exists for the variable $P^i$: if the restriction $\rho$ is not satisfiable and the variable $P^i$ does not occur in $\phi'$, then the quantified formula unexpectedly evaluates to 1. Anticipating a similar problem for the ternary semantics, we settle on the following assumption:

For all formulas of the form $\phi$ = `ex2` $P^i$ `where` $\rho : \phi'$ and for all $w$: it holds that

$$w \vDash (\underset{P \in \mathrm{RV}(\phi)}{\texttt{\&}} \boldsymbol{\rho}(P)) \texttt{ => ex2 } P^i : \rho \tag{5}$$

where "`=>`" stands for implication (by an obvious formula transformation). (Alternatively, we might demand that there is an occurrence of $P^i$ in $\rho$ for all existentially quantified formulas.) If we let $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ stand for the property that for all $P^i$ in $\mathrm{RV}(\phi)$ it holds that $w \vDash \rho(P^i)$, then we may state the correctness of the conjunctively normalized semantics as follows.

**Proposition 8** Given assumption (5), $w \vDash \phi$ and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ both hold if and only if $[\![\phi]\!]^N w = 1$.

**Proof** The proof is by induction.

*Case* $\phi = \texttt{~}\phi'$. We observe that $\mathrm{RV}(\phi) = \mathrm{RV}(\phi')$. Then, $[\![\texttt{~}\phi']\!]^N w = 1$ if and only if $[\![\phi']\!]^N w = 0$ and $[\![\boldsymbol{\rho}^{\mathrm{RV}}(\phi)]\!]^N w = 0$ if and only if (by applying induction hypothesis twice) ($w \nvDash \phi'$ or $w \nvDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi)$) and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ if and only if $w \nvDash \phi'$ and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ if and only if $w \vDash \phi$ and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi)$.

*Case* $\phi = \phi'$ `&` $\phi''$. We observe that $\mathrm{RV}(\phi) = \mathrm{RV}(\phi') \cup \mathrm{RV}(\phi'')$ holds and that $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ holds if and only if both $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi')$ and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi'')$ hold. Then, $[\![\phi]\!]^N w = 1$ if and only if $[\![\phi']\!]^N w = 1$ and $[\![\phi'']\!]^N w = 1$ if and only if (by induction) $w \vDash \phi'$ and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi')$ and $w \vDash \phi''$ and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi'')$ if and only if $w \vDash \phi'$ `&` $\phi''$ and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi'$ `&` $\phi'')$.

*Case* $\phi =$ `ex2` $P^i$ `where` $\rho : \phi'$.

We need to consider the two directions separately.

($\Rightarrow$) Assume that $w \vDash \phi$ and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi)$. Then, there is a set $M$ such that $w[P^i \mapsto M] \vDash \phi'$ and $w[P^i \mapsto M] \vDash \rho$. But, since $\mathrm{RV}(\phi)$ does not mention $P^i$, we also have that $w[P^i \mapsto M] \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi)$. Taken together with $w[P^i \mapsto M] \vDash \rho$, we infer that $w[P^i \mapsto M] \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi')$. Now, together with the fact $w[P^i \mapsto M] \vDash \phi'$, we use the induction hypothesis to establish that $[\![\phi']\!]^N w[P^i \mapsto M] = 1$. Consequently, $[\![\phi]\!]^N w = 1$.

($\Leftarrow$) Intuitively, this direction is valid because an occurrence of $P^i$ in $\phi'$ guarantees that $\rho = \boldsymbol{\rho}(P^i)$ holds thanks to the inclusion of restrictions on variable occurrences in atomic formulas; in the absence of such a variable occurrence, the assumption (5) can be used to find an alternative $M$ that makes $\rho$ hold without affecting the truth value of $[\![\phi']\!]^N$.

We proceed more formally. Assume that $[\![\phi]\!]^N w = 1$. Then, there is a $M$ such that $[\![\phi']\!]^N w[P^i \mapsto M] = 1$. By inductive hypothesis, we infer that $w[P^i \mapsto M] \vDash \phi'$ and $w[P^i \mapsto M] \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi')$. From $w[P^i \mapsto M] \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi')$, we infer that $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi)$, since $P^i$ is not a free variable in $\boldsymbol{\rho}^{\mathrm{RV}}(\phi)$. That leaves us with the need to prove that $w \vDash \phi$ holds.

Now, if $P^i \in \mathrm{RV}(\phi')$ holds, then $w[P^i \mapsto M] \vDash \rho$ holds—due to $w[P^i \mapsto M] \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi')$—and $w \vDash \phi$ holds.

On the other hand, if $P^i$ is not a free variable in $\phi'$, then we can use assumption (5) to find an $M$ that makes $w[P^i \mapsto M] \vDash \rho$ hold without impacting the truth of $w[P^i \mapsto M] \vDash \phi'$. By induction hypothesis, it holds that $w[P^i \mapsto M] \vDash \phi'$ and $w[P^i \mapsto M] \vDash \rho$. Thus, we conclude that $w \vDash \phi$.

*Case* $[\![P^i \,\texttt{sub}\, P^j]\!]^N$. $[\![P^i \,\texttt{sub}\, P^j]\!]^N w = 1$ if and only if (by definition) $w \vDash P^i \,\texttt{sub}\, P^j$ and $[\![\boldsymbol{\rho}^{\mathrm{RV}}(P^i \,\texttt{sub}\, P^j)]\!]^N w = 1$ if and only if (by induction) $w \vDash P^i \,\texttt{sub}\, P^j$ and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(P^i \,\texttt{sub}\, P^j)$ and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\boldsymbol{\rho}^{\mathrm{RV}}(P^i \,\texttt{sub}\, P^j))$ if and only if (by Proposition 6(b)) $w \vDash P^i \,\texttt{sub}\, P^j$ and $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(P^i \,\texttt{sub}\, P^j)$. □

**The ternary semantics** Let $\mathbb{B}^\perp = \mathbb{B} \cup \{\perp\}$ be the *extended Boolean domain*. We use $\perp$ to denote a "don't care" situation, one where not all the restrictions hold. Boolean operators $\wedge^3$ and $\neg^3$ are defined on this domain as for the usual case with the added rule that if any argument is $\perp$, then the result is $\perp$.

$$\begin{array}{rcl}
[\![\texttt{\~{}}\phi']\!]^3 w &=& \neg^3 [\![\phi']\!]^3 w \\
[\![\phi' \,\texttt{\&}\, \phi'']\!]^3 w &=& [\![\phi']\!]^3 w \;\wedge^3\; [\![\phi'']\!]^3 w \\
[\![\texttt{ex2}\; P^i \;\texttt{where}\; \rho \texttt{:}\, \phi']\!]^3 w &=& \begin{cases} 1 & \text{if } \exists M : [\![\phi']\!]^3 w[P^i \mapsto M] = 1 \\ 0 & \text{if } \forall M : [\![\phi']\!]^3 w[P^i \mapsto M] \neq 1 \text{ and} \\ & \qquad \exists M : [\![\phi']\!]^3 w[P^i \mapsto M] = 0 \\ \perp & \text{if } \forall M : [\![\phi']\!]^3 w[P^i \mapsto M] = \perp \end{cases} \\
[\![P^i \,\texttt{sub}\, P^j]\!]^3 w &=& \begin{cases} 1 & \text{if } w \vDash P^i \,\texttt{sub}\, P^j \\ & \text{and } [\![\boldsymbol{\rho}^{\mathrm{RV}}(P^i \,\texttt{sub}\, P^j)]\!]^3 w = 1 \\ 0 & \text{if } w \nvDash P^i \,\texttt{sub}\, P^j \\ & \text{and } [\![\boldsymbol{\rho}^{\mathrm{RV}}(P^i \,\texttt{sub}\, P^j)]\!]^3 w = 1 \\ \perp & \text{if } [\![\boldsymbol{\rho}^{\mathrm{RV}}(P^i \,\texttt{sub}\, P^j)]\!]^3 w \neq 1 \end{cases}
\end{array}$$

(The definition is similar for the other atomic formulas.)

The case of existential quantification above reflects the intuition that if a witness $M$ exists that makes $\phi'$ true, then $\phi$ is true; moreover, if there is an $M$ that makes $\phi'$ false, and no $M$ makes $\phi'$ true, then $\phi$ is false; and, finally, if all witnesses $M$ make $\phi$ not satisfy the restrictions $\boldsymbol{\rho}^{\mathrm{RV}}(\phi')$, then $w$ does not satisfy $\boldsymbol{\rho}^{\mathrm{RV}}(\phi)$.

Something seems to be missing in these semantics: the enforcement of a relativization. The proposition below shows that the relativization bubbles up automatically if needed:

**Proposition 9** Given assumption (5), the following holds.

(a) $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi) \;\Leftrightarrow\; [\![\phi]\!]^3 w \neq \bot$.

(b) $w \vDash \phi \mathbin{\&} \boldsymbol{\rho}^{\mathrm{RV}}(\phi) \;\Leftrightarrow\; [\![\phi]\!]^3 w = 1$

(c) $w \vDash \tilde{}\phi \mathbin{\&} \boldsymbol{\rho}^{\mathrm{RV}}(\phi) \;\Leftrightarrow\; [\![\phi]\!]^3 w = 0$

**Proof** (Idea) The proof is similar to that of Proposition 8 and is omitted. □

Part (a) states that the truth of all syntactic restrictions $\boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ is equivalent to $[\![\phi]\!]^3 w$ not being $\bot$.

## 3.2 Automata-theoretic realization of the ternary semantics

The decision procedure of Section 2.2 can be modified to reflect the ternary semantics.

First, we classify states as either 1, 0, or $\bot$ states; that is, we replace $Q^F$ of an automaton $A$, as defined in Section 2.2, with a valuation or labeling function $\lambda : Q \to \mathbb{B}^{\bot}$. For a word $u$, the valuation $\lambda(\textsc{last}A(u))$ of the last state of the run over $u$ is the truth value *calculated* by $A$ over $u$. We denote this value $A(u)$, and we call $A$ a *ternary automaton*. Of course, the logic-automata connection can now be expressed: $A_\phi(u) = [\![\phi]\!]^3 u$. Ternary automata can be minimized in the same way conventional deterministic automata are minimized with the difference that the initial state partition is given by the at most three maximal sets on which $\lambda$ is constant.

Second, we modify the automata-theoretic constructions. For the case of conjunction, we simply change the way product states are labeled; such a state is 1 if both component states are one, it is 0 if both components are 0 or if one is 0 and the other 1, and it is $\bot$ if one of the components is $\bot$. Negation is handled by simply switching 1 states in 0 states and vice versa.

Third, we must generalize the projection operators and futurization to the ternary domain. We define $\textsc{proj}^{3,i}$, $\textsc{bproj}^{3,i}$, and $\textsc{fut}^{3,i}$:

$$\textsc{proj}^{3,i}A(u) \quad = \quad \begin{cases} 1 & \text{if } \exists M : A(u[P^i \mapsto M]) = 1 \\ 0 & \text{if } \exists M : A(u[P^i \mapsto M]) = 0 \\ & \qquad \text{and } \forall M : A(u[P^i \mapsto M]) \neq 1 \\ \bot & \text{if } \forall M : A(u[P^i \mapsto M]) = \bot \end{cases}$$

$$\textsc{bproj}^{3,i}A(u) \quad = \quad \begin{cases} 1 & \text{if } \exists M \text{ with } \max M < |u| : A(u[P^i \mapsto M]) = 1 \\ 0 & \text{if } \exists M \text{ with } \max M < |u| : A(u[P^i \mapsto M]) = 0 \\ & \qquad \text{and } \forall M \text{ with } \max M < |u| : A(u[P^i \mapsto M]) \neq 1 \\ \bot & \text{if } \forall M \text{ with } \max M < |u| : A(u[P^i \mapsto M]) = \bot \end{cases}$$

$$\textsc{fut}^{3,i}A(u) \quad = \quad \begin{cases} 1 & \text{if } \exists v \in {\Sigma_0^i}^* : A(u \cdot v) = 1 \\ 0 & \text{if } \exists v \in {\Sigma_0^i}^* : A(u \cdot v) = 0 \\ & \qquad \text{and } \forall v \in {\Sigma_0^i}^* : A(u \cdot v) \neq 1 \\ \bot & \text{if } \forall v \in {\Sigma_0^i}^* : A(u \cdot v) = \bot \end{cases}$$

**Proposition 10**

$$\textsc{proj}^{3,i}A = \textsc{bproj}^{3,i}(\textsc{fut}^{3,i}A)$$

**Proof** Omitted. □

Fourth, we explain how these operators are implemented as automata algorithms. The construction of $\textsc{fut}^{3,i}A$ is quite obvious: we label any state $1$ for which a path along a null-but-$i$ extension to a state labeled $1$ exists; among the remaining states, we label those $0$ for which a null-but-$i$ extension exists that leads to a $0$-labeled state in $A$; and finally, those not yet labeled retain their $\perp$-label. In a subset implementation of $\textsc{bproj}^{3,i}$, a subset is labeled $1$ if it contains a state labeled $1$; when it contains no such state, it is labeled $0$ if it contains a state labeled $0$; and, when all states in the subset are labeled $\perp$, the subset state is also labeled $\perp$.

We call the resulting algorithm the *ternary decision procedure*.

# 4 Ternary valuations for restricted languages

All languages considered will be regular and over the alphabet $\Sigma = \mathbb{B}^k$. For a language $L$, the *canonical right-congruence* $\sim_L$ is defined as $u \sim_L v$ if and only if $\forall w : u \cdot w \in L \Leftrightarrow v \cdot w \in L$, where $u, v, w \in \Sigma^*$. The set of congruence classes is denoted $\Sigma^*/{\sim_L}$. This set can be regarded as the state set of a canonical, finite-state automaton recognizing $L$.

Consider languages $L$, sometimes called the *property*, and $R$, assumed non-empty, called a *restriction*. Thus, $L_\phi$ and $L_{\boldsymbol{\rho}^{\mathrm{RV}}(\phi)}$, for $\boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ defined in Section 3.1, constitute are such a pair for any subformula $\phi$ of $\phi_0$. The *conjunctively normalized representation* is $L' = L \cap R$, and the *conjunctively normalized congruence* is $\sim_{L \cap R}$.

Note that Proposition 8 tells us that $[\![\phi]\!]^N$ is just the characteristic function for the set $L \cap R$ with $L = L_\phi$ and $R = L_{\boldsymbol{\rho}^{\mathrm{RV}}(\phi)}$.

For a string $u$, *an accepting extension* $v$ is a string such that $u \cdot v \in L \cap R$ and a *rejecting extension* $v$ is such that $u \cdot v$ in $\overline{L} \cap R$ . The *ternary valuation* determined by $L$ and $R$ is a function $\chi_{L,R}(u)$, defined to be $1$ if $u \in L \cap R$, $0$ if $u \in \overline{L} \cap R$, and $\perp$ if $u \notin R$.

Note that Proposition 9 tells us that that $[\![\phi]\!]^3 = \chi_{L,R}$ with $L$ and $R$ identified as above.

The *ternary congruence* $\sim_{L,R}$ is then defined by $u \sim_{L,R} v$ if for all $w$ it holds that $\chi_{L,R}(u \cdot w) = \chi_{L,R}(v \cdot w)$. The equivalence classes of a ternary congruence can be viewed as the states of a ternary automaton $A$. More precisely, $A$ is $(\Sigma, Q, q_0, \rightarrow, \chi^F)$, where $Q = \Sigma^*/\sim_{L,R}$ along with $q^0 = \{u \mid u \sim_{L,R} \epsilon\}$ and $\rightarrow$ are defined as usual; moreover, $\lambda$ is defined so that $A(u) = \chi_{L,R}(u)$ holds—this makes sense: the valuation function $\lambda$ is constant on all strings reaching the same state. We say that $A$ *recognizes* $\chi$. Often, we will call $A$ the $\chi_{L,R}$-automaton. Note that this automaton respects $\sim_R$: for any state, there is a uniquely determined equivalence class of $\sim_R$ that all strings reaching the state belongs to. This is just another way of saying that $\sim_{L,R}$ refines $\sim_R$. Obviously, $\sim_{L,R}$ also refines $\sim_{L \cap R}$.

## 4.1 Relating the conjunctive and ternary congruences

A *thin language* $R$ is a non-empty set of strings such that

$$\forall u, v : u \not\sim_R v \ \Rightarrow \forall w : u \cdot w \notin R \vee v \cdot w \notin R \tag{6}$$

In particular, it can be seen that the canonical automaton for $R$ has exactly one accepting state: just make $u$ be a string that reaches one accepting state, $v$ a string that reaches a different accepting state, and $w$ the empty string.

**Proposition 11**

(a) The *first-order restriction* $R_{\text{singleton}(i)} = \{u \in (\mathbb{B}^k)^* \mid P^i$-track $i$ contains exactly one occurrence of a $1\}$ is thin.

(b) The $\$$-*restriction*

$$\begin{aligned} R_{\text{\$-restrict}(i)} = \{u \in \mathbb{B}^k \mid &\text{the occurrences of } 1 \text{ in } P^i\text{-track are all in positions}\\ &\text{no greater than that of the first occurrence of a } 1\\ &\text{in track } \$\} \ \cap \ R_{\text{singleton(\$)}} \end{aligned}$$

is thin.

(c) If $R$ and $R'$ are thin and $R \cap R' \neq \emptyset$, then $R \cap R'$ is thin.

(d) Let $R$ be thin, and let $L$ be any language. If $u \sim_{L \cap R} v$ and $u$ has an accepting extension, then $u \sim_{L,R} v$.

(e) If $u$ and $v$ both have no accepting extensions, then $u \sim_R v \Leftrightarrow u \sim_{L,R} v$.

(f) Assume that $R$ is thin. Then, $\Sigma^*/\sim_{L,R}$ is a union of equivalence classes of $\Sigma^*/\sim_{L\cap R}$ and $\Sigma^*/\sim_R$. More precisely, either $\Sigma^*/\sim_{L,R} = \Sigma^*/\sim_{L\cap R}$ holds or

$$\Sigma^*/\sim_{L,R} = (\Sigma^*/\sim_{L\cap R} \setminus \{S_{L\cap R}\}) \cup \mathcal{S}$$

holds, where $S_{L\cap R} = \{u \mid \forall v : u \cdot v \notin L \cap R\}$ and $\mathcal{S} = \{S \in \Sigma^*/\sim_R \mid \exists u \in S : \forall v : u \cdot v \notin L \cap R\}$.

(g) $|\Sigma^*/\sim_{L\cap R}| \leq |\Sigma^*/\sim_{L,R}| \leq |\Sigma^*/\sim_{L\cap R}| + |\Sigma^*/\sim_R|$ -1.

**Proof**

(a) We note that there are three equivalence classes corresponding to this language: strings containing zero occurrences, exactly one occurrence, or two or more occurrences of a 1 in the $P^i$-track. Consider $u$ and $v$ such that $u \not\sim_R v$. Let $w$ be any string, and assume that $u \cdot w \in R$. Then $u$ contains at most one occurrence of a 1 in the $P^i$-track. We must prove that $v \cdot w \notin R$. *Case* (1): $u$ contains no 1s in the $P^i$-track. Then by assumption that $u \cdot w \in R$, $w$ contains exactly one occurrence of 1 in the $P^i$-track. But, by assumption that $u \not\sim_R v$, $v$ contains either one 1 or two or more 1s. In all cases, $v \cdot w \notin R$. *Case* (2): $u$ contains exactly one occurrence of a 1. Then $w$ contains no occurrences and $v$ contains zero or two or more occurrences. Thus, again it will hold that $v \cdot w \notin R$.

(b) The proof is similar to the previous case. Also, it can be seen that the intersection with the singleton language is necessary for thinness.

(c) Consider $u$ and $v$ such that $u \not\sim_{R\cap R'} v$. Then, either $u \not\sim_R v$ or $u \not\sim_{R'} v$. Assume the former and that $u \cdot w \in R \cap R'$. Then, $v \cdot w \notin R$ thanks to $R$ being thin, and thus $u \cdot w \notin R \cap R'$.

(d) Let $w$ be the accepting extension of $u$. From $u \sim_{L\cap R} v$, we infer that $v \cdot w \in L \cap R$. Assume for a contradiction that $u \not\sim_R v$. Combined with the thinness of $R$, and the fact that $u \cdot w \in R$, we would conclude that $v \cdot w \notin R$. This contradicts that $v \cdot w \in L \cap R$. Thus, it must hold that $u \sim_R v$. It follows from elementary considerations that $u \sim_{L\cap R} v$ and $u \sim_R v$ implies that $u \sim_{L,R} v$.

(e) Assume that $u$ and $v$ both have no accepting extensions and that $u \sim_R v$. Let $w$ be any string. Then, $u \cdot w$ and $v \cdot w$ are both in $R$ or none is. In the second case, $\chi_{L,R}(u \cdot w) = \chi_{L,R}(v \cdot w) = \bot$. In the first case, it follows from the assumption that $u$ and $v$ both have no accepting extensions that $\chi_{L,R}(u \cdot w) = \chi_{L,R}(v \cdot w) = 0$.

(f) If $S_{L\cap R}$ is empty, then all strings $u$ have an accepting extension. Then it follows by (d) that $\Sigma^*/\sim_{L\cap R}$ and $\Sigma^*/\sim_{L,R}$ are identical. Otherwise, when $S_{L\cap R}$ is nonempty, it follows again by 4. that equivalence classes of $\sim_{L\cap R}$ and $\sim_{L,R}$ for strings that have accepting extensions coincide. Strings that do not have such extensions fall into equivalence classes of $\sim_R$ according to (e).

(g) This follows from (f).

□

From this proposition, it follows easily that for any $\phi$ the regular language representing the conjunction of the restrictions in $\boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ is thin if variables are subjected to only first-order relativizations or to \$-relativizations (or to both). The proposition also tells us that $\Sigma^*/\sim_{L,R}$ is pieced together from $\Sigma^*/\sim_{L\cap R}$ plus a subset of $\Sigma^*/\sim_R$.

## 4.2 The ternary decision procedure compared

**Theorem 1** Assume that all restrictions are thin languages and that their automata have at most $N$ states. Then, the size of each intermediate, minimized automaton (representing some subformula) in the ternary decision procedure is the same, to within an additive constant of $N - 1$, as the size of corresponding automaton under the conjunctive semantics.

**Proof** This follows from (g) of the previous proposition. □

This result is the justification for the practical use of the ternary semantics since usually the number of first-order variables in simultaneous use is quite small. But note that the size of the additive constant is exponential in the number of free first-order variables. Also, thanks to Proposition 11(f), we see that the automata of the ternary approach are, apart from the $\Sigma^*/\sim_R$ parts, the same as those that occur when the automaton of every subformula is normalized conjunctively.

# 5 The sexpartite approach

We show next how to get rid of the assumption in Theorem 1 that the automata of the restrictions are bounded in size. We do so by re-introducing a normalization step at the usual place of the relativization where the variable is introduced by a quantifier. This will allow us to prune away states from the automaton representing the ternary semantics. We want to get rid of all states following a state if all these states have the same membership status with respect to $L$ whenever they are in $R$.

## 5.1 Interesting strings, approximations, and sexpartification

To proceed more rigorously, we define a string $u$ to be *interesting* (for $L$ and $R$) if it has (a) some accepting extension and (b) some rejecting extension. Also, a *"don't care"* extension is one that makes a string fall outside $R$. Note that all prefixes of an interesting string are also interesting. In other words, an uninteresting string cannot be extended so as to become interesting. Note also that if $u \sim_{L\cap R} v$ and $u$ is interesting then $v$ is also interesting. Define the Boolean valuation $\iota_{L,R}(u)$ so that it denotes whether a string $u$ is interesting. For an uninteresting string, let $\textsc{cut}(u)$ be the shortest uninteresting prefix of $u$. The *approximation* $\alpha_{L,R}(u)$ of an uninteresting $u$ is defined by

$$\alpha(u) = \begin{cases} 1 & \text{if } \textsc{cut}(u) \text{ has an accepting extension} \\ 0 & \text{if } \textsc{cut}(u) \text{ has a rejecting extension} \\ \perp & \text{if all extensions of } \textsc{cut}(u) \text{ are "don't care"} \end{cases} \tag{7}$$

(These three cases are clearly mutually exclusive.)

For $u \neq \epsilon$, we define $u^-$ to be the prefix $v$ such that $v \cdot a = u$ for some $a \in \Sigma$.

We note that if $\textsc{cut}(u) \neq \epsilon$, then $\textsc{cut}(u)^-$ is interesting. Moreover, if $\textsc{cut}(u) = \epsilon$, then $\alpha(u) \in \{0, 1\}$, since we assume that $R \neq \emptyset$.

When $u$ is interesting, we desire to make $\alpha$ exact so we define $\alpha_{L,R}(u)$ to be $\chi_{L,R}(u)$. We make the *sexpartite valuation* $\chi^6_{L,R}$ be the interest status together with the approximation: $\chi^6_{L,R}(u) = (\iota(u), \alpha(u))$. The valuation $\chi^6_{L,R}$ is also called the *sexpartification* of $\chi_{L,R}$ since it can be defined from $\chi_{L,R}$ alone. The *canonical sexpartite congruence* $\sim^6_{L,R}$ is defined from the valuation as for the ternary case.

### 5.1.1 The exactness property

When $L$ and $R$ are clear from the context, we often omit them as subscripts. The uninteresting equivalence classes of $\Sigma^*/\!\sim^6$ are just that: if $u$ is uninteresting, with $(\iota(u), \alpha(u)) = (0, X)$, then for any extension $v$, $(\iota(u), \alpha(v)) = (0, X)$. Thus, the only transition in $\Sigma^*/\sim^6$ from an uninteresting equivalence class is to itself. Moreover, there are at most three such classes.

The approximation $\alpha(u)$ satisfies the following properties: if $u$ is interesting or $\chi(u)$ is not $\perp$, then $\alpha(u) = \chi(u)$ holds; otherwise, if $u$ is uninteresting and $\chi(u)$ is $\perp$, then $\alpha(u)$ may take on any value in $\{0, 1, \perp\}$. The fact that $\alpha(u) = \chi(u)$ usually holds—with the only possible exception being that $\chi(u)$ is $\perp$ for an uninteresting $u$—is called the *exactness property*.

We note that if $u$ is uninteresting and $\alpha(u)$ is not $\perp$, then it is possible, and somewhat counterintuitive, that for all $v$ the value $\chi(u \cdot v)$ is $\perp$. When $\alpha(u) \neq \perp$ holds, it is only for $u = \text{CUT}(u)$ that an extension $v$ is guaranteed to exist such that $\chi(u \cdot v) \neq \perp$ holds.

### 5.1.2 Recovering $\chi^6$ and $\chi$ from $\alpha$

In the following, we will be concerned only with the approximation valuation $\alpha$ because of the following property:

**Proposition 12** Let $\chi^6 = (\iota, \alpha)$ be a sexpartite valuation. Then the following property holds:

$$\iota(u) = 1 \quad \Leftrightarrow \; \exists v^0, v^1 : \alpha(u \cdot v^0) = 0 \;\wedge\; \alpha(u \cdot v^1) = 1$$

Thus, the approximation $\alpha$ alone carries all the information of the sexpartite valuation $\chi^6$.

We may even recuperate $\chi$ from $\alpha$ by using the operator $\times_\perp$ on valuations that for a ternary $\chi$ and a binary $\rho$ behaves according to

$$(\chi \times_\perp \rho)(u) \quad = \quad \begin{cases} \chi(u) & \text{if } \rho(u) = 1 \\ \perp & \text{if } \rho(u) = 0 \end{cases} \tag{8}$$

We write $\chi \times_\perp R$ to denote $\chi \times_\perp \rho$, where $\rho$ is the characteristic function for $R$. Then, it is easy to see that $\chi$ can be recovered from $\alpha$:

$$\chi_{L,R} = \alpha_{L,R} \times_\perp R$$

## 5.2 Sexpartification algorithm

Let us consider some properties of $\sim_{L,R}$ through a study of the automaton $A$ recognizing it. First, we note that the notions of “interesting,” “accepting extension,” “rejecting extension,” and “don’t care” extension apply to states of $A$ as well. So, we can partition the states of $A$ into at most four sets, namely the set of interesting states $I$ and three sets of non-interesting states. The non-interesting sets are: $N^1$, which consists of states allowing some accepting extension (but no rejecting extension); $N^0$, which consists of states allowing some rejecting extension (but no accepting extension), and $N^\perp$, which consists of states allowing only don’t care extensions. Some of these sets may be empty; for simplicity, we assume that none is.

Second, we infer various properties from the definition of $\chi_{L,R}$. (1) $N^\perp$ is a singleton set (assuming that $A$ is minimal). (2) The only transitions among these sets of states are as follows: $I \to N^0$, $I \to N^1$, $I \to N^\perp$, $N^0 \to N^\perp$, and $N^1 \to N^\perp$, where

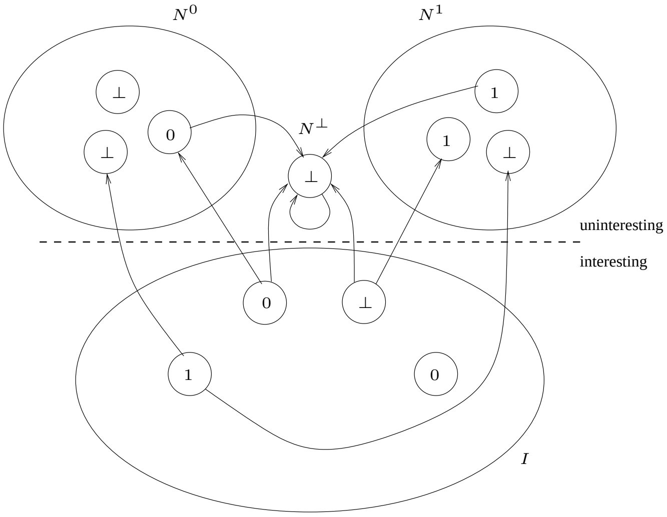


Figure 4: The transition structure of $\Sigma^*/\!\sim_{L,R}$.

$M \to N$ means that there is some transition from $M$ to $N$. Thus, Figure 4 illustrates the structure of $\Sigma^*/\sim_{L,R}$.

With these observations, we can present an algorithm called *sexpartification* that from $A$ recognizing $\chi_{L,R}$ calculates an automaton $\textsc{sexp}(A)$ recognizing $\alpha_{L,R}$.

First, we construct from $A$ a *collapsed* automaton $A'$ by collapsing states in the set $N^0$ to one state, which for simplicity we also call $N^0$. This state is defined to have a transition to itself on all letters; in particular, the outgoing transitions, all of which are to $N^\perp$, are removed. (Of course, the removal of transitions is at odds with the traditional ways of shrinking automata.) We do the same for $N^1$. If there are no incoming transitions from $I$ to $N^\perp$, we remove $N^\perp$. Each of the at most three states of the form $N^Z$ is labeled $Z$. Each interesting state, i.e. each state in $I$, keeps its label. This completes the construction of $A'$.

Second, we minimize $A'$ to obtain $\textsc{sexp}(A)$. The shape of this automaton is depicted in Figure 5.

**Proposition 13** $\textsc{sexp}(A)$ calculated through sexpartification from an automaton $A$ recognizing $\chi_{L,R}$ is the minimal automaton recognizing $\alpha_{L,R}$.

**Proof** We consider the collapsed automaton $A'$ since the second step does not affect what is recognized. We consider only the case where the empty string is interesting. (Other cases are simpler.) It follows that the initial state is interesting. From the definition of $\chi^6_{L,R}$, it can be seen that the state reached in $A'$ over a word $u$ correctly identifies $\alpha_{L,R}(u)$ as long as the run stays within $I$. When a letter $a$ is such that the run of $A'$ over $u \cdot a$ leaves $I$ on the transition on $a$, $u \cdot a$ is uninteresting, and it can be seen from the definition of $\chi^6_{L,R}$ that $\textsc{cut}(u \cdot a) = u \cdot a$. Moreover, the $N^X$ state reached is such that $X$ is the value of $\alpha_{L,R}(u \cdot a)$. Finally, we have already argued that the equivalence classes $N^X$ allow no outgoing transitions except to themselves. Thus,

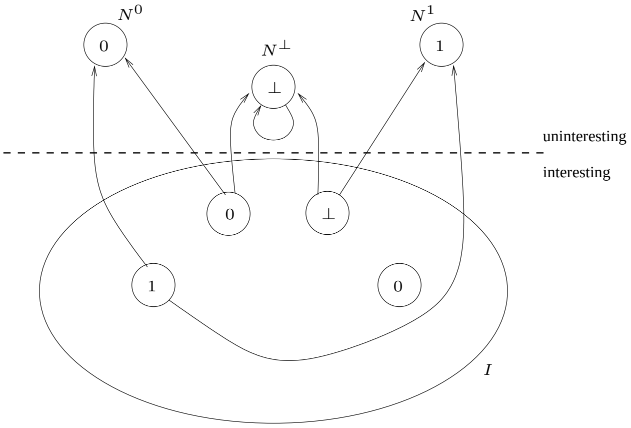


Figure 5: The transition structure of $\Sigma^*/\sim^6_{L,R}$.

$A'$ so constructed recognizes $\alpha_{L,R}$. □

We state a set of necessary and sufficient conditions for establishing that an automaton $A$ is a sufficiently good approximation to $\chi_{L,R}$ for establishing that its sexpartification is that of $\alpha_{L,R}$:

**Proposition 14** For establishing $\text{SEXP}(A) = \alpha_{L,R}$, it is necessary and sufficient that (i) for any interesting $u$, $\chi_{L,R}(u) = A(u)$ holds and (ii) for any minimal uninteresting $u$, the following three conditions obtain

(N1) If $\text{LAST}\alpha_{L,R}(u) = N^1$, then there is an accepting extension for $A$ of $u$ and no rejecting extensions.

(N0) If $\text{LAST}\alpha_{L,R}(u) = N^0$, then there is a rejecting extension for $A$ and no accepting extensions.

(N⊥) If $\text{LAST}\alpha_{L,R}(u) = N^\perp$, then there are but don't care extensions for $A$.

**Proof** "⇐" Condition (i) makes $A$ behave just like $\chi_{L,R}$ for interesting states and conditions (N1), (N0), and (N⊥) ensure that uninteresting states are correctly identified.

"⇒" (i) if $u$ is interesting, then $\text{SEXP}(A)(u) = \alpha_{L,R}(u) = \chi_{L,R}(u)$, but $u$ will also be interesting relative to the state $\text{LAST}A(u)$, and sexpartification will ensure that $A(u) = \text{SEXP}(A)$. (ii) (N1) Assume that $\text{LAST}\alpha_{L,R}(u) = N^1$ and that $u$ is minimal. Then there is a $v$ such that $\chi_{L,R}(u \cdot v) = 1$. But $\text{SEXP}(A)(u \cdot v) = \alpha_{L,R}(u \cdot v) = 1$, so $A(u \cdot v)$ is also 1. A rejecting extension $u \cdot v$, i.e. one such that $A(u \cdot v) = 0$ would also be impossible sense $\text{LAST}(\text{SEXP}(A))(u)$ is also the $N^1$ state. (N0) is similar to (N1). (N⊥) Since $\text{LAST}A(u)$ is $N^\perp$, there can be no $v$ such that $A(u \cdot v) \neq \perp$. □

An automaton $A$ satisfying the conditions of the proposition is called a *sexpartite automaton for* $L$ *and* $R$.

**Proposition 15** (a) $|\Sigma^*/\sim^6_{L,R}| \leq |\Sigma^*/\sim_{L,R}|$.

(b) $\sim_{L,R}$ may not refine $\sim^6_{L,R}$.

(c) For interesting strings, $\sim_{L,R}$ *locally refines* $\sim^6_{L,R}$, that is, if $u$ is interesting, then $u \sim_{L,R} v$ implies that $u \sim^6_{L,R} v$.

**Proof** Items (a) and (c) follow from the sexpartification construction. The property (b) that $\sim^6_{L,R}$ is not coarser than $\sim_{L,R}$ can be seen by construction of a $\chi$ corresponding to some $L$ and $R$ such that there is a string $u \cdot v$ with $u$ bringing the $\chi$-automaton into a $N^0$-state and with $v$ from there driving the automaton into a $N^\perp$-state (which will be unique). Also, there must be a string $w$ bringing the automaton into the $N^\perp$-state without passing through any $N^0$-state. Then, $u \cdot v \sim_{L,R} w$ holds, but $u \cdot v \sim^6_{L,R} w$ does not hold, since after sexpartification $u \cdot v$ leads to the $N^0$-state, but $w$ leads to the $N^\perp$-state. □

We shall later (in Section 8.2) introduce a property of $R$ that strengthens (c) such that $\sim_{L,R}$ and $\sim^6_{L,R}$ locally coincide.

### 5.3 An example

Consider the alphabet $\Sigma = \{a, b\}$, $R_c = \{w \mid |w| = 0 \mod 3\}$, and $L_c = a \cdot a \cdot \Sigma^* \cup b \cdot a \cdot a \cdot \Sigma^*$. Since $R_c$ is the set of strings whose length is 0 modulo 3, it follows that $R_c$ is thin. The automaton recognizing $L_c$ is shown in Figure 6, along with automata for $\chi_{L_C,R_C}$ and $\chi^6_{L_C,R_C}$. In figure, we have used diamond shapes for the states that are labeled $\perp$. All states in the depiction of $\chi^6_{L_C,R_C}$ are interesting, except for the states $N^0$ and $N^1$.

This example is constructed to show that after sexpartification, interesting states may no longer respect $\sim_R$, even when the restriction is a thin language. (We have already noted that the states of $\chi$ respect $\sim_R$, i.e. that $\sim_{L,R}$ refines $R$.) To see this, consider the interesting $\chi$-automaton states $q'$ and $q''$, which correspond to different $\sim_{R_c}$ classes. They are fused as a result of sexpartification as can be seen in the depiction of the $\chi^6_{L_C,R_C}$-automaton. Incidentally, this automaton is isomorphic to the original one, except for the labeling of states. The example thus shows how sexpartification may, in lucky cases, completely undo the complication of a restriction that is conjoined to language.

# 6 Conjunctions of sexpartite representations

In order to normalize restrictions to $R' \cap R''$, it is necessary to make further assumptions. We say that restrictions $R'$ and $R''$ are *compatible* if the following holds:

$$\forall u : (\exists v' : u \cdot v' \in R') \wedge (\exists v'' : u \cdot v'' \in R'') \Rightarrow (\exists v : u \cdot v \in R' \cap R'')$$

**Proposition 16** Any two restrictions $R'$ and $R''$ that are intersections of sets of the form $R_{\text{singleton}(i)}$ and $R_{\$-\text{restrict}(i)}$ of Proposition 11 are compatible.

**Proof** Left to the reader. □

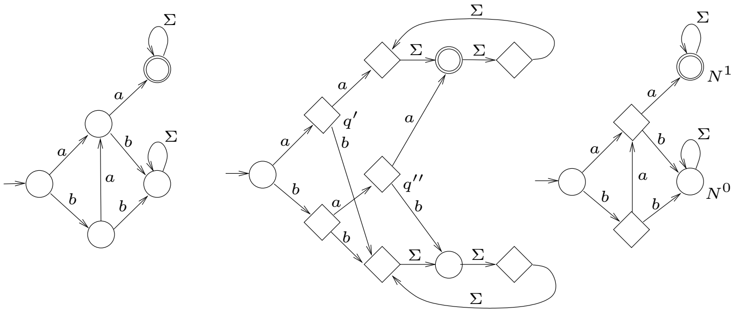


Figure 6: Automata for $L_C$, $\chi_{L_C,R_C}$, and $\chi^6_{L_C,R_C}$.

Because $\chi$ can be recovered from $\alpha$ (see 5.1.2), we may easily establish that $\alpha_{L,R\cap R'}$ $= \text{SEXP}(\alpha_{L,R} \times_\perp R \times_\perp R')$. But it is obvious to ask whether the product $\times_\perp R$ is really necessary; intuitively, it appears to be the case that the information is already built into $\alpha_{L,R}$. Although we will not need the result, we show that the product with the characteristic function of $R$ in this identity is indeed unnecessary when $R$ and $R'$ are compatible:

**Proposition 17** If $R$ and $R'$ are compatible, then

$$\alpha_{L,R\cap R'} = \text{SEXP}(\alpha_{L,R} \times_\perp R').$$

More generally, if we replace $\alpha_{L,R}$ with any sexpartite automaton $B$ for $L$ and $R$, then $\alpha_{L,R\cap R'} = \text{SEXP}(B \times_\perp R')$.

**Proof** We use the method of Proposition 14, where $R$ of the proposition is $R \cap R'$ and $A$ is $\alpha_{L,R} \times_\perp R'$. (For the more general formulation, the argument is similar, except that $\alpha_{L,R}$ is replaced by $B$. The exactness property and the notion of interesting remain the same.)
*Case* (i): The string $u$ is interesting for $L$ and $R\cap R'$. We must show that $\chi_{L,R\cap R'}(u) = A(u)$. We note that $u$ is also interesting for $L$ and $R$; so, by the exactness property, $\chi_{L,R}(u) = \alpha_{L,R}(u)$ holds. By definition of $\chi_{L,R\cap R'}$, we have $\chi_{L,R\cap R'}(u) = (\chi_{L,R} \times_\perp R')(u) = (\alpha_{L,R} \times_\perp R')(u)$. Thus, $\chi_{L,R\cap R'}(u) = A(u)$ holds.
*Case* (ii) We now assume that $u$ is uninteresting and that $u^-$ is interesting.
*Subcase* (N1) Assume that $\text{LAST}\alpha_{L,R\cap R'}(u) = N^1$.

First, we prove that there is an accepting extension in $A$ from $\text{LAST}A(u)$. Let $v$ be such that $\chi_{L,R\cap R'}(u \cdot v) = 1$ (such a $v$ exists because $\text{LAST}\alpha(u) = N^1$ and $u$ is a minimal uninteresting string). Thus, $u \cdot v \in R \cap R'$ holds. In particular, $u \cdot v \in R$ holds, and by the exactness property, $\chi_{L,R}(u \cdot v) = \alpha_{L,R}(u \cdot v)$=1 holds, whence we infer that $(\alpha_{L,R} \times_\perp R')(u \cdot v) = 1$. That is, there is an accepting extension in $A$.

Second, we prove that there is no rejecting extension in $A$ from $\text{LAST}(u)$. To do so, we assume for contradiction that there is a $v$ such that $(\alpha_{L,R} \times_\perp R')(u \cdot v) = 0$.

Now, if $u{\cdot}v$ is interesting for $L$ and $R$, then $\chi_{L,R}(u{\cdot}v) = \alpha_{L,R}(u{\cdot}v) = 0$ (because of the exactness property and because we just assumed that $(\alpha_{L,R} \times_\perp R')(u \cdot v) = 0$

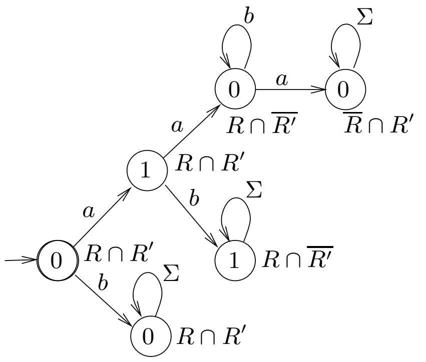
holds). Moreover, we know that $u \cdot v \in R'$. Thus, we also have $\chi_{L,R\cap R'}(u \cdot v) = 0$, but that contradicts the assumption that $\textsc{last}\alpha_{L,R\cap R'}(u) = N^1$.

So, we may assume that $u \cdot v$ is uninteresting for $L$ and $R$. But, since $u$ is interesting for $L$ and $R$ and $u \cdot v$ is not, there is a $\hat{u}$ such that $u \cdot \hat{u}$ is uninteresting, $u \cdot \hat{u}$ is a prefix of $u \cdot v$, and $u \cdot \hat{u}^-$ is interesting. Moreover, there is a $\hat{v}$ such that $u \cdot \hat{u} \cdot \hat{v} \in \overline{L} \cap R$, because of the minimality of $u \cdot \hat{u}$ as an uninteresting string and because $\alpha_{L,R}(u \cdot \hat{u})$ is also $0$ by definition of the approximation function. (For the more general formulation, we note that $A(u \cdot \hat{u})$ may be either $\perp$ or $0$.)

By assumption that $(\alpha_{L,R} \times_\perp R')(u \cdot v) = 0$, we have that $u \cdot v \in R'$, whence we may find $v'$ such that $u \cdot \hat{u} \cdot v' \in R'$. By the condition of compatibility, we find a $\tilde{v}$ such that $u \cdot \hat{u} \cdot \tilde{v} \in R \cap R'$. By the exactness property applied twice, we find that $\chi_{L,R}(u \cdot \hat{u} \cdot \tilde{v}) = 1$ and $\chi_{L,R\cap R'}(u \cdot \hat{u} \cdot \tilde{v}) = 0$, that is $u \cdot \hat{u} \cdot \tilde{v}$ is in $L$ and is not in $L$. This is a contradiction. Thus, there is no rejecting extension.
*Subcase (N0)* This case is similar to that of (N1).
*Subcase (N$\perp$)* Assume that $\textsc{last}\alpha_{L,R\cap R'}(u) = N^\perp$. We must prove that every extension in $A$ from $\textsc{last}(u)$ is don't care. So, let $v$ be a string so that $A(u \cdot v) \neq \perp$, say $A(u \cdot v) = 1$. Then, $u \cdot v$ is in $R'$, but if $u \cdot v \in R$ also held, then we could not have $\textsc{last}\alpha_{L,R\cap R'}(u) = N^\perp$ by the exactness property; thus, $u \cdot v \notin R$ holds.

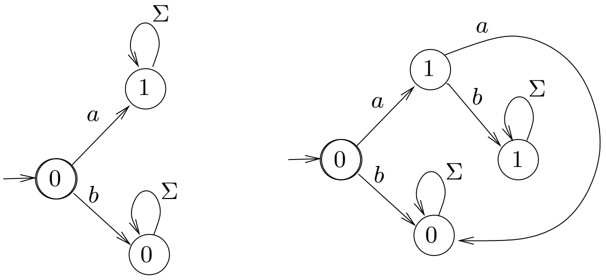
By our assumption $A(u \cdot v) = 1$, we infer that $\alpha_{L,R}(u \cdot v) = 1$ holds, and that $u \cdot v$ is uninteresting for $L$ and $R$—if $u \cdot v$ was interesting, then $\alpha_{L,R}(u \cdot v)$ should be $\perp$ by the exactness property, since $u \cdot v \notin R$ holds. But, since $u$ is interesting for $L$ and $R$, we may find a $\hat{u}$ such that $u \cdot \hat{u}$ is uninteresting, $u \cdot \hat{u}^-$ is interesting, and $u \cdot \hat{u}$ is a prefix of $u \cdot v$. Hence, $\alpha_{L,R}(u \cdot \hat{u})$ is also $1$ (or, in the general case, possibly $\perp$) and there is a $\hat{v}$ such that $u \cdot \hat{u} \cdot \hat{v}$ is in $L \cap R$. Also, since $u \cdot v \in R'$, there is $v'$ such that $u \cdot \hat{u} \cdot v' \in R'$. By assumption of compatibility, we can then find $\tilde{v}$ such that $u \cdot \hat{u} \cdot \tilde{v}$ is in $R \cap R'$. In particular, since $u \cdot \hat{u} \cdot \tilde{v}$ is in $R$, we have that $\chi_{L,R}(u \cdot \hat{u} \cdot \tilde{v}) = 1$ by the exactness property. Also, by the exactness property and the assumption that $\textsc{last}\alpha_{L,R\cap R'}(u) = N^\perp$ we have that $\chi_{L,R\cap R'}(u \cdot \hat{u} \cdot \tilde{v}) = \perp$, which is a contradiction. □

The condition of compatibility in Proposition 17 is necessary. To see this, consider the languages $L$, $R$, and $R'$ defined in Figure 7, where we have also shown the transition structure of an automaton for $L$. Each state is labeled with $1$ or $0$ according to whether it accepts or not; additionally, each state is marked with the membership status of a string reaching the state with respect to $R$ and $R'$. The idea is to construct $\chi_{L,R\cap R'}$ so that the $\alpha_{L,R\cap R'}$-automaton already on $a$ enters the $N^1$ state, see Figure 8. It will do so because after $a$ all extensions, except the empty string, lead to states not in $R \cap R'$ and because $u$ itself is in $L$, $R$, and $R'$. Intuitively speaking, the future has been restricted to $1$ except for most places, which are outside the restriction. In contrast, $\alpha_{L,R}$ "sees" more don't-care extensions of $u$, including some that are $0$. Therefore, $\alpha_{L,R}$ only goes into an uninteresting state after reading $aa$ or $ab$. Moreover, we have arranged it so that $aaa$ is in $R'$, but not in $R$ (otherwise, $\alpha_{L,R\cap R'}$ would not enter the $N^1$ state on $a$); this is the source of the failure of $\textsc{sexp}(\alpha_{L,R} \times_\perp R')$ to become identical to $\alpha_{L,R\cap R'}$ as can be seen in Figure 9. Technically, the compatibility requirement fails for the string $aa$: the empty extension brings it into $R$ and the extension $a$ brings it into $R'$, but there is no extension of $aa$ that brings it into both.



$L = a \cup ab \cup \Sigma^*$
$R = \Sigma^* \backslash a^3\Sigma^*$
$R' = \epsilon \cup a \cup b\Sigma^* \cup a^3\Sigma^*$

Figure 7: An automaton for a language $L$ where restrictions $R$ and $R'$ are not compatible.

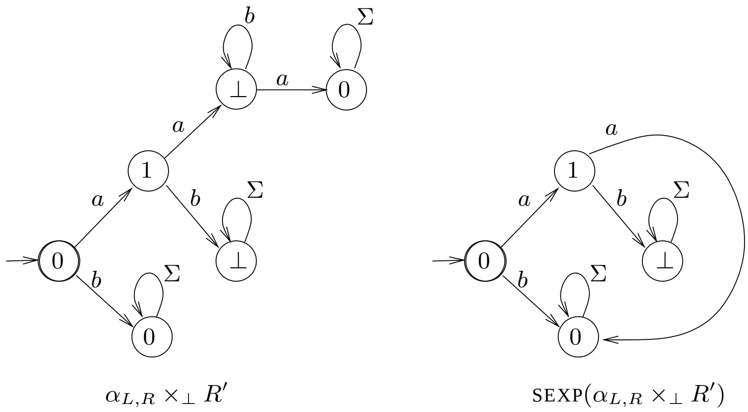


$\alpha_{L,R\cap R'}$ $\alpha_{L,R}$

Figure 8: The approximation automata for $L$.

**Proposition 18** $\alpha_{L'\cap L'',R} = \mathrm{SEXP}(\alpha_{L',R} \wedge^3 \alpha_{L'',R})$. More generally, if we replace $\alpha_{L',R}$ with any sexpartite automaton $B'$ for $L'$ and $R$ and $\alpha_{L'',R}$ with any sexpartite automaton $B''$ for $L''$ and $R$, then $\alpha_{L'\cap L'',R} = \mathrm{SEXP}(B' \wedge_\perp B'')$.

**Proof** Again, we model our proof on Proposition 14. For notational simplicity, let $\alpha = \alpha_{L'\cap L'',R}$, $\alpha' = \alpha_{L',R}$, and $\alpha'' = \alpha_{L'',R}$. We use similar conventions for the ternary valuations $\chi$, $\chi'$, and $\chi''$. (For the more general formulation, the argument is similar, except that $\alpha'$ is replaced by $B'$ and $\alpha''$ is replaced by $B''$. The exactness property and the notion of interesting remain the same.)
*Case (i)* Let $u$ be interesting. Therefore, the $\alpha'$-automaton on $u$ is not in $N^\perp$ and the $\alpha''$-automaton is not in $N^\perp$. Moreover, $u$ is interesting for $L'$ and $R$ or for $L''$ and $R$; assume the former, without loss of generality. Then, $\chi'(u) = \alpha'(u)$ holds. Since $\alpha''$ on $u$ is not in $N^\perp$, the following holds: if $\alpha''(u)$ is $\perp$, then $u$ is not in $R$ and if $\alpha''(u)$ is not $\perp$, then $\alpha''(u) = \chi''(u)$; in either case, we have that $\chi(u) = \alpha'(u) \wedge^3 \alpha''(u)$.
*Case* (ii). Let $u$ be uninteresting with $u^-$ interesting.



$\alpha_{L,R} \times_{\perp} R'$

$\textsc{sexp}(\alpha_{L,R} \times_{\perp} R')$

Figure 9: The approximation automaton $\alpha_{L,R} \times_{\perp} R'$ and its sexpartification.

*Subcase* (N1) Assume that $\textsc{last}\alpha_{L,R\cap R'}(u) = N^1$. Then, there is a $v$ such that $\chi(u{\cdot}v)$ is 1, whence $\chi'(u \cdot v)$ and $\chi''(u \cdot v)$ are also 1. By the exactness property, we infer that $\alpha'(u \cdot v)$ and $\alpha''(u \cdot v)$ also are 1. Thus, there is an accepting extension for $\alpha_{L',R} \wedge^3 \alpha_{L'',R}$.

Assume now for a contradiction that there is a $v$ such that $(\alpha_{L',R}(u{\cdot}v) \wedge^3 \alpha_{L'',R}(u{\cdot}v)) = 0$ holds. Thus, $\alpha_{L',R}(u \cdot v)$ and $\alpha_{L'',R}(u \cdot v)$ are both 0. If the $\alpha'$-automaton has entered the $N^0$ state after $u$ (in the general case, read "a $N^0$ state") or after a longer prefix of $u \cdot v$, then there is a $\hat{u}$ and a $\hat{v}$ such that $u \cdot \hat{v}$ is a prefix of $u \cdot v$ and $u \cdot \hat{v} \cdot \hat{u}$ is in $\overline{L'} \cap R$. But then, $\chi(u \cdot \hat{v} \cdot \hat{u})$ must be equal to 0, which is inconsistent with the subcase assumption that $\textsc{last}\alpha_{L,R\cap R'}(u) = N^1$. So, the $\alpha'$-automaton cannot enter the $N^0$ state after $u$. Consequently, and since we still assume that $\alpha_{L',R}(u \cdot v) = 0$, the string $u \cdot \hat{v}$ is interesting for $L'$ and $R$ and, of course by similar reasoning, for $L''$ and $R$. Hence, we have that $\chi'(u \cdot v)$ and $\chi''(u \cdot v)$ are both 0, but that contradicts again the subcase assumption.

*Subcase* (N0) This case is similar to subcase (N1).

*Subcase* (N$\perp$) Assume that $\textsc{last}\alpha_{L,R\cap R'}(u) = N^{\perp}$. Then, for all $v$, $u \cdot v$ is not in $R$, whence $u$ is uninteresting for $L'$ and $R$. In other words, both the $\alpha'$-automaton and the $\alpha''$-automaton are in an uninteresting state after reading $u$.

Now fix $v$. We must prove that $(\alpha_{L',R}(u \cdot v) \wedge^3 \alpha_{L'',R}(u \cdot v)) = \perp$. So, the only interesting cases are that both automata entered interesting states that are not $N^{\perp}$. For example, we may assume that both entered the $N^1$ state (in the general case, read "an $N^1$ state"). (The other three cases are similar.) Then, there must exist a prefix $u'$ of $u$ such that $u'$ is uninteresting for $L'$ and $R$ and minimal for this property. A similar uninteresting prefix $u''$ exists for $L''$ and $R$. Without loss of generality, we may assume that $u'$ is a prefix of $u''$. We know that for $u''^{-}$ there are extensions in both $L' \cap L'' \cap R$ and $L' \cap \overline{L''} \cap R$. Moreover, for $u''$ all extensions are in $\overline{R}$ or in $L' \cap L'' \cap R$ and there is an extension in $R$. Therefore, the $\alpha$-automaton must enter the $N^1$ state on $u$. That contradicts the subcase assumption. □

## 7 Sexpartite semantics for WS1S

We define $[\![\phi]\!]^6$ to be the approximation function $\alpha$ for the ternary valuation $\chi = [\![\phi]\!]^3$. The challenge is to calculate the approximation function of a composite formula from those of its constituents. For basic formulas, the approach is evident: we define $[\![\phi]\!]^6$ to be the automaton (or function) $\textsc{sexp}([\![\phi]\!]^3)$, where we adopt the convention that sexpartification calculates the approximation function, not the sexpartite valuation, as explained in the remarks after Proposition 13. The other cases are treated below.

### 7.1 Sexpartite negation

For negation $\phi = \neg\phi'$, we define $[\![\neg\phi']\!]^6 = \neg^3[\![\phi']\!]^6$. There is an obvious automata-theoretic algorithm for achieving this operation. It holds for reasons of symmetry that if $[\![\phi']\!]^6 = \textsc{sexp}([\![\phi']\!]^3)$, then $[\![\neg\phi']\!]^6 = \textsc{sexp}([\![\neg\phi']\!]^3) = \neg^3[\![\phi']\!]^6$.

### 7.2 Sexpartite conjunction

For conjunction $\phi = \phi' \wedge \phi''$, the case is more complicated, since $[\![\phi']\!]^3$ and $[\![\phi'']\!]^3$ may be based on different restrictions. Assume that $R' = L_{\boldsymbol{\rho}^{\mathrm{RV}}(\phi')}$, $R'' = L_{\boldsymbol{\rho}^{\mathrm{RV}}(\phi'')}$, and that $R = R' \cap R'' = L_{\boldsymbol{\rho}^{\mathrm{RV}}(\phi)}$. Moreover, define $L' = L_{\phi'}$, $L'' = L_{\phi''}$, and $L = L_\phi = L_{\phi'} \cap L_{\phi''}$.

We can express the semantics of conjunction according to:

$$[\![\phi' \wedge \phi'']\!]^6 = \textsc{sexp}(([\![\phi']\!]^6 \wedge^3 [\![\phi'']\!]^6) \times_\perp \boldsymbol{\rho}^{\mathrm{RV}}(\phi)) \tag{9}$$

This is correct: if $[\![\phi']\!]^6 = \alpha_{L',R'}$ and $[\![\phi'']\!]^6 = \alpha_{L'',R''}$ hold, then we have the following identities:

$$\begin{array}{rl}
 & \textsc{sexp}(([\![\phi']\!]^6 \wedge^3 [\![\phi'']\!]^6) \times_\perp \boldsymbol{\rho}^{\mathrm{RV}}(\phi)) \\
= & \textsc{sexp}(([\![\phi']\!]^6 \times_\perp \boldsymbol{\rho}^{\mathrm{RV}}(\phi'))\ \wedge^3\ ([\![\phi'']\!]^6 \times_\perp \boldsymbol{\rho}^{\mathrm{RV}}(\phi''))) \\
= & \textsc{sexp}((\alpha_{L',R'} \times_\perp R')\ \wedge^3\ (\alpha_{L'',R''} \times_\perp R'')) \\
= & \textsc{sexp}(\chi_{L',R'} \wedge^3 \chi_{L'',R''}) \\
= & \textsc{sexp}([\![\phi']\!]^3 \wedge^3 [\![\phi'']\!]^3) \\
= & \textsc{sexp}([\![\phi]\!]^3) \\
= & [\![\phi' \wedge \phi'']\!]^6
\end{array}$$

### 7.3 Sexpartite projection

We fix an index $i$ as found in the existential quantification $\phi = \texttt{ex2}\ P^i\ \texttt{where}\ \rho\colon \phi'$. To simplify notation, we write $u^M$ for the string $u[i \mapsto M]$.

**Proposition 19** For a given ternary valuation $\chi' = \chi_{L',R'}$, we define $\chi = \textsc{proj}^{3,i}\chi'$ and $R = \{u \mid \chi(u) \neq \bot\}$. Assume that there is some $R^i$ with $R' = R \cap R^i$. Moreover, assume that for all $u$

$$u \in R \quad \Rightarrow \forall M : u^M \in R \tag{10}$$

holds. Then these equalities between function hold:

$$\chi = \textsc{proj}^{3,i}\chi' = \textsc{proj}^{3,i}(\textsc{sexp}\chi' \times_\bot R^i) \times_\bot R$$

**Proof** We use the following notation: $\alpha' = \textsc{sexp}\chi'$, $R' = R \cap R^i$ $B' = \alpha' \times_\bot R^i$, and $B = \textsc{proj}^{3,i}B'$. We must prove that $\chi = B \times_\bot R$.

**Claim 1** The following three properties hold:

i. $\chi'(u) \neq \bot \Rightarrow B'(u) = \chi'(u)$

ii. $u \in R \Rightarrow B'(u) = \chi'(u)$

iii. $u \in R \Rightarrow B(u) = \chi(u)$

**Proof**

i. Assume that $\chi'(u) \neq \bot$ holds. Then, we know by the exactness property that $\alpha'(u) = \chi'(u)$ holds. Moreover, from assumptions $R' = R \cap R^i$ and $\chi'(u) \neq \bot$, we know that $u$ is in $R^i$. Thus, we conclude $B'(u) = (\alpha' \times_\bot R^i)(u) = \alpha'(u) = \chi'(u)$.

ii. Assume that $u$ is in $R$. If $u$ is also in $R^i$, then $\chi'(u) \neq \bot$, and we use (i). If $u$ is not in $R^i$, then $B'(u) = (\alpha' \times_\bot R^i)(u) = \bot = \chi'(u)$.

iii. Assume that $u \in R$. Consider any $M$. Then $u^M$ is in $R$ by (10), so by (ii), $B'(u^M) = \chi'(u^M)$. Thus, according to the definition of projection, we obtain that $B(u) = \chi(u)$.

□

From (iii) of the Claim, we infer that $\chi(u) = (B \times_\bot R)(u)$, since the $\times_\bot$-product with $R$ takes care of the situations when $u$ is not in $R$. □

Then, the following identity holds:

$$[\![\texttt{ex2}\ P^i\ \texttt{where}\ \rho\colon \phi']\!]^6 = \textsc{sexp}(\textsc{proj}^{3,i}([\![\phi']\!]^6 \times_\bot \rho) \times_\bot \boldsymbol{\rho}^{\mathrm{RV}}(\phi)) \tag{11}$$

To see the correctness of this identity, note that $R = \{u \mid \chi(u) \neq \bot\}$ is the set of $w$ such that $w \vDash \boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ (by Proposition 9(a)); similarly, $R' = \{u \mid \chi'(u) \neq \bot\}$ is described by $\boldsymbol{\rho}^{\mathrm{RV}}(\phi')$. Additionally, $R^i$ can be chosen to be the set represented by $\rho$. Thus, $\boldsymbol{\rho}^{\mathrm{RV}}(\phi')$ is equivalent to $\boldsymbol{\rho}^{\mathrm{RV}}(\phi) \wedge \rho$ (where for convenience we assume that $P^i$ occurs in $\phi'$). Thus, the identity $R' = R \cap R^i$ holds. Also, the condition (10) holds since $\boldsymbol{\rho}^{\mathrm{RV}}(\phi)$ does not depend on the $P^i$-track. Thus, from the Proposition the following equalities obtain:

$$\begin{aligned}
& \textsc{sexp}(\textsc{proj}^{3,i}(\llbracket\phi'\rrbracket^6 \times_\perp \rho) \times_\perp \boldsymbol{\rho}^{\mathrm{RV}}(\phi)) \\
= \ & \textsc{sexp}(\textsc{proj}^{3,i}(\alpha' \times_\perp R_i) \times_\perp R) \\
= \ & \textsc{sexp}(\textsc{proj}^{3,i}\chi') \\
= \ & \textsc{sexp}(\llbracket \texttt{ex2}\, P^i \texttt{ where}\, \rho\colon \phi'\rrbracket^3) \\
= \ & \llbracket \texttt{ex2}\, P^i \texttt{ where}\, \rho\colon \phi'\rrbracket^6
\end{aligned}$$

# 8 The sexpartite decision procedure

Propositions 18 and 19 are not quite as appealing as we would like: both involve extensive normalization. Fortunately, for first-order and $\$-$restrictions, we can show that the normalizations are superfluous or can be made computationally inexpensive.

## 8.1 Orthogonality of conjunctions of thin languages

In the sequel, we will often see that the state of the $R$-automaton is implicitly determined by the state of a sexpartite automaton. Sometimes, knowing the state $r$ is important, since a product construction with the $R$-automaton is to be carried out from $r$. Since it is the goal to avoid the construction of the full $R$-automaton, we need an effective way of representing its states. We will take advantage of the following property.

**Proposition 20** Let $R = R_1 \cap \cdots \cap R_n$, where each $R_i$ is a thin language. If $u \sim_R v$ and there is a $w$ such that $u \cdot w \in R$, then for all $i$, $1 \leq i \leq n$, it holds that $u \sim_{R_i} v$.

**Proof** Assume that $u \sim_R v$ and $u \cdot w \in R$ hold. Moreover, for a contradiction, assume that for some $i$, $u \not\sim_{R_i} v$ holds. By assumption that $R_i$ is a thin language, either $u \cdot w \notin R_i$ or $v \cdot w \notin R_i$ holds, whence from the fact $u \cdot w \in R_i$, we infer that $v \cdot w \notin R_i$ holds. But then, we would have that $v \cdot w \notin R$, which contradicts the assumptions that $u \cdot w \in R$ and $u \sim_R v$. □

The proposition entails that if we know that a state $q$ in an automaton $A$ determines a state $r$ of the automaton for $R$ (in the sense that $\forall u : \textsc{last}A(u) = q \Rightarrow \textsc{last}R(u) = r$) and $r$ is not the rejecting sink state, then $r$ is determined by the tuple of states of the $R^i$-automata.

## 8.2 Crispness

The identification of different interesting states (after collapse and minimization) that we saw in the example of Section 5.3 makes it difficult in general to recover the state of the $R$-automaton from the sexpartification. Fortunately, this phenomenon does not occur for the restrictions that we interested in.

Let us consider languages $L$ and $R$ as expressed through $\chi = \chi_{L,R}$. Let us say that string $u$ is *readily distinguished* from $v$ if there is a letter $a \in \Sigma$ such that for all $w$ it holds that $u \cdot a \cdot w \notin R$ but there is a $\hat{w}$ such that $v \cdot a \cdot \hat{w} \in R$. Intuitively speaking, the automaton for $R$ is in a non-accepting sink ($N^0$ or $N^\perp$) after reading $u \cdot a$, but it goes into a state from where it can still accept after reading $v \cdot a$. We say that $R$ is *crisp* if for all $u$ and $v$ with $u \not\sim_R v$, either $u$ is readily distinguished from $v$ or $v$ is readily distinguished from $u$.

Crispness of $R$ ensures that sexpartification does not merge interesting states of $\chi$ that are not equivalent with respect to $\sim_R$.

**Proposition 21** Assume that $R$ is crisp.

(a) Then, $\forall u, v: \ \iota_{L,R}(u) \ \wedge \ u \sim^6_{L,R} v \ \Rightarrow u \sim_R v$ holds.

(b) Moreover, if $R$ is also thin and $u$ is interesting, then $u \sim_{L,R} v \Leftrightarrow u \sim^6_{L,R} v \Leftrightarrow u \sim_{L \cap R} v$.

(c) Consider the automaton $A$ for $L \cap R$ and the $\alpha$-automaton representing the approximation valuation for $L$ and $R$. Then there is a transition-respecting mapping $f$ from the states of $A$, except the rejecting sink state, to the states of the $\alpha$-automaton except for the $N^0$ and $N^\perp$ states. On this domain and codomain, $f$ is surjective.

**Proof**

(a) The proof is by contraposition. We assume that $\iota(u)$ and $u \not\sim_R v$. If $\iota(v)$ does not hold, then $u \not\sim^6_{L,R} v$ holds, because $\iota(u)$ holds by assumption. Thus, we may further assume that $\iota(v)$ holds. Let $q'$ be the state reached in the $\chi$-automaton on $u$, and let $q''$ be defined similarly for $v$. We may assume that it is $u$ that is not readily distinguished from $v$ according to some letter $a$. Then, it will be the case that $(q', a, N^\perp)$ and $(q'', a, q)$ are transitions of the $\chi$-automaton, where $q$ is different from $N^\perp$. Consequently, the states $q'$ and $q''$ are not united during the minimization phase of sexpartification.

(b) The first bi-implication follows from (a) and Proposition 15(c). For the second bi-implication, the direction $\Rightarrow$ holds because $u \cdot w \in L \cap R$ if and only if ($u \cdot w \in R$ and ($u \cdot w \in R$ implies $u \cdot w \in L$)) if and only if, by (a) and the exactness property (since $u$ is interesting and the antecedent $u \sim^6_{L,R} v$ holds), ($v \cdot w \in R$ and ($v \cdot w \in R$ implies $v \cdot w \in L$)) if and only if $v \cdot w \in L \cap R$; the direction $\Leftarrow$ holds by virtue of Proposition 11(d), which relies on thinness of $R$.

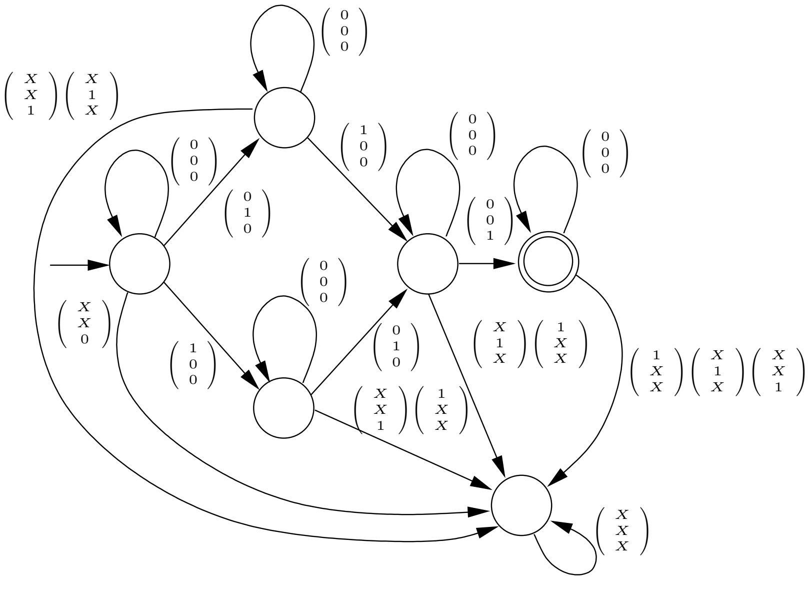
(c) It can be seen that the subautomaton induced by the interesting states of the $\alpha_{L,R}$-automaton is identical, except for state labels, to a subautomaton of $A$. (A subautomaton maybe incomplete: for some states and labels, no outgoing transitions may be defined.) Consequently, the notion of interesting state makes sense for $A$. And, we define $f$ on the subautomaton of $A$ so that it maps isomorphically to the interesting states are seen $\alpha_{L,R}$-automaton. A transition out of this set of interesting states corresponds in $\alpha_{L,R}$ to a transition to the $N^1$, $N^\perp$, or $N^0$ state. In the latter two cases, the $A$ automaton proceeds to a rejecting sink state (this is why $f$ cannot map the rejecting sink state to a state of the $\alpha_{L,R}$-automaton). In the first case, the $A$ automaton state reached is mapped by $f$ to $N^1$; so is every other state, except the rejecting sink state, that is further reachable from this state.

□

Part (c) of this Proposition is a key property: it tells us that the automaton of the conjunctive representation is the same as the automaton of the sexpartite representation except that the latter represents by at most three states the uninteresting states.

**Proposition 22** Any language $R \subseteq \Sigma^k$ that is the conjunction of $R_{\$-\text{restrict}(i)}$ for $i < k$ and $R_{\text{singleton}(i)}$ for $i \in \mathcal{S}$, where $\mathcal{S} \subseteq \{0, \ldots, k-1\}$ and where the $P^k$-track is used to encode the variable \$, is crisp.

**Proof** We proceed somewhat informally by directly studying the automaton $A$ for $R$. In particular, we fix $k = 3$ and we assume that $\mathcal{S} = \{1, 2\}$. The automaton $A$ is shown in Figure 10. We note that it has an accepting state, reached only after the first 1 in the \$-track has occurred and after each of the first-order variable tracks have seen exactly one occurrence of a 1. It also has a rejecting sink. The remaining $2^2$ states keep track of which 1s have occurred for first-order tracks. All states except the sink state has a transition to the sink state in addition to a transition to a non-sink state. □



$\begin{pmatrix}0\\0\\0\end{pmatrix}$
$\begin{pmatrix}X\\X\\1\end{pmatrix}\begin{pmatrix}X\\1\\X\end{pmatrix}$
$\begin{pmatrix}0\\0\\0\end{pmatrix}$
$\begin{pmatrix}1\\0\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\\0\end{pmatrix}$
$\begin{pmatrix}0\\1\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\\1\end{pmatrix}$
$\begin{pmatrix}0\\0\\0\end{pmatrix}$
$\begin{pmatrix}X\\X\\0\end{pmatrix}$
$\begin{pmatrix}1\\0\\0\end{pmatrix}$
$\begin{pmatrix}0\\1\\0\end{pmatrix}$
$\begin{pmatrix}X\\1\\X\end{pmatrix}\begin{pmatrix}1\\X\\X\end{pmatrix}$
$\begin{pmatrix}1\\X\\X\end{pmatrix}\begin{pmatrix}X\\1\\X\end{pmatrix}\begin{pmatrix}X\\X\\1\end{pmatrix}$
$\begin{pmatrix}X\\X\\1\end{pmatrix}\begin{pmatrix}1\\X\\X\end{pmatrix}$
$\begin{pmatrix}X\\X\\X\end{pmatrix}$

Figure 10: The automaton for first-order and \$-restrictions on $P^1$ and $P^2$.

The two preceding propositions tell us that the sexpartite representation of M2L(Str) subformulas in an WS1S encoding is identical to the ternary representation, except for the collapsing of $N^Z$ states in the ternary automaton. In particular, each interesting state specifies the state of each restriction automaton. This fact will help us formulate algorithms for conjunction and projection that avoid most explicit normalization.

## 8.3 Algorithm for conjunction

Given automata for $\alpha' = \alpha_{L',R'}$ and $\alpha'' = \alpha_{L'',R''}$, which are approximations based on crisp, thin, and compatible $R'$ and $R''$, a product automaton $B = (\Sigma, Q, q_0, \rightarrow, \lambda)$ recognizing $\alpha_{L'\cap L'',R'\cap R''}$ can be constructed as follows. We assume that the $\alpha'$-automaton is $(\Sigma, Q', q'_0, \rightarrow', \lambda')$ and the $\alpha''$-automaton is $(\Sigma, Q'', q''_0, \rightarrow'', \lambda'')$. The language $R'$ is represented by an automaton $(\Sigma, Q_{R'}, q^0_{R'}, \rightarrow_{R'}, Q^F_{R'})$ and $R''$ is represented by an automaton $(\Sigma, Q_{R''}, q^0_{R''}, \rightarrow_{R''}, Q^F_{R''})$. The state of the $R'$-automaton that is a rejecting sink is denoted $r'_{\text{rej}}$(it exists by the assumption that $R'$ is crisp); the state $r''_{\text{rej}}$ of the $R''$-automaton is defined similarly.

By the above assumptions and Proposition 22(b), any interesting state $q'$ of the $\alpha'$-automaton determines a state $r'$ of the $R'$-automaton; a similar observation holds for the $\alpha''$-automaton. We now let

$$Q = Q' \times Q'' \;\cup\; Q' \times Q_{R''} \;\cup\; Q_{R' is} \times Q'' \;\cup\; \{N^1, N^0, N^\perp\}.$$

Assuming again that the initial states are interesting (to avoid various special cases), we let $q_0 = (q'_0, q''_0)$. To define the transition relation $\rightarrow$ of $B$, we introduce the relation $\overset{a}{\rightsquigarrow}'$ on $Q' \cup (Q_{R''} \backslash \{r''_{\text{rej}}\}) \cup \{N^0, N^\perp\}$ defined according to:

$$\begin{array}{lll}
q' \overset{a}{\rightsquigarrow}' \hat{q}' & \text{if} & (q', a, \hat{q}) \in\rightarrow' \text{ and } \hat{q}' \text{ is interesting} \\
q' \overset{a}{\rightsquigarrow}' \hat{r}' & \text{if} & (q', a, \hat{q}) \in\rightarrow',\ \hat{q}' \text{ is the } N^1\text{-state, and } \hat{r}' \text{ is determined by } \hat{q}' \\
q' \overset{a}{\rightsquigarrow}' N^\perp & \text{if} & (q', a, \hat{q}) \in\rightarrow' \text{ and } \hat{q}' \text{ is the } N^\perp\text{-state} \\
q' \overset{a}{\rightsquigarrow}' N^0 & \text{if} & (q', a, \hat{q}) \in\rightarrow' \text{ and } \hat{q}' \text{ is the } N^0\text{-state} \\
r' \overset{a}{\rightsquigarrow}' \hat{r}' & \text{if} & (r', a, \hat{r}') \in\rightarrow_{R'} \text{ and } \hat{r}' \neq r'_{\text{rej}} \\
r' \overset{a}{\rightsquigarrow}' N^\perp & \text{if} & (r', a, \hat{r}') \in\rightarrow_{R'} \text{ and } \hat{r}' = r'_{\text{rej}}
\end{array}$$

Note that in the second line above $\hat{r}'$ is not $r'_{\text{rej}}$ because $\hat{q}'$ is the $N^1$-state (and $\hat{q}'$ thus determines a state of the $R'$-automaton that is not $r'_{\text{rej}}$). The transition relation $\overset{a}{\rightsquigarrow}''$ on $Q'' \cup (Q_{R'} \backslash \{r''_{\text{rej}}\}) \cup \{N^0, N^\perp\}$ is defined similarly.

Now, define $\rightarrow$ of the automaton $B$ to consist of the following transitions for a letter $a$, where $s' \overset{a}{\rightsquigarrow}' \hat{s}'$ and $s'' \overset{a}{\rightsquigarrow}'' \hat{s}''$:

$$\begin{array}{lll}
((s', s''), a, (\hat{s}', \hat{s}'')) & \text{if} & s', s'' \notin \{N^0, N^\perp\} \text{ and } (s' \in Q' \text{ or } s'' \in Q'') \\
((s', s''), a, N^\perp) & \text{if} & \hat{s}' = N^\perp \text{ or } \hat{s}'' = N^\perp \\
((s', s''), a, N^1) & \text{if} & s' \in Q_{R'} \text{ and } s'' \in Q_{R''} \\
((s', s''), a, N^0) & \text{if} & (\ \hat{s}' = N^0 \text{ and } \hat{s}'' \neq N^\perp) \text{ or } (\ \hat{s}'' = N^0 \text{ and } \hat{s}' \neq N^\perp)
\end{array}$$

Here, $N^0$, $N^1$, and $N^\perp$ are sink states, for which $\lambda$ takes on the value 0, 1, and $\perp$, respectively. For states of the form $(q', q'')$, we define $\lambda((q', q'')) = \lambda'(q') \wedge^3 \lambda''(q'')$. For states of the form $(q', r'')$, we define $\lambda((q', r'')) = \lambda'(q')$, if $r'' \in Q^F_{R''}$, and $\lambda((q', r'')) = \perp$, if $r'' \notin Q^F_{R''}$. As in the usual algorithmic treatment of product automata, we consider in the following only the states of $B$ that are reachable from the initial state.

**Proposition 23** The automaton $B$ constructed above recognizes $\alpha_{L'\cap L'',R'\cap R''}$. Moreover, there is a surjective mapping from the reachable states of the non-minimized product automaton for $(L \cap R')$ and $(L'' \cap R'')$, except for the rejecting sink state, to the reachable states of $B$ (except for one state).

**Proof** (Idea) Using the exactness property, we see that $\chi_{L'\cap L'',R'\cap R''} = (\alpha' \wedge^3 \alpha'') \times_\perp (R' \cap R'') = (\alpha' \times_\perp R'') \wedge^3 (\alpha'' \times_\perp R')$. Thus, it holds that $\alpha_{L'\cap L'',R'\cap R''} = \textsc{sexp}((\alpha' \times_\perp R'') \wedge^3 (\alpha'' \times_\perp R'))$. Now using Proposition 17, we may evaluate $\textsc{sexp}((\alpha' \times_\perp R'') \wedge^3 (\alpha'' \times_\perp R'))$ as

$$\textsc{sexp}(\textsc{sexp}(\alpha' \times_\perp R'') \wedge^3 \textsc{sexp}(\alpha'' \times_\perp R')) \tag{12}$$

To reflect the expression $\textsc{sexp}(\alpha' \times_\perp R'') \wedge^3 \textsc{sexp}(\alpha'' \times_\perp R')$ in (12), we make an automaton $C$ based on tuples of the form $((q', r''), (q'', r'))$ when states $q'$ and $q''$ are both interesting. The states $r'$ and $r''$ are determined by $q'$ and $q''$, respectively. The assumption of compatibility ensures that if both $q'$ and $q''$ are interesting then $(q', r'')$ of $\alpha' \times_\perp R''$ is also interesting. The pair $(q', r'')$ can be viewed as a state of $\textsc{sexp}(\alpha' \times_\perp R'')$ because the interesting part of $\alpha' \times_\perp R''$ is locally isomorphic to $\textsc{sexp}(\alpha' \times_\perp R'')$ (due to crispness and thinness and Proposition 21(b)).

But since $r'$ and $r''$ are determined already by $q'$ and $q''$, respectively, we may generate an isomorphic subautomaton for states of the form $((q', r''), (q'', r'))$ by just considering tuples of the form $(q', q'')$. This is how the automaton $B$ acts for interesting states: it simulates $C$. We note that during the construction of the $Q' \times Q''$-part of the state space of $B$ it is possible to simultaneously keep track of the determined $r'$ and $r''$ states—information that is needed in some cases when $q'$ or $q''$ turn to $N^1$ states. We omit a detailed discussion, but we remark that this auxiliary information is needed only for the frontier or queue of not-yet-fully explored product states; the information does not need to be stored along with the states.

We are not done describing the simulation. Say that on some transition the pair $(q', q'')$ becomes $(\hat{q}', \hat{q}'')$ with $\hat{q}'$ being the $N^1$-state and $\hat{q}''$ still interesting. The sexpartification of $\alpha' \times_\perp R''$ then yields an $N^1$-state as well. Therefore, the automaton $C$ enters a state $(N^1, (\hat{q}'', \hat{r}'))$. Thus, to continue simulating the $C$ automaton using pairs, we let the $B$ automaton enter the state $(\hat{r}', \hat{q}'')$; this is explained in detail through the rules that define its transitions. If instead a transition turns $(q', q'')$ into $(\hat{q}', \hat{q}'')$ with $\hat{q}'$ being the $N^0$-state, the the simulation of the $C$ automaton may be stopped, since the outer sexpartification in (12) will yield either an $N^0$ or an $N^\perp$ state. Other cases are explained in a similar vein.

Finally, the surjective mapping from the product automaton of the conjunctive semantics is constructed from the mappings $f'$ and $f''$ that exist according to Proposition 21(c). □

## 8.4 Algorithm for projection

To formulate an algorithm for projection that largely does away with normalizations, we substitute the identity $\textsc{proj}^{3,i} A = \textsc{bproj}^{3,i}(\textsc{fut}^{3,i} A)$ of Proposition 10 in the reformulation of projection in Proposition 19. Thus, our starting point is the identities

$$\begin{aligned}\alpha_{L,R} &= \mathrm{SEXP}\big(\mathrm{PROJ}^{3,i}\chi_{L',R'}\big)\\ &= \mathrm{SEXP}\big(\mathrm{PROJ}^{3,i}(\alpha_{L',R'} \times_\perp R^i) \times_\perp R\big)\\ &= \mathrm{SEXP}\big(\mathrm{BPROJ}^{3,i}(\mathrm{FUT}^{3,i}(\alpha_{L',R'} \times_\perp R^i)) \times_\perp R\big)\end{aligned}$$

where we use the notations of Proposition 19. The challenge is to construct a subset automaton $E$ recognizing $\mathrm{BPROJ}^{3,i}(\mathrm{FUT}^{3,i}(\alpha_{L',R'} \times_\perp R^i))$ and to avoid the subsequent product $\times_\perp R$. To do so we focus on representing $\alpha_{L',R'} \times_\perp R^i$ as an equivalent automaton $D$. We note that each interesting state $q$ of the $\alpha'$-automaton (where $\alpha' = \alpha_{L',R'}$) determines a state $r^i$ of $R^i$. So, in analogy with the construction of the automaton for conjunction, we may omit in $D$ the explicit construction of pairs corresponding to $\alpha_{L',R'} \times_\perp R^i$. And, when the $\alpha_{L',R'}$-automaton exits to an $N^1$-state, the automaton $D$ simulates $R^i$ whose rejecting states are then relabeled $\perp$. A state from this copy of $R^i$ is denoted $r^i_+$. Similarly, when the $\alpha_{L',R'}$-automaton exits to an $N^0$-state, $D$ also simulates $R^i$, but the accepting states are now labeled 0 and the rejecting states are labeled $\perp$. A state from this copy of $R^i$ is denoted $r^i_-$. When $\alpha_{L',R'}$-automaton exits to $N^\perp$, $D$ also enters a $N^\perp$-state. To avoid the $\times_\perp R$ product, we note that any subset of $D$ containing an interesting state $q$ of $\alpha'$-automaton determines the state $r$ of $R$; also, any subset containing no interesting states can be replaced by some $N^Z$-state, and the $\times_\perp R$-product is again unnecessary, because of the outer sexpartification. Thus, the effect of the $\times_\perp R$-product can be effectuated by a simultaneous traversal of the subset automaton and the automaton $R$, where subset states not satisfying $R$ are labeled $\perp$. We also note that if a subset contains say both $N^0$ and $N^\perp$ then $N^\perp$ can be removed without changing the accepted language; more generally, we may prune all subsets states so that they contain at most one non-interesting state.

**Proposition 24** Given the assumptions of Proposition 19 and the further assumption that restrictions are crisp and thin and that the automaton $E$ described above recognizes $\mathrm{SEXP}\big(\mathrm{PROJ}^{3,i}\chi_{L',R'}\big)$.

Moreover, the number of states of $E$ is at most $N \cdot 2^{|R_i|+1}$, where $N$ is the number of subset states encountered when determinizing $\mathrm{BPROJ}^i(\mathrm{FUT}^{3,i}F)$, where $F$ recognizes $L' \cap R'$.

**Proof** (Idea) We have already outlined the reasons why the above construction is correct. As regards the size of $E$, we study the partial mapping $f$ from states of the automaton $F$ recognizing $L' \cap R'$ to states of the $\alpha'$-automaton that exists according to Proposition 21(c). We note that this mapping is undefined only for the rejecting sink state $q_{\mathrm{rej}}$ of $F$, since there are two corresponding states, $N^0$ and $N^\perp$, in the $\alpha'$-automaton.

We may define a mapping $g$ that maps states $q$ of $F$ to subsets of $C$ defined as follows: if $q$ is interesting then let $g(q) = \{q'\}$; if $q$ is not interesting, but allows an accepting extension, then let $r^i$ be the state of the $R^i$-automaton determined (by Proposition 11(d) and Proposition 20) and let $g(q) = \{r^i_+\}$; and if $q$ is $q_{\mathrm{rej}}$, then $g(q) = \{N^\perp, N^0\} \cup \{r^i_-) \mid r^i \neq r_{\mathrm{rej}}\}$. Then, for any string $u$, the subset $N$ of states of $E$ reachable on $u$ is related to the subset $M$ of reachable states of $\mathrm{FUT}^{3,i}F$ on $u$:

$$N = \cup_{q \in M}\sigma(q)$$

for some function $\sigma$ that selects a nonempty subset of $g(q)$, where $q \in M$. The numeric bound of the proposition follows from combinatorial properties of $g$: only one $q$ is mapped to a non-singleton set and that set has $|R_i| + 1$ members. □

## 8.5 The sexpartite decision procedure compared

**Theorem 2** For first-order and \$-restrictions, WS1S can be decided in a way such that

(a) The sizes of the intermediate automata occurring during the sexpartite decision procedure are at most those of the conjunctively normalized semantics, except for the subset construction (and before minimization), where the automata of the sexpartite decision may be up to 32 times bigger. These bounds ignore an additive constant of 1.

(b) The conjunctively normalized automata may be exponentially bigger than the sexpartite automata.

**Proof**

(a) That the property holds for minimized automata follows from Proposition 21(b). The bounds on the number of states come from Proposition 23 and Proposition 24, where we have used the fact that the biggest $R^i$-automata stem from a simultaneous \$-restriction and a first-order restriction, which yield an automaton with $4$ states. Thus, the factor of Proposition 24 is $2^{4+1} = 32$.

(b) It suffices to consider a subformula of the form $\phi_{\mathrm{1st-vars}} = ((\cdots(p^1 = p^1 \,\&\, \cdots)\& \, p^j = p^j)$, where each $p^i$ is a first-order variable. The sexpartite automaton has one state, namely an $N^1$ state, whereas the automaton for the conjunctively normalized semantics has $2^j + 2$ states. To see this, for $j = 2$, contemplate the automaton of Figure 10, which is identical to the automaton obtained under the conjunctively normalized semantics and which has $2^2 + 2 = 6$ states. The sexpartification for this formula operates on the same automaton except that the 5 rejecting states are turned into $\perp$-states. Consequently, a single $N^1$ state arises from the collapse.

□

# 9 In practice

We showed experimental evidence in [8] that WS1S could be as fast a way to decide string-theoretic problems as M2L(Str) but only after sometimes solving by hand state explosion problems like the one discussed in Section 2.4.

Since June 1998, the Mona tool has been based on the ternary semantics for WS1S, and our state explosion problems stemming from running M2L(Str) formulas through

WS1S have disappeared. Moreover, with a default relativization mechanism that we have added to Mona, M2L(Str) formulas can be directly embedded in WS1S. The running times under these semantics is in all non-contrived cases the same (to within 5% or so) as for the ad hoc semantics we used before. (In practice, we used first-order relativizations that are not thin languages, but which enjoy similar properties.) Thus, we can state that also from a practical point of view the techniques presented here solve both the translation problem and the first-order semantics problem. We have not yet implemented the sexpartite semantics.

Future work should look at factorization techniques that would be more generally applicable. Consider for example the formula $P = Q$ & $\phi_{\mathrm{1st-vars}}$, where $\phi_{\mathrm{1st-vars}}$ simply introduces a number $j$ of first-order variables (as described in the proof of Theorem 2). This formula produces automata with a number of states exponential in $j$, even under sexpartification.

Also the sexpartite semantics should be investigated in practice; the factor 32 blow-up of subset automata before minimization may turn out to be a theoretical limit that is never encountered in practice—perhaps, a tighter analysis will show that it is too pessimistic.

**Acknowledgements** Anders Møller implemented the ideas presented here and contributed many useful insights. Jacob Elgaard found exploding Mona code from which the parity example was derived. Ken McMillan kindly discussed relativization issues with me. I thank the referees for their many positive and detailed encouragements to complete the job. Glenn Bruns asked pointed questions that led to many improvements. The patience of the editor, Olivier Danvy, is also kindly acknowledged.

# References

[1] A. Ayari and D.A. Basin. Bounded model construction for monadic second-order logics. In *Computer Aided Verification*, pages 99–112, 2000.

[2] D. Basin and N. Klarlund. Automata based symbolic reasoning in hardware verification. *Formal Methods in System Design*, pages 255–288, 1998. Extended version of “Hardware verification using monadic second-order logic,” *Computer aided verification : 7th International Conference, CAV ’95*, LNCS 939, 1995.

[3] J. Bell and M. Machover. *A Course in Mathematical Logic*. North-Holland, 1977.

[4] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing surveys*, 24(3):293–318, September 1992.

[5] J.R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundl. Math.*, 6:66–92, 1960.

[6] C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

[7] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS ’95, LNCS 1019*, 1996.

[8] A. Møller J. Elgaard, N. Klarlund. Mona 1.x: new techniques for WS1S and WS2S. In *Computer Aided Verification, CAV ’98, Proceedings*, volume 1427 of *LNCS*. Springer Verlag, 1998.

[9] P. Kelb, T. Margaria, M. Mendler, and C. Gsottberger. Mosel: a flexible toolset for Monadic Second-order Logic. In *Computer Aided Verification, CAV '97, Proceedings*, LNCS 1217, 1997.

[10] N. Klarlund. Mona & Fido: the logic-automaton connection in practice. In *CSL '97 Proceedings*. LNCS 1414, Springer-Verlag, 1998.

[11] N. Klarlund and A. Møller. *MONA Version 1.3 User Manual*. BRICS, 1998. URL: `http://www.brics.dk/mona`.

[12] Nils Klarlund, Anders Moller, and Michael I. Schwartzbach. MONA implementation secrets. *International Journal of Foundations of Computer Science*, 13(4):571–586, 2002.

[13] A. Møller and M. Schwartzbach. The Pointer Assertion Logic Engine. In *Proceedings of ACM SIGPLAN Conference of Programming Language Design and Implementation*, 2001.

[14] M. Smith and N. Klarlund. Verification of a sliding window protocol using IOA and MONA. In *FORTE/PSTV 2000: IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII), and Protocol Specification, Testing, and Verification (PSTV XX)*, pages 19–34. Kluwer Academic Publishers, 2000.

[15] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.

[16] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, chapter Languages, automata, and logic. Springer Verlag, 1997.

[17] B.A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *Sib. Math. J*, 3:103–131, 1962. In Russian. English translation: *AMS Transl.*, 59 (1966), pp. 23-55.

[18] N. Vaillette. Logical specification of finite-state transductions for NLP. *Natural Language Engineering*, 9(1), 2003.