A theory of restrictions for logics and automata

Nils Klarlund

AT&T Labs–Research (klarlund@research.att.com)

Abstract. BDDs and their algorithms implement a decision procedure for Quantified Propositional Logic. BDDs are a kind of acyclic automata. Unrestricted automata (recognizing unbounded strings of bit vectors) can be used to decide more expressive monadic second-order logics. Prime examples are WS1S, a number-theoretic logic, or a string-based notation such as those proposed in some introductory texts. It is not clear which one is to be preferred. Also, the inclusion of first-order variables in either version is problematic since their automata-theoretic semantics depends on restrictions.

In this paper, we provide a mathematical framework to address these problems. We introduce three and six-valued characterizations of regular languages under restrictions. From properties of the resulting congruences, we are able to carry out detailed state space analyses that allows us to solve the two problems in WS1S in a way that require no extra normalization calculations compared to a naive decision procedure for string-oriented logic.

We report briefly on the practical experiments that support our results. We conclude that WS1S with first-order variables is the superior choice among monadic second-order logics.

1 Motivation

Büchi[2] and Elgot[4], and independently Trakhtenbrot[13], argued almost 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 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 comparison 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. A distinguishing feature of this *number-theoretic* approach is that the semantics refer to *all* the natural numbers or *all* of finite subsets.

In contrast, the logical semantics often suggested in explanations of the logic-automaton connection, such as in [11, 12], is tied to the finiteness of the strings of a regular language. Here, the notation is interpreted over a string, which is

fixed for the purpose of the semantics. The string defines a set of positions from 0 to the length of the string minus 1; then, first-order variables range over this set, and second-order variables over its subsets. This *string-theoretic* approach is appealing for certain applications, for example in the description and verification of parameterized hardware[1]. Among other names, these logics have been called MSO(S)[12], SOM[+][11], and M2L(Str)[5, 7]. They vary slightly, but we will identify them as M2L(Str) in this paper.

There are at least three important reasons for preferring the number-theoretic approach. (1) Its mathematical semantics is simpler. (2) WS1S appears to be the stronger logic: it is easy to encode Presburger arithmetic in WS1S, but no similar encoding is known for the string-theoretic formulation. Presburger arithmetic by itself is a promising verification technique, see [3, 10]. (3) There are semantic problems in the string-theoretic formulation as pointed out in [7]; for example, what does a first-order variable denote if the string is empty and thus define no positions?

Even so, it is not obvious that any string-theoretic problem solved by a decision procedure for M2L(Str) can be effectively encoded in WS1S. More precisely, we desire an *efficient translation algorithm*, which we define to be one that in linear time transforms any formula ϕ in M2L(Str) to a formula in WS1S ϕ' such that ϕ' is decided in time linear in the time that ϕ is decided. Let us call the question of finding such an algorithm the *translation problem.* In practice, of course, we want something stronger: the total running time of going around WS1S should be no longer than using the M2L(Str) decision procedure directly.

Another problem with monadic second-order logics is that first-order variables and terms are handled by formula rewritings transforming them into second-order variables subjected to logical restrictions. Consequently, automata corresponding to subformulas are not simply determined by the mathematical semantics, but also by details of the rewritings. Alternatively, extra automata product operations can be used to *normalize* these intermediate automata with automata corresponding to the restrictions. The *first-order semantics problem* is to find a representation that is no bigger than a normalized representation, while not requiring extra normalization steps.

Contributions of this paper

In this paper, we propose solutions to the translation problem and the first-order semantics problem. Our solutions are based on a theory of restrictions that we develop as follows.

We formulate a syntax for WS1S, where restrictions are made explicit, and we provide initially three different semantics: (1) the *ad hoc* semantics that correspond to the usual treatment of first-order variables, (2) the *conjunctive semantics*, where *all* the intermediate automata are conjoined with restrictions, and (3) the *three-valued semantics.* We explain why the ad hoc semantics must be rejected, and why the conjunctive semantics would slow down the decision procedure. We show that the three-valued semantics makes most normalizations

unnecessary. Also, we indicate how the three-valued semantics can be realized using an automata-theoretic approach adapted from the standard WS1S decision procedure.

To study the question of automata sizes, we give a detailed congruence-theoretic analysis of a regular language under restrictions. We introduce a notion of a *thin* language, and we show that the restrictions occurring in the treatment of first-order variables and in the translation problem are thin. We prove that languages under thin restrictions make comparisons of the conjunctive semantics and the the three-valued semantics easy: the latter are 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 three-valued semantics are, to within a constant factor, the same as the sizes of automata of the conjunctive semantics.

We strengthen this result by exhibiting congruences based on a *six-valued semantics* that are no bigger (to an additive constant of 3) than those of the conjunctive semantics. Our main result is that the resulting decision procedure, while requiring only few normalizations, involve intermediate automata that are up to exponentially smaller than the ones occurring under the conjunctive semantics.

Finally, we report on our integration of the theory presented here into the tool Mona[9], which implements a decision procedure for WS1S. We conclude that WS1S, and not a string-oriented logic, is the superior logical interface to automata calculations.

2 WS1S: review and issues

Nutshell WS1S can be presented as follows. A formula ϕ is *composite* and of the form `~`ϕ', ϕ' `&` ϕ'', or `ex2` P^i: ϕ', or *atomic* and of the form P^i `sub` P^j, P^i `<=` P^j, P^i `=`P^j $\setminus$ P^k, or P^i `=`P^j `+1`. 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 [9, 12]. The treatment of first-order terms is discussed later.

Semantics of WS1S Given a fixed main formula ϕ_0, which we sometimes regard as an abstract syntax tree (with its root facing up), we define its semantics inductively relative to a string w over the alphabet $\mathbb{B}^k$, where $\mathbb{B} = \{0, 1\}$ and k is the number of variables in ϕ_0. We assume that ϕ_0 is closed and that each variable is bound in at most one occurrence of an existential quantifier. Generally, we assume that all formulas are subformulas of ϕ_0. 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:

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

where we have indicated 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 $(a^1, \ldots, a^k)^t$. The interpretation of P^i defined by w is the finite set $\{j \mid \text{the } j\text{th bit in the } P^i\text{-track is 1}\}$. Note that suffixing w with any string consisting of letters of the form $(0, \ldots, 0)^t$ does not change the interpretation of any variable. Therefore, we will say that w is *minimum* if it possesses no such non-empty suffix.

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

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

where we use the notation $w[P^i \mapsto M]$ for the shortest string w' that interprets all variables P^j, $j \neq i$, as w does, but interprets P^i as M. Note that if we here assume that w is minimum, then w is of the form $\tilde{w} \cdot w_0$, where all tracks, except the P^i-track, in w_0 are all 0s and either $\tilde{w}$ is empty or at least one non-P^i track in $\tilde{w}$ is of the form $\mathbb{B}^* \cdot 1$. Then, w' is of the form $\tilde{w} \cdot w''$, where w'' is 0 everywhere except for the P^i-track, which is of the form $\mathbb{B}^* \cdot 1$ if non-empty.

Note that the interpretation of ϕ_0 is independent of w, since it is a closed formula. Thus, ϕ_0 is either true or false, and we write either $\vDash \phi_0$ or $\nvDash \phi_0$. For any formula ϕ, we associate the *language* $L_\phi = \{w \mid w \vDash \phi\}$.

2.1 Automata-theoretic semantics

The automata-theoretic semantics defines a decision procedure that associates to each ϕ the minimum automaton A_ϕ accepting the language L_ϕ. For atomic formula, a small automaton (with at most three states) can be directly constructed. For a formula ϕ of the form $\texttt{~}\phi'$, the automaton A_ϕ is taken to be the complement of the automaton $A_{\phi'}$, which is calculated by induction. Note that this automata-theoretic semantics of negation is symmetric: the complement automaton is gotten by just reversing final and non-final states. The case of conjunction is handled by an automata-theoretic product construction, followed by a minimization construction. Finally, the case of quantification is slightly more complicated. Consider $\phi = w \vDash \texttt{ex2 } P^i : \phi'$. We calculate A_ϕ from $A_{\phi'}$ by means of an intermediate, nondeterministic automaton $A_{\phi''}$ that is gotten from $A_{\phi'}$ in two steps. First, any state for which a path exists to an accepting state along a string of letters of the form $(0, \ldots, 0, X, 0, \ldots, 0)^t$ (where the X means that the value of the ith component is irrelevant) is made accepting. Second, for any transition of the form (s, a, s') from state s to s', we add the transition (s, a, s''), where s'' is the state reached according to the unique transition

$(s, \overline{a}, s'')$ with $\overline{a}$ being the same letter as a except that the ith component is negated. The automaton A_ϕ is then calculated by determinizing $A_{\phi''}$, followed by a minimization construction.

2.2 Semantics of first-order variables

Adding first-order variables to 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, where the sole element denotes the value of p, see [12, 11, 8]. The restriction can be imposed by conjoining a singleton predicate `singleton(`P`)` to the formula where P is quantified. This *ad hoc strategy* means that the semantics of a formula containing p is not robust: its meaning on interpretations w not fulfilling `singleton(`P`)` is not well-defined. Even if the restriction is imposed whenever p occurs in an atomic formula, the semantics is not closed under complementation. For example, the formula $\phi = p$`=0`, where p is first-order is handled as $\phi' = P$`={0}`, where P is second-order. But the complement of ϕ' is `~(`P `= {0})`, something that is different from the representation of `~(`p `= 0)`, namely `~(`P `= {0}) & singleton(`P`)`. The solution is to conjoin the restriction to every subformula ϕ in a procedure we call *normalization*. Then, we would have a simple explanation of the language $L(\phi)$ that we call the *conjunctive semantics*.

The practical problem with the conjunctive semantics is that additional product and minimization calculations would be necessary: for each automaton A representing a subformula ϕ and each free variable P^i, the automaton representing the singleton property for P^i must be conjoined to A. Such extra calculational work slow down the decision procedure, probably by a factor of at least two. (Complementation, which is normally fast since it consists of flipping acceptance statuses of states, now would involve a product and a minimization operation; and product operations would involve at least one additional product and minimization even if the restrictions are calculated separately.) So in practice, the Mona implementation (prior to the one implemented with the results of this article) used the ad hoc strategy: the restriction for variable p is conjoined only to atomic formulas where p occur and to the formula in the existential quantification introducing p.

Ad hoc emulation of string semantics in WS1S A simplified syntax for the string-theoretic version of monadic second-order logic is the same as nutshell WS1S syntax. The satisfaction relation is denoted $\models_{string}$; it is the same as for WS1S except that quantification is changed to:

$$w \models_{string} \texttt{ex2}\ P^i : \phi' \text{ iff } \exists M \subseteq \{0, \ldots, |w| - 1\} : w[P^i \mapsto M] \models \phi'$$

where the notation $w[P^i \mapsto M]$ now has a different meaning: it denotes the string w altered so that the P^i track describes M. Thus, the witness string $w[P^i \mapsto M]$ for the existential quantification has the same length as w. The interpretation of ϕ_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 ϕ_0 holds for a string w of length i. For example, a closed formula can be written that under this semantics holds if and only if w is of even length.

To emulate $\vDash_{string}$ in $\vDash$, we must restrict all second-order terms to sets of numbers less than or equal to the last position in the string. Thus, we introduce a first-order variable \$ that simulates the entity $|w| - 1$. A *\$-constraint* for a variable expresses that the variable is a subset of $\{0, \ldots, \$\}$. Then, we normalize all formulas by conjoining \$-constraints for all free variables. The result is a WS1S formula ϕ' with one free variable \$ such that $i \vDash_{string} \phi \Leftrightarrow w \vDash \phi'$, where the \$-track of w interprets \$ as i. For example, the formula `ex1` p: `ex1` q : $p = q$ becomes in WS1S

```
ex2 P : ex2 Q :
    singleton(P) & singleton(Q) & singleton($)
    & P<=$ & Q<=$ & P sub Q & Q sub P
```

as expressed in nutshell syntax, whereas the M2L(Str) formulation is

```
ex2 P : ex2 Q :
    singleton(P) & singleton(Q)
    P sub Q & Q sub P
```

Proposition 2.1. *Under the translation outlined above, the minimized, canonical automata arising during the WS1S decision procedure are essentially the same as the ones arising during the M2L(Str) procedure except for one or two additional states.*

Proof. The WS1S automaton can be gotten from the M2L(Str) automaton by considering the \$-track as some P^i track and by adding states s_{accept} (an accepting state) and s_{reject} (a rejecting state). The transition relation of the new automaton is the same as for the old one as long as the \$-component is 0. When the \$-component is 1, corresponding to the end of the string under the M2L(Str) representation, a transition is made to s_{accept} or s_{reject} according to the accept status of the state that would have been reached in the old automaton. From s_{accept}, a transition is made to s_{reject} if the \$-component is 1 or if any other component corresponding to a first-order variable is 1; otherwise, the transition is made to s_{accept}. The s_{reject} state is connected to itself on all letters. The WS1S automaton so described may not be minimum, since the reject state may already have been present in the automaton. All other states of the old automaton are still distinct when considered as part of the new automaton.

Our practical experiments with running string-based examples translated into WS1S were based on this ad hoc strategy. We discovered the following problem.

Parity example Consider the formula `ex1` p: (p `in` P^1 `<=>` $\cdots$ `<=>` p `in` P^n) under the string-theoretic semantics. The formula holds if and only if there is a position

contained in an even number of the sets P^i. Translated into nutshell WS1S under the ad hoc strategy, the formula becomes:

$$\begin{array}{lll} \texttt{ex2} & P: & (P\ \texttt{in}\ P^1\ \texttt{\&}\ \texttt{singleton(}P\texttt{)}\ \texttt{\&}\ \texttt{singleton(\$)}\ \texttt{\&}\ P^1\texttt{<=\$})\ \texttt{<=>} \\ & & \cdots\texttt{<=>} \\ & & (P\ \texttt{in}\ P^n\ \texttt{\&}\ \texttt{singleton(}P\texttt{)}\ \texttt{\&}\ \texttt{singleton(\$)}\ \texttt{\&}\ P^n\texttt{<=\$}). \end{array} \tag{1}$$

Proposition 2.2. *The parity formula (1) produces intermediate automata whose size is doubly exponential in n. But if the restrictions are conjoined to all subformula, not only the atomic ones, then all intermediate automata have less than 6 states.*

We formalize the ad hoc semantics in the next section; but already here, it is clear that it is inadequate for restrictions.

3 WS1S with restrictions and a three-valued semantics

To give a precise understanding of restrictions, we introduce *nutshell WS1S-R*, a variation on WS1S where restrictions are made explicit. Existential quantification becomes $\texttt{ex2}\ P^i\ \texttt{where}\ \rho\texttt{:}\ \phi'$. Let $\boldsymbol{\rho}(P^i) = \rho$ be the *restriction* of variable P^i. Also, we assume that each P^i is restricted, possibly to the formula $P^i\texttt{=}P^i$, i.e., `true`. The semantics we will propose for this syntax rely on an exact understanding of the binding mechanisms in play. We say that in $\boldsymbol{\rho}(P^i)$, variable P^i is *ρ-bound*. Variable P^i is *existentially bound* in both $\boldsymbol{\rho}(P^i)$ and ϕ'. A variable occurrence P^i is *free in the conventional part of* ϕ if P^i is free in ϕ in the usual sense, where ϕ is regarded as an independent formula, and the occurrence is not within a restriction of an existential quantification within ϕ. The *relevant variables*, $\mathrm{RV}(\phi)$, for formula ϕ is the least set of variables P such that there is an occurrence of P that is not ρ-bound and that is free in the conventional part of ϕ or free in the conventional part of $\boldsymbol{\rho}(P')$, where $P' \in \mathrm{RV}(\phi)$. We define the *induced restriction* $\boldsymbol{\rho}^*(\phi)$ to be the conjunction of the restrictions of relevant variables, that is, $\bigwedge_{P^i \in \mathrm{RV}(\phi)}$.

To carry out inductive arguments, we define the partial ordering $\trianglelefteq$ among subformulas (regarded as nodes in the abstract syntax tree) as follows: $\phi \trianglelefteq \phi'$ if ϕ is a subformula of ϕ' or if there is a formula $\psi = \texttt{ex2}\ P^i\ \texttt{where}\ \boldsymbol{\rho}(P^i) : \phi''$ such that ϕ is a subformula of $\boldsymbol{\rho}(P^i)$ and ϕ' is a subformula of ϕ''. The partial ordering $\trianglelefteq$ is well-founded (a post-order labeling of nodes with numbers $0, 1, \ldots$ produces an ordering containing $\trianglelefteq$). Note for each $P \in \mathrm{RV}(\phi)$, $\boldsymbol{\rho}(P) \trianglelefteq \phi$. This will ensure that the semantic definitions to follow make sense.

The ad hoc semantics We state the ad hoc semantics using a meaning function $[\![\phi]\!]^{ah}$ (anticipating multi-valued semantics):

$$[\![\tilde{}\,\phi']\!]^{ah}w = \neg[\![\phi']\!]^{ah}w$$
$$[\![\phi' \;\texttt{\&}\; \phi'']\!]^{ah}w = [\![\phi']\!]^{ah}w \;\wedge\; [\![\phi'']\!]^{ah}w$$
$$[\![\texttt{ex2}\ P^i\ \texttt{where}\ \rho\colon \phi']\!]^{ah}w = \begin{cases} 1 & \text{if } \exists M : [\![\phi']\!]^{ah}w[P^i \mapsto M] = 1 \text{ and } [\![\boldsymbol{\rho}^*(P^i)]\!]^{ah} = 1 \\ 0 & \text{if } \forall M : [\![\phi']\!]^{ah}w[P^i \mapsto M] = 0 \text{ or } [\![\boldsymbol{\rho}^*(P^i)]\!]^{ah} = 0 \end{cases}$$
$$[\![P^i\ \texttt{sub}\ P^j]\!]^{ah}w = \begin{cases} 1 & \text{if } w \vDash P^i\ \texttt{sub}\ P^j \text{ and } [\![\boldsymbol{\rho}^*(P^i\ \texttt{sub}\ P^j)]\!]^{ah}w = 1 \\ 0 & \text{if } w \nvDash P^i\ \texttt{sub}\ P^j \text{ or } [\![\boldsymbol{\rho}^*(P^i\ \texttt{sub}\ P^j)]\!]^{ah}w = 0 \end{cases}$$

We have only shown the semantics of one kind of atomic formula; the others are treated similarly. (The normalization of atomic formulas is optional.)

The conjunction semantics This semantics is the same as the ad hoc semantics except that the restrictions are also applied to the case of `&` and `~`.

The three-valued semantics Let $\mathbb{B}^- = \mathbb{B} \cup \{-\}$ be the *extended Boolean domain.* We use $-$ 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 $-$, then the result is $-$.

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

Something seems to be missing in this semantics: the enforcement of a restriction of a variable in an existential quantification. The proposition below shows that the restriction bubbles up automatically if needed. The semantics works only if we require that every restriction is satisfiable given that the restrictions referred to by the restriction are already true. Formally, for subformula $\phi = \texttt{ex2}\ P^i\ \texttt{where}\ \rho\colon \phi'$ of ϕ_0, we require

$$\vDash \Big(\mathop{\texttt{\&}}_{P \in \mathrm{RV}(\phi') \setminus \{P^i\}} \boldsymbol{\rho}(P)\Big)\ \texttt{=>}\ \texttt{ex2}\ P^i\colon \rho \tag{2}$$

The semantics is now justified as:

Proposition 3.1. *Given the requirement (2), the following holds.*

(a) $w \nvDash \boldsymbol{\rho}(P^i)$ *for some* P^i *in* $\mathrm{RV}(\phi) \quad \Leftrightarrow \quad w \nvDash \boldsymbol{\rho}^*(\phi) \quad \Leftrightarrow \quad [\![\phi]\!]^3w = -.$
(b) $w \vDash \phi\ \texttt{\&}\ \boldsymbol{\rho}^*(\phi) \quad \Leftrightarrow \quad [\![\phi]\!]^3w = 1$
(c) $w \vDash \tilde{}\,\phi\ \texttt{\&}\ \boldsymbol{\rho}^*(\phi) \quad \Leftrightarrow \quad [\![\phi]\!]^3w = 0$

Automata-theoretic realization of the three-valued semantics The procedure outlined in Section 2.1 can be modified to reflect the three-valued semantics. The case of existential quantification requires a slightly more sophisticated reclassification of the acceptance statuses of states prior to the subset construction. Let us call the resulting algorithm the *three-valued decision procedure*.

4 Congruences 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 $u \sim_L v$ iff $\forall w : u \cdot w \in L \Leftrightarrow v \cdot w \in L$, where $u, v, w \in \Sigma^*$. The set of congruence classes is denoted $\Sigma^*/\sim_L$. This set can be regarded as the canonical, finite-state automaton.

Consider languages L, sometimes called the *property*, and R, assumed non-empty, called a *restriction*. Thus, L_ϕ and $L_{\boldsymbol{\rho}^*(\phi)}$ constitute such a pair for any subformula ϕ of ϕ_0. The *conjunction representation* is $L' = L \cap R$, and the *conjunction congruence* is $\sim_{L \cap R}$. The *three-valued representation* is not a language, but a function $\chi^3_{L,R}(u)$, defined to be 1 if $u \in L \cap R$, 0 if $u \in \overline{L} \cap R$, and $-$ if $u \notin R$. The *three-valued* congruence $\sim^3_{L,R}$ is then defined by $u \sim^3 v \Leftrightarrow$ for all w, $\chi^3_{L,R}(u \cdot w) = \chi^3_{L,R}(v \cdot w)$.

4.1 Relating the conjunction and three-valued semantics

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

$$\forall u, v, w : u \not\sim_R v \Rightarrow u \cdot w \notin R \vee v \cdot w \notin R \tag{3}$$

In particular, the canonical automaton for R has exactly one accepting state.

Proposition 4.1. *1.* $R_{singleton(i)} = \{u \in \mathbb{B}^k \mid$ *track* i *contains exactly one occurrence of a* $1\}$ *is thin.*

2. The language

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

is thin.

3. If R and R' are thin and $R \cap R' \neq \emptyset$, then $R \cap R'$ is thin.

4. Let R be thin, and let L be any language. If $u \sim_{L \cap R} v$ and u has an accepting extension, then $u \sim_R v$ (and, consequently, $u \sim^3_{L,R} v$).

5. If u and v both have no accepting extensions, then $u \sim_R v \Leftrightarrow u \sim^3_{L,R} v$.

6. Thus, if R is thin, then $|\Sigma^*/\sim^3_{L,R}| \leq |\Sigma^*/\sim_{L \cap R}| + |\Sigma^*/\sim_R|$.

From this proposition, it follows easily that all $\boldsymbol{\rho}^*(\phi)$ are thin languages if variables are subjected to first-order restrictions or \$-restrictions (or both). The proposition also tells us that $\Sigma^*/\sim^3_{L,R}$ is pieced together from $\Sigma^*/\sim_{L \cap R}$ plus a subset of $\Sigma^*/\sim_R$.

Proposition 4.2. *Assume that all restrictions are thin languages. If the automata of restrictions are bounded in size, then the sizes of the intermediate, minimized automata in the three-valued decision procedure are the same, to within an additive constant, as the sizes of corresponding automata under the conjunctive semantics.*

This result is the justification for the practical use of the three-valued semantics since usually the number of first-order variables in simultaneous use is quite small. (The size of the additive constant is exponential in the number of free first-order variables.) And as with the ad hoc semantics, normalizations are not required for most subformulas, and the automata are, apart from the $\Sigma^*/\sim_R$ parts, the same as those that occur when the automaton of every subformula is normalized.

4.2 The six-valued representation

We show next how to get rid of the boundedness assumption in Proposition 4.2. Define a string u to be *interesting* if it has (a) some extension v, called an *accepting extension*, such that $u \cdot v$ in $L \cap R$, and (b) some extension $\overline{v}$, called a *rejecting extension*, such that $u \cdot \overline{v}$ in $\overline{L} \cap R$. 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. The truth-value $\iota(u)$ denotes whether a string is interesting. Let $cut(u)$ be the shortest uninteresting prefix of u if such a prefix exists; otherwise, when all prefixes are interesting, $cut(u)$ is defined to be u. The *membership status* $\epsilon(u)$ of uninteresting u is defined by

$$\epsilon(u) = \begin{cases} 1 & \text{if } cut(u) \text{ has an accepting extension} \\ 0 & \text{if } cut(u)u \text{ has a rejecting extension} \\ - & \text{if all extensions of } cut(u) \text{ are "don't care"} \end{cases} \tag{4}$$

(These three cases are clearly mutually exclusive.) When u is interesting, $\epsilon(u)$ is defined to be $\chi^3_{L,R}(u)$. Define the *sexpartite representation* $\chi^6_{L,R}$ to be $(\iota(u), \epsilon(u))$. The *canonical six-valued congruence* $\sim^6_{L,R}$ is defined from the representation as before. Now, an equivalence class M is either interesting or non-interesting. In the latter case, there is a value $E \in \mathbb{B}_-$ such that for all $u \in M$, $\epsilon(u) = E$; moreover, for all v, $u \cdot v$ is also in M. Thus, the non-interesting equivalence classes are graph-theoretic sinks when $\Sigma/\sim^6_{L,R}$ is regarded as a finite-state automaton. There are between 0 and 3 such classes, depending on L and R.

Let c be a natural number and let $\xi : \Sigma^* \Rightarrow \mathbb{B}$ be a Boolean characterization of all strings. We say that $\sim$ *quasi-refines* $\approx$ *up to* c *under* ξ when there are strings $u_1, \ldots, u_c$ such that

$$\begin{array}{l} \forall u, u' : \xi(u) \wedge u \sim u' \;\Rightarrow\; \xi(u') \wedge u \approx u' \text{ and} \\ \forall u, u' : \neg\xi(u) \wedge u \sim u' \;\Rightarrow\; \neg\xi(u') \wedge \exists i, j : u \approx u_i \wedge u' \approx u_j \end{array} \tag{5}$$

Thus, ξ respects $\sim$ (that is, it can mapped through $\Sigma^*/\sim$) and $\sim$ is as least as fine as $\approx$ on strings for which ξ holds; but, when ξ doesn't hold, strings are mapped to one of the c designated equivalence classes of $\approx$.

Proposition 4.3. *If R is thin, then $\sim_{L\cap R}$ quasi-refines $\sim^6_{L,R}$ up to 3 under $\xi(u) =$ "there is an accepting extension of u."*

Thus, the six-valued congruence squeezes the parts of $\Sigma^*/\sim^3_{L,R}$ that corresponds to $\Sigma^*/\sim_R$ (as explained after Proposition 4.1) into at most three classes.

4.3 Six-valued semantics for WS1S and sexpartite automata

Under the six-valued semantics, the automaton corresponding to ϕ calculates $\sim^6_{L_\phi, L_{\rho^*(\phi)}}$ by a six-way partition of the states. For non-interesting strings, it may erroneously calculate a value in $\{0, 1\}$, where the three-valued semantics specifies $-$. Consequently, a product with the automaton for the restriction of a variable must be carried out before the qualifier elimination in the WS1S-R decision procedure. However, it can be shown that no minimization is necessary following this step. Let us call the resulting algorithm the *six-valued decision procedure*. Thus, we may improve Proposition 4.2:

Theorem 4.4. *Assume that all restrictions are thin languages.*

1. *The sizes of the intermediate, minimized automata occurring during the six-valued decision procedure are (to within an additive constant) less than those of the conjunctive semantics.*
2. *The conjunctive automata may be exponentially bigger than the six-valued automata.*
3. *The six-valued decision procedure require no normalization for products and complementations.*

5 In practice

We showed experimental evidence in [6] that we had found WS1S to 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 2.2.

Since June 1998, the Mona tool has been based on the three-valued semantics for WS1S, and our state explosion problems stemming from running M2L(Str) formulas through WS1S have disappeared. Moreover, with a default restriction mechanism that we have added to Mona, M2L(Str) formulas can be directly embedded in WS1S. The running times under this semantics are 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 restrictions that are not thin languages, but which enjoy similar properties.) We have not yet implemented the six-valued semantics, but there is no reason not to expect that it will run as fast, while sometimes making intermediate automata smaller.

Thus, we believe to have established WS1S as the superior choice for a practical logical notation associated with automata.

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 restriction issues with me. And thanks to the referees for pointing out some errors in an earlier version.

References

1. David Basin and Nils 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.
2. J.R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundl. Math.*, 6:66–92, 1960.
3. Tevfik Bultan, Richard Gerber, and William Pugh. Symbolic model checking of infinite state systems using presburger arithmetic. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV '97)*, volume 1254 of *LNCS*, pages 400–411. Springer, 1997.
4. C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.
5. 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.
6. Anders Møller Jacob Elgaard, Nils Klarlund. Mona 1.x: new techniques for ws1s and ws2s. In *Computer Aided Verification, CAV '98, Proceedings*, volume 1427 of *LNCS*. Springer Verlag, 1998.
7. 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.
8. N. Klarlund. Mona & Fido: the logic-automaton connection in practice. In *CSL '97 Proceedings*. LNCS 1414, Springer-Verlag, 1998.
9. Nils Klarlund and Anders Møller. *MONA Version 1.3 User Manual*. BRICS, 1998. URL: `http://www.brics.dk/mona`.
10. Thomas R. Shiple, James H. Kukula, and Rajeev K. Ranjan. A comparison of Presburger engines for EFSM reachability. In *Computer Aided Verification, CAV '98, Proceedings*, volume 1427 of *LNCS*. Springer Verlag, 1998.
11. Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.
12. Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, chapter Languages, automata, and logic. Springer Verlag, 1997.
13. 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.