A Coalgebraic Decision Procedure for WS1S

Dmitriy Traytel[1]

1 **Fakultät für Informatik, Technische Universität München, Germany**
traytel@in.tum.de

Abstract

Weak monadic second-order logic of one successor (WS1S) is a simple and natural formalism to specify regular properties. WS1S is decidable, although the decision procedure's complexity is non-elementary. Typically, decision procedures for WS1S exploit the logic–automaton connection, i.e., they escape the simple and natural formalism by translating formulas into equally expressive regular structures such as finite automata, regular expressions, or games. In this work, we devise a coalgebraic decision procedure for WS1S that stays within the logical world by directly operating on formulas. The key operation is the derivative of a formula, modeled after Brzozowski's derivatives of regular expressions. The presented decision procedure has been formalized and proved correct in the interactive proof assistant Isabelle.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases WS1S, decision procedure, coalgebra, Brzozowski derivatives, Isabelle

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

In his seminal work [8], Büchi envisioned weak monadic second-order logic of one successor (WS1S) to become a "more conventional formalism [that] can be used in place of regular expressions [...] for formalizing conditions on the behavior of automata". This vision became truth—WS1S has been used to encode decision problems in hardware verification [3], program verification [22], network verification [4], synthesis [19], as well as many others.

WS1S is a logic that supports first-order quantification over natural numbers and second-order quantification over finite (therefore "Weak") sets of natural numbers, and beyond this has few additional special predicates, such as $<$ to compare first-order variables. Equivalence of WS1S formulas is decidable, although the complexity for deciding it is non-elementary [27]. Nevertheless, the MONA tool [20] shows that the daunting theoretical complexity can often be overcome in practice by employing a multitude of smart optimizations. Similarly to Büchi, MONA's manual [24] calls WS1S a "simple and natural notation" for regular languages.

Traditionally[1], decision procedures for WS1S do not try to benefit from the conventional, simple, and natural logical notation. Instead, by exploiting the logic–automaton connection, formulas are translated into finite automata which are then minimized. During the translation all the rich algebraic formula structure including binders and high-level constructs is lost. On the other hand, the subsequent minimization might have benefited from some simplifications on the formula level.

Concerning the algebraic structure, regular expressions are situated somewhere in between WS1S formulas and automata. In earlier work [39, 40], we propose a semantics-preserving translation of WS1S formulas into regular expressions. Thereby, equivalence of

[1] The only notable exception, we are aware of, is the decision procedure implemented in the Toss tool [17] (Sect. 7).

formulas is reduced to equivalence of regular expressions. To decide the latter, we employ a coalgebraic procedure based on an ε-acceptance test and Brzozowski derivatives [7]—the coalgebra structure on regular expressions [32].

In this paper, we go one step further by defining a syntactic coalgebra structure (a short recapitulation of language coalgebra is given Sect. 2) directly on WS1S formulas (whose syntax and semantics are introduced in Sect. 3). The main contributions are:

- We define a symbolic *derivative* operation for a WS1S formula (Sect. 4).
- We define an *acceptance test* whether a formula holds in the empty interpretation (Sect. 5).
- Taking the two above notions together, we obtain a decision procedure for WS1S that operates only on formulas (Sect. 6).
- We formalize the procedure in Isabelle/HOL [29] and prove its correctness.

On the one hand, the obtained decision procedure can be considered an elegant toy—implementable only with a few hundred lines of Standard ML (and thus well-suited for formalization) and teachable in class. At this stage, our intention is not to compete with MONA's thousands of lines of tricky performance optimizations (which still can be outperformed by our procedure on well selected formulas), but rather with another verified procedure from our own previous work [39, 40]. Here, the procedure presented in this work shows a significant performance improvement (Sect. 6).

On the other hand, being able to safely decide small formulas is already of some value. In an experiment, we have randomly generated small formulas and compared the outcome of our verified algorithm with the results produced by MONA and two other (less mature) tools: Toss [17] and dWiNA [14]. As the result, we were able to point the developers of the latter two tools to corner cases where their tools gave a wrong answer [35, 36]. Admittedly, one could have performed the same test without a formalized decision procedure, but then in case of a discrepancy, determining who is right might be difficult with random formulas.

Finally, we are confident that symbolic decision procedures need not to hide behind traditional automata-based ones in terms of performance in general. Sect. 7 on related work supports this claim by several successful examples, which we expect to carry over to our setting.

Notational Conventions We employ a mixture of standard mathematical and functional programming notations. Function definitions are written in pseudo-Standard ML and we indicate the ML types, written postfix, where we consider them helpful. We assume no familiarity with Isabelle/HOL. Formal languages are sets of words. Words are represented by $\alpha\ list$, finite lists of elements of type α. Lists are either empty ($[]$) or constructed by the infix operator $::$ of type $\alpha \Rightarrow \alpha\ list \Rightarrow \alpha\ list$. The n-th element of the list xs is accessed by $xs[n]$ (zero-based). The length of xs is written $|xs|$; applied to a set this notation also denotes the set's cardinality. Lists can be iterated over using the standard combinator $\mathsf{fold} : (\alpha \Rightarrow \beta \Rightarrow \beta) \Rightarrow \alpha\ list \Rightarrow \beta \Rightarrow \beta$ with the characteristic equations $\mathsf{fold}\ f\ []\ b = b$ and $\mathsf{fold}\ f\ (x :: xs)\ b = f\ x\ (\mathsf{fold}\ f\ xs\ b)$. The type $bool$ is inhabited by two elements: 1 and 0.

2 Languages Are Coalgebras

The coalgebraic view on formal languages is, to our knowledge, due to Rutten [32]. The key observation is that the final coalgebra $(\alpha\ lang, \mathsf{out} : \alpha\ lang \Rightarrow F(\alpha\ lang))$ of the functor $F(S) = bool \times (\alpha \Rightarrow S)$ exists and is isomorphic to the standard set of words view on languages.

Using the finality of $\alpha\ lang$ we can define functions of type $\tau \Rightarrow \alpha\ lang$ by providing an F-coalgebra on τ, i.e., a function of type $\tau \Rightarrow bool \times (\alpha \Rightarrow \tau)$ that essentially describes a deterministic (not necessarily finite) automaton without an initial state. To clarify this

$$\begin{tikzcd}
\tau \arrow[r, "\langle o, \delta\rangle"] \arrow[d, "\mathsf{Lang}\ \langle o,\delta\rangle"'] & F(\tau) \arrow[d, "F\,(\mathsf{Lang}\ \langle o,\delta\rangle)"] \\
\alpha\ lang \arrow[r, "\mathsf{out}"] & F(\alpha\ lang)
\end{tikzcd}$$

Figure 1 Unique morphism $\mathsf{Lang}\ \langle o, \delta\rangle$ to the final coalgebra ($\alpha\ lang$, out)

automaton analogy, it is customary to present the F-coalgebra as two functions $\langle o, \delta\rangle$ with τ being the states of the automaton, $o : \tau \Rightarrow bool$ denoting accepting states, and $\delta : \alpha \Rightarrow \tau \Rightarrow \tau$ being the transition function. From a given $\langle o, \delta\rangle$, we obtain the function $\mathsf{Lang}\ \langle o, \delta\rangle : \tau \Rightarrow \alpha\ lang$ that assigns to a separately given initial state $t : \tau$ the language $\mathsf{Lang}\ \langle o, \delta\rangle\ t : \alpha\ lang$ and makes the diagram in Figure 1 commute.

Exploiting the isomorphism to the set of words representation, the final coalgebra $\alpha\ lang$ itself corresponds to the automaton, whose states are languages, acceptance is given by $o\ L = [\,] \in L$, and the transition function by left quotients $\delta\ a\ L = (L)_a = \{w \mid aw \in L\}$. Note that $\mathsf{Lang}\ \langle o, \delta\rangle\ (L : \alpha\ lang) = L$ in this case.

A second consequence of the finality of $\alpha\ lang$ is the coinduction principle. A relation $R : \alpha\ lang \Rightarrow \alpha\ lang \Rightarrow bool$ is a *bisimulation* iff for all languages $L, K : \alpha\ lang$ such that $R\ L\ K$ holds, we have that $[\,] \in L \Longleftrightarrow [\,] \in K$ and for all $a : \alpha$ it holds $R\ (L)_a\ (K)_a$. We write $L \sim K$ if there exists a bisimulation R such that $R\ L\ K$ and call such L and K *bisimilar*.

▸ **Theorem 1** (Coinduction). *Assume $L \sim K$. Then $L = K$.*

A necessary prerequisite for deciding bisimilarity of languages is that the languages in question admit a finite syntactic representation.[2] One example of such a finite syntactic representation are regular expressions. Although later we will work with a different finite syntactic representation (WS1S formulas), it is nevertheless instructive to shortly outline a coalgebraic decision procedure for regular expression equivalence.

Regular expressions are defined inductively as

$$RE = \varnothing \mid \boldsymbol{\varepsilon} \mid a \mid RE + RE \mid RE \cdot RE \mid RE^*$$

where $a \in \Sigma$ for a fixed alphabet $\Sigma : \alpha\ list$. The language of a regular expression r is defined as $\mathsf{L}\ r = \mathsf{Lang}\ \langle \varepsilon, \mathsf{d}\rangle\ r$ where $\varepsilon : RE \Rightarrow bool$ is the test whether the regular expression accepts the empty word and $\mathsf{d} : \alpha \Rightarrow RE \Rightarrow RE$ is the Brzozowski derivative [7] defined as usual:

$$\begin{array}{llll}
\varepsilon\ \varnothing & = 0 & \mathsf{d}\ a\ \varnothing & = \varnothing \\
\varepsilon\ \boldsymbol{\varepsilon} & = 1 & \mathsf{d}\ a\ \boldsymbol{\varepsilon} & = \varnothing \\
\varepsilon\ a & = 0 & \mathsf{d}\ a\ b & = \text{if } a = b \text{ then } \boldsymbol{\varepsilon} \text{ else } \varnothing \\
\varepsilon\ (r + s) & = \varepsilon\ r \vee \varepsilon\ s & \mathsf{d}\ a\ (r + s) & = \mathsf{d}\ a\ r + \mathsf{d}\ a\ s \\
\varepsilon\ (r \cdot s) & = \varepsilon\ r \wedge \varepsilon\ s & \mathsf{d}\ a\ (r \cdot s) & = \mathsf{d}\ a\ r \cdot s + \text{if } \varepsilon\ r \text{ then } \mathsf{d}\ a\ s \text{ else } \varnothing \\
\varepsilon\ (r^*) & = 1 & \mathsf{d}\ a\ (r^*) & = \mathsf{d}\ a\ r \cdot (r^*)
\end{array}$$

Note, that we exploit the coalgebraic view on regular expressions to define their semantics [21]. With the given definitions of the syntactic F-coalgebra on RE consisting of ε and d, the unique morphism L to the final coalgebra $\alpha\ lang$ corresponds to the language notion in the set of words world (usually defined by recursion on RE). Moreover, for the unique morphism we obtain the expected characteristic theorems: $\varepsilon\ r \Longleftrightarrow [\,] \in \mathsf{L}\ r$ and $\mathsf{L}(\mathsf{d}\ a\ r) = (\mathsf{L}\ r)_a$.

[2] This is the case for regular languages, but also for e.g., context-free languages, where equivalence (and bisimilarity) is undecidable. I.e., the finite syntactic representations prerequisite is not sufficient.

The coalgebraic decision procedure for regular expression equivalence iteratively constructs a relation $P : RE \Rightarrow RE \Rightarrow bool$ whose direct image under L is a bisimulation. For example, suppose we want to prove the regular expressions r and s being equivalent. We start with the pair (r, s), check $\varepsilon\ r \Longleftrightarrow \varepsilon\ s$ and add it to the relation P. Then we close P under the pairwise derivative d for all letters of the alphabet. Whenever a new pair (t, u) is added to P, it is checked to fulfill $\varepsilon\ t \Longleftrightarrow \varepsilon\ u$. Once P is closed under derivatives and all checks have been passed we know that it is a (syntactic) bisimulation and therefore by coinduction the languages of the input regular expressions r and s coincide.

The implementation bisim of this procedure employs a worklist algorithm to saturate P. In order to be reusable, bisim generalizes over d (abstract parameter δ), ε (abstract parameter o), syntactic representations of regular languages (types σ and τ). The abstract parameter ι is used as some initial transformation (or translation) of the syntactic representations. It will be of importance later when we instantiate σ with WS1S formulas.

```
$\mathsf{bisim} : \alpha\ list \Rightarrow (\sigma \Rightarrow \tau) \Rightarrow (\alpha \Rightarrow \tau \Rightarrow \tau) \Rightarrow (\tau \Rightarrow bool) \Rightarrow \sigma \Rightarrow \sigma \Rightarrow bool$
$\mathsf{bisim}\ \Sigma\ \iota\ \delta\ o\ s\ t =$
  let
    $\mathsf{closure}\ ([\,], \_) = 1$
    $\mathsf{closure}\ ((s, t) :: ws, P) = \text{if } o\ s \neq o\ t \text{ then } 0 \text{ else}$
      let
        $\mathsf{add\_new}\ a\ (ws, P) =$
          $\text{let } st = (\delta\ s\ a, \delta\ t\ a) \text{ in if } st \in P \text{ then } (ws, P) \text{ else } (st :: ws, \{st\} \cup P)$
      $\text{in } \mathsf{closure}\ (\mathsf{fold}\ \mathsf{add\_new}\ \Sigma\ (ws, P))$
    $st_0 = (\iota\ s, \iota\ t)$
  $\text{in } \mathsf{closure}\ ([st_0], \{st_0\})$
```

The presented algorithm is a slight generalization of a framework for deciding regular expression equivalence [30]. If it terminates, bisim decides language equivalence of σ (given notions of languages $\hat{L}$ and L for σ and τ respectively, and executable arguments ι, δ, o that behave well with respect to the language notions).

▸ **Theorem 2.** *Fix*

$$\Sigma : \alpha\ list, \quad L : \tau \Rightarrow \alpha\ lang, \quad \hat{L} : \sigma \Rightarrow \alpha\ lang, \quad \iota : \sigma \Rightarrow \tau, \quad \delta : \alpha \Rightarrow \tau \Rightarrow \tau, \text{ and } \quad o : \tau \Rightarrow bool.$$

Assume that for all $s : \sigma$, $t : \tau$, and $a \in \Sigma^*$ we have

$$L\ t \subseteq \Sigma^*, \quad L\ (\iota\ s) = \hat{L}\ s, \quad L\ (\delta\ a\ t) = (L\ t)_a, \quad o\ t \Longleftrightarrow [\,] \in L\ t, \text{ and } \quad |\{\mathsf{fold}\ \delta\ w\ t \mid w \in \Sigma^*\}| < \infty.$$

Then $\mathsf{bisim}\ \Sigma\ \iota\ \delta\ o\ s\ s' \Longleftrightarrow \hat{L}\ s = \hat{L}\ s'$.

Proof. Because of the finiteness assumption $\mathsf{closure}$ (hence also bisim) does always terminate (the set of pairs to be explored is finite).

$$\mathsf{bisim}\ \Sigma\ \iota\ \delta\ o\ s\ s' \overset{1}{\Longleftrightarrow} L\ (\iota\ s) \sim L\ (\iota\ s') \Longleftrightarrow \hat{L}\ s \sim \hat{L}\ s' \overset{2}{\Longleftrightarrow} \hat{L}\ s = \hat{L}\ s'$$

Equivalence 1 is justified by the fact that the only possibility for $\mathsf{closure}$ to return 1 is to make the worklist ws empty. Whenever the worklist becomes empty, the set of processed pairs P is a bisimulation. The nontrivial direction of 2 is justified by coinduction. ◂

The instantiation $\mathsf{eqv}_{RE}\ \Sigma = \mathsf{bisim}\ \Sigma\ (\lambda r.\ r)\ (\lambda a\ r.\ |\mathsf{d}\ a\ r|_{\mathsf{ACI}})\ \varepsilon$, where $|-|_{\mathsf{ACI}}$ is a function computing some normal form of regular expressions with respect to associativity, commutativity, and idempotence of the $+$ constructor, decides regular expression equivalence. This follows from Theorem 2 with $L = \hat{L} = \mathsf{Lang}\ \langle\varepsilon, \mathsf{d}\rangle$, where the finiteness assumption holds by a classic result due to Brzozowski [7].

3 Syntax and Semantics of WS1S

We briefly introduce WS1S, as well as a standard encoding of interpretations as formal words. More thorough introductions are given elsewhere [24,34]. Formulas are defined inductively as

$$\Phi = \mathsf{T} \mid \mathsf{F} \mid \mathsf{FO}\ \mathit{var} \mid \mathit{var} < \mathit{var} \mid \mathit{var} \in \mathit{var} \mid \Phi \vee \Phi \mid \neg\Phi \mid \exists_1\ \Phi \mid \exists_2\ \Phi$$

There are two kinds of quantifiers: $\exists_1$ quantifies over *first-order* variables, which denote natural numbers; $\exists_2$ quantifies over *second-order* variables, which denote finite sets of natural numbers. We use *de Bruijn indices* for variable bindings—therefore, no names are attached to quantifiers. An occurrence of a bound variable is just an index of type $\mathit{var} = \mathit{nat}$, that refers to its binder by counting the number of quantifiers between the occurrence and the binder. For example, the formula $\exists_1\ \mathsf{FO}\ 0 \vee (\exists_2\ 1 \in 0)$ would be customarily written as $\exists_1 x.\ \mathsf{FO}\ x \vee (\exists_2 X.\ x \in X)$ indicating second-order variables by capital letters. The first 0 and the 1 refer to the outer first-order quantifier, while the second 0 refers to the inner second-order quantifier. *Loose* de Bruijn indices, that are lacking a binder, are considered to be free. For example, $0 < 1$ corresponds to the formula $x < y$ for free x and y.

De Bruijn indices must be manipulated with great care and are hard to read. However, we prefer their usage for three reasons: First, they enable equality of formulas to hold modulo renaming of variables without further ado. Second, for any formula, its free variables are naturally ordered by the de Bruijn index. On several occasions, we will benefit from this order by using a simple list, the n-th element of which is associated to the variable with the index n. A priori this does not seem to be an advantage over using a functional assignment. However one of the occasions will be the underlying alphabet for the formula's language—an alphabet consisting of arbitrary functions is not a good idea for a decision procedure. Third, de Bruijn indices were successfully employed in the formalization.

The usual abbreviations define conjunction $\varphi\wedge\psi = \neg(\neg\varphi\vee\neg\psi)$ and universal quantification $\forall_i\ \varphi = \neg\exists_i\ (\neg\varphi)$ for $i \in \{1, 2\}$.

The truth of a formula depends on an *interpretation* of its free variables. An interpretation assigns each free variable of a formula a value. In our case, an interpretation is represented by a list of finite sets of natural numbers. The n-th entry of an interpretation is the assignment to the free variable of the formula denoted by the loose index n. Therefore, a first-order variable x is at first also assigned a finite set $I[x]$, with the condition that it must be non-empty. The value assigned by the interpretation to x is then taken to be the minimum of $I[x]$ written, $\mathsf{Min}\ (I[x])$. The idea to encode first-order variables as non-empty sets where the minimum counts, rather than singleton sets comes from MONA [24, Sect. 3.2] and leads to a more efficient decision procedure (also in our setting).

We define the WS1S-semantics of a formula φ w.r.t. an interpretation I, written $I \models \varphi$.

$$\begin{aligned}
I &\models \mathsf{T} && = 1\\
I &\models \mathsf{F} && = 0\\
I &\models (x < y) && = I[x] \neq \{\} \wedge I[y] \neq \{\} \wedge \mathsf{Min}\ (I[x]) < \mathsf{Min}\ (I[y])\\
I &\models (x \in X) && = I[x] \neq \{\} \wedge \mathsf{Min}\ (I[x]) \in I[X]\\
I &\models (\mathsf{FO}\ x) && = I[x] \neq \{\}\\
I &\models (\varphi \vee \psi) && = I \models \varphi \vee I \models \psi\\
I &\models (\neg\varphi) && = \neg(I \models \varphi)\\
I &\models (\exists_1\ \varphi) && = \exists p.\ (\{p\} :: I) \models \varphi\\
I &\models (\exists_2\ \varphi) && = \exists P.\ |P| < \infty \wedge (P :: I) \models \varphi
\end{aligned}$$

Intuitively, T is truth, F is falsity, $x < y$ compares assignments of first-order variables, $x \in X$ checks that the assignment to x is contained in the assignment to X. The (less standard) formula $\mathsf{FO}\ x$ asserts that x is a first-order variable. The Boolean connectives and quantifiers behave as expected. De Bruijn indices allow us to conveniently extend the interpretation by simply prepending the value for the most recently quantified variable when recursively entering the scope of a quantifier.

If $I \vDash \varphi$ holds, we say *I satisfies* φ or I is a *model* of φ. Formulas and interpretations must be *wellformed*: e.g., first-order variables shall not be used as second-order variables, and vice versa; no out-of-bounds accesses to the interpretation happen in $\vDash$. To ease the presentation, we omit the formal definition of wellformedness (it can be found in our formalization [38]) and consider only wellformed formulas and interpretations.

Another aspect of $\vDash$, that we want to factor out at first is the native support for first-order quantification. Therefore, in Sects. 4 and 5, we will use a different semantics that treats both quantifiers alike, written $I \Vdash \varphi$. It only differs from $\vDash$ in the quantifier cases:

$$I \Vdash (\exists_i\ \varphi) = \exists P.\ |P| < \infty \wedge (P :: I) \Vdash \varphi \quad \text{for } i \in \{1,2\}$$

In Sect. 6, we will see how a formula under the $\vDash$-semantics can be easily syntactically translated into an equivalent formula under the $\Vdash$-semantics by inserting $\mathsf{FO}\ x$ in the right places.

There exists an alternative semantics for monadic-second order logic on finite words: M2L(Str) [2, 23] (we drop the (Str) from now on). Early versions of MONA implemented the M2L semantics. While WS1S is considered to be number theoretic (as intended by Büchi), the M2L semantics fits more naturally into the realm of automata. The difference between the two semantics lies only in the treatment of quantifiers. In WS1S, a quantified variable may be assigned arbitrarily large sets. In M2L, allowed assignments are bounded by the some number $\#I$ greater than all numbers occurring in the interpretation I, i.e., $\forall x \in \bigcup_{i=0}^{|I|} I[i].\ x < \#I$. The number $\#I$ is intrinsic to an interpretation (although here for simplicity we pretend it is a separate value). For WS1S its choice does not matter for satisfiability.

Our goal is to develop a decision procedure for WS1S. However, a decision procedure for M2L will arise on the way as a natural by-product. Therefore, we introduce its semantics formally, written $I \vDash_< \varphi$. Again, only the quantifier cases differ from the definition of $\vDash$.

$$I \vDash_< (\exists_1\ \varphi) = \exists p.\ (p < \#I) \wedge (\{p\} :: I) \vDash_< \varphi$$
$$I \vDash_< (\exists_2\ \varphi) = \exists P.\ (\forall p \in P.\ p < \#I) \wedge (P :: I) \vDash_< \varphi$$

Although equally expressive as WS1S and somehow unusual in its treatment of quantifiers, the logic M2L has certain advantages, e.g., it yields a better complexity for the bounded model construction problem [2]. Therefore, M2L is not just an ad hoc intermediate construct, and obtaining a decision procedure for it (basically for free) is of some value.

As for WS1S, we first use a simplified version of M2L that ignores the quantifier types.

$$I \Vdash_< (\exists_i\ \varphi) = \exists P.\ (\forall p \in P.\ p < \#I) \wedge (P :: I) \Vdash_< \varphi \quad \text{for } i \in \{1,2\}$$

Two formulas are equivalent if they have the same models. Thus, we consider the language of a formula to be the set of its models encoded as words. The encoding of models (or interpretations in general) is standard: a finite set of natural numbers X is transformed into a list of Booleans, where 1 in the n-th positions means that $n \in X$. For interpretations the injective encoding function enc applies this transformation pointwise and pads the lists with 0s at the end to have length $\#I$. For example for $I = [\{1, 2, 3\}, \{0, 2\}]$ with $\#I = 4$, we obtain $\mathsf{enc}\ I = [[0, 1, 1, 1], [1, 0, 1, 0]]$, which we transpose to obtain the formal word w_I over the alphabet $bool^{|I|}$ (each letter is a vector, i.e., a list of fixed length).

$$w_I = \left[\begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}\right]$$

In w_I a row corresponds to an assignment to a variable. For first-order variables a row must contain at least one 1 and the first 1 from the left counts as the assigned value. Finally, the *(simplified) WS1S-language* of a formula is defined as $\mathsf{L}\ \varphi = \{w_I \mid I \Vdash \varphi\}$ and the *(simplified) M2L-language* as $\mathsf{L}_<\ \varphi = \{w_I \mid I \Vdash_< \varphi\}$. The real languages, that we are after, are defined in the same way but using $\models$ and $\models_<$ instead of $\Vdash$ and $\Vdash_<$, with some further wellformedness conditions on I and φ (Sect. 6).

4 Formula Derivatives

Brzozowski derivatives compute symbolically for a regular expression r the regular expression $\mathsf{d}\ a\ r$ whose language is the left quotient of r's language by a. A *formula derivative* is the analogous operation on a formula. Given a formula φ, its derivative $\delta\ v\ \varphi$ is a formula whose language is the left quotient of φ's language by the letter $v : bool^n$, where n is the number of free variables of φ. Formula derivatives are defined by primitive recursion as follows.

$$\delta\ v\ \mathsf{T} = \mathsf{T} \qquad \delta\ v\ (\mathsf{FO}\ x) = \begin{cases} \mathsf{T} & \text{if } v[x] \\ \mathsf{FO}\ x & \text{otherwise} \end{cases}$$

$$\delta\ v\ \mathsf{F} = \mathsf{F}$$

$$\delta\ v\ (x < y) = \begin{cases} x < y & \text{if } \neg v[x] \wedge \neg v[y] \\ \mathsf{FO}\ y & \text{if } v[x] \wedge \neg v[y] \\ \mathsf{F} & \text{otherwise} \end{cases} \qquad \delta\ v\ (x \in X) = \begin{cases} x \in X & \text{if } \neg v[x] \\ \mathsf{T} & \text{if } v[x] \wedge v[X] \\ \mathsf{F} & \text{otherwise} \end{cases}$$

$$\delta\ v\ (\varphi \vee \psi) = \delta\ v\ \varphi \vee \delta\ v\ \psi$$

$$\delta\ v\ (\neg\varphi) = \neg\delta\ v\ \varphi$$

$$\delta\ v\ (\exists_i\ \varphi) = \exists_i\ (\delta\ (1 :: v)\ \varphi \vee \delta\ (0 :: v)\ \varphi) \quad \text{for } i \in \{1, 2\}$$

Here we see the first important usage of the $\mathsf{FO}\ x$: to establish the closure of Φ under δ.

Let us try to intuitively understand some of the base cases of the definition. The language of F is the empty set. The quotient of the empty set is again empty. Thus, the derivative of F should be again F. For $\mathsf{FO}\ x$, the language will contain all words which have a 1 in the x-th row in some letter. Thus, if we quotient this language by a letter that has a 0 in the x-th row, we should obtain the same language represented by formula $\mathsf{FO}\ x$. However, once the first 1 in the x-th row has been discovered by quotienting by a letter that has a 1 in the x-th row, no further restrictions on the remaining words to be accepted remain—i.e., the quotient is the universal language represented by formula T.

Other base cases follow similarly. The most interesting recursive case is that of the existential quantifiers. The first observation that should be made is that if φ is a formula with $n+1$ free variables, then $\exists_i\ \varphi$ is a formula with n free variables (assuming that the bound variable 0 is used in φ). This implies that when we derive $\exists_i\ \varphi$ by v, we cannot derive φ by v, but only by some vector $b :: v$ because all variables (de Bruijn indices) are shifted. Since the Boolean b is unknown, we consider both possibilities: $b = 1$ and $b = 0$. The existential quantifier requires just one witness assignment for the quantified variable. This means only one of the possibilities has to hold—that is why the recursive calls $\delta\ (1 :: v)\ \varphi$ and $\delta\ (0 :: v)\ \varphi$ are connected by disjunction.

To reason more directly about interpretations and derivatives, we lift the list constructor :: on words to interpretations. This yields the operator CONS, with the characteristic property $w_{(\mathsf{CONS}\ v\ I)} = v :: w_I$ assuming $|v| = |I|$, defined as follows.

$$\mathsf{CONS}\ []\ [] = []$$

$$\mathsf{CONS}\ (1 :: v)\ (X :: I) = (\{0\} \cup (X + 1)) :: \mathsf{CONS}\ v\ I$$

$$\mathsf{CONS}\ (0 :: v)\ (X :: I) = (X + 1) :: \mathsf{CONS}\ v\ I$$

The set $X+1$ is the pointwise increment of X, namely $\{x+1 \mid x \in X\}$. Additionally, we require $\#(\mathsf{CONS}\ v\ I) = \#I + 1$. We obtain the following key characterization of the derivative (again modulo wellformedness considerations of I, φ, and v).

▸ **Theorem 3.** *$I \Vdash \delta\ v\ \varphi \Leftrightarrow \mathsf{CONS}\ v\ I \Vdash \varphi$ and $I \Vdash_{<} \delta\ v\ \varphi \Leftrightarrow \mathsf{CONS}\ v\ I \Vdash_{<} \varphi$*

Proof. Two straightforward inductions on φ. ◂

The set of all word derivatives $\{\mathsf{fold}\ \delta\ w\ \varphi \mid w \in \Sigma^*\}$ over some alphabet Σ is infinite. However, the syntactic identification of formulas that can be rewritten into each other using only associativity, commutativity and idempotence of $\vee$ (ACI) results in a finite search space which the decision procedure has to explore. This fundamental theorem mimics Brzozowski's result about dissimilar derivatives of regular expressions modulo ACI of $+$ [7].

▸ **Theorem 4.** *A formula φ has finitely many distinct word derivatives modulo ACI of $\vee$.*

Proof. By induction on φ. The most interesting case is $\exists_i\ \varphi$. By IH we know that φ has a finite set D of distinct derivatives modulo ACI. Some of the formulas in D can be disjunctions. If we repeatedly split outermost disjunctions in D until none are left, we obtain a finite set X of disjuncts. Each word derivative $\mathsf{fold}\ \delta\ w\ (\exists_i\ \varphi)$ is ACI equivalent to some $\exists_i\ (\bigvee Y)$ for some $Y \subseteq X$. Since X is finite, its powerset is also finite. Hence, there are finitely many distinct $\mathsf{fold}\ \delta\ w\ (\exists_i\ \varphi)$ modulo ACI.

Other cases are either obvious or carry over smoothly from Brzozowski's proof. ◂

Note that Theorem 4 is purely syntactic: no semantic (or language) equivalence is involved. In particular, this theorem does not follow from the Myhill-Nerode theorem nor from the fact that there are only finitely many semantically inequivalent formulas of bounded quantifier rank. Although not all language equivalent formulas are also ACI equivalent, ACI suffices for gaining finiteness.

▸ **Example 5.** The formula $\varphi = \exists_2(1 \in 0)$ ($= \exists_2 X.\ x \in X$ in conventional notation) has two distinct derivatives modulo ACI over the alphabet $\Sigma = \mathit{bool}^1 = [(0), (1)]$ (we abbreviate $\mathsf{fold}\ \delta\ w$ by δ_w).

- $\delta_{[]}\ \varphi = \varphi \equiv_{\mathsf{ACI}} \exists_2(1 \in 0 \vee 1 \in 0) = \delta_{[(0)]}\ \varphi \equiv_{\mathsf{ACI}} \delta_{(0)^n}\ \varphi$ for $n > 1$
- $\delta_{[(1)]}\ \varphi = \exists_2(\mathsf{T} \vee \mathsf{F}) \equiv_{\mathsf{ACI}} \exists_2((\mathsf{T} \vee \mathsf{F}) \vee (\mathsf{T} \vee \mathsf{F})) = \delta_{[(1),(0)]}\ \varphi \equiv_{\mathsf{ACI}} \delta_{((0)^n \cdot (1) :: w)}\ \varphi$ for $n : \mathit{nat},\ w \in \Sigma^*$

5 Accepting Formulas

Defining a function that checks whether a regular expression accepts the empty word is straightforward. A priori, the task seems barely harder for formulas—check whether the empty interpretation $\{\}^n$ (encoded by the empty word) satisfies a formula with n free variables. We start with the following attempt.

$$\begin{array}{llll}
o_{<}\ \mathsf{T} & = 1 & o_{<}\ (\mathsf{FO}\ x) & = 0 \\
o_{<}\ \mathsf{F} & = 0 & o_{<}\ (\varphi \vee \psi) & = o_{<}\ \varphi \vee o_{<}\ \psi \\
o_{<}\ (x < y) & = 0 & o_{<}\ (\neg\varphi) & = \neg o_{<}\ \varphi \\
o_{<}\ (x \in X) & = 0 & o_{<}\ (\exists_i\ \varphi) & = o_{<}\ \varphi \quad \text{for } i \in \{1, 2\}
\end{array}$$

As the name of the function indicates this acceptance test will work only for the M2L semantics, justifying M2L's automata theoretic reputation. As we will see the acceptance test for the number theoretic WS1S is by far more involved.

First, we should understand why $o_<$ works well for M2L but fails for WS1S. Therefore, we consider the formula $\varphi = \exists_1\ (1 < 0)$ ($= \exists y.\ x < y$ in conventional notation) that behaves differently under the two semantics. Interpreted in WS1S, φ is true: for each natural number x there exists a bigger one. Interpreted in M2L, the fact whether φ is true depends on the interpretation I and the value of x: for example if $\#I = 4$ then the quantifier is allowed to assign to y only values smaller than 4. If additionally $x = 3$, the formula becomes false. When checking acceptance, we only consider interpretations I with $\#I = 0$. Therefore, $o_<\ \varphi$ rightly returns $\mathsf{0}$ in the M2L setting.

Asserting $o_<\ (\exists_i\ \varphi) = o_<\ \varphi$ effectively corresponds to the restriction of M2L on the quantified values. Fortunately, unrestricted WS1S quantifiers can be related to M2L quantifiers by repeatedly padding the interpretations I with the letter $0_\Sigma = 0^{|I|}$. This means that we can reuse $o_<$ for WS1S if we can get rid of the padding. In the context of automata this step is called futurization and is implemented by traversing the automaton backwards using only 0_Σ-labeled transitions [23]. Here, we work with formulas. Thus traversing means deriving and traversing backwards means deriving from the right. We therefore introduce a second derivative operation $\reflectbox{$\delta$}$ (a mirrored δ) that computes the right quotient and is symmetric to the derivative δ. However, the base formulas are not very symmetric themselves, such that the definition of $\reflectbox{$\delta$}$ is slightly more complicated than the one for δ. In particular Φ is not closed under $\reflectbox{$\delta$}$—we need to extend it with three further base formulas: $x <_\mathsf{F} y$, $x <_\mathsf{T} y$, and $x \in_\mathsf{T} y$ with the following WS1S semantics (the M2L semantics is the same, but not really relevant).

$$\begin{aligned}
I \Vdash (x <_\mathsf{F} y) &= I[x] \neq \{\} \wedge (I[y] = \{\} \vee (I[y] \neq \{\} \wedge \mathsf{Min}\ (I[x]) < \mathsf{Min}\ (I[y])))\\
I \Vdash (x <_\mathsf{T} y) &= I[y] = \{\} \vee (I[x] \neq \{\} \wedge I[y] \neq \{\} \wedge \mathsf{Min}\ (I[x]) < \mathsf{Min}\ (I[y]))\\
I \Vdash (x \in_\mathsf{T} X) &= I[x] = \{\} \vee (I[x] \neq \{\} \wedge \mathsf{Min}\ (I[x]) \in I[X])
\end{aligned}$$

It is not easy to convey the intuition behind $x <_\mathsf{F} y$, $x <_\mathsf{T} y$, and $x \in_\mathsf{T} X$. In principle, those are random names for formulas denoting "what is left" after deriving from the right. These remainders are closely related to their non-subscripted cousins, behaving differently only if one of the assigned sets is empty. The subscript T indicates the acceptance of the empty model.

Next, we define the right derivative $\reflectbox{$\delta$}$. Also the M2L acceptance test $o_<$ must be lifted to the additional formulas. Although we will not use the derivative δ on the new formulas, we want our previously stated theorems still to hold universally. Thus, we extend δ accordingly.

$$\begin{aligned}
\reflectbox{δ}\ v\ \mathsf{T} &= \mathsf{T}\\
\reflectbox{δ}\ v\ \mathsf{F} &= \mathsf{F}\\
\reflectbox{δ}\ v\ (x < y) &= \begin{cases} x < y & \text{if } \neg v[y]\\ x <_\mathsf{F} y & \text{otherwise}\end{cases}\\
\reflectbox{δ}\ v\ (x <_\mathsf{F} y) &= \begin{cases} x <_\mathsf{T} y & \text{if } v[x] \wedge \neg v[y]\\ x <_\mathsf{F} y & \text{otherwise}\end{cases}\\
\reflectbox{δ}\ v\ (x <_\mathsf{T} y) &= \begin{cases} x <_\mathsf{T} y & \text{if } \neg v[y]\\ x <_\mathsf{F} y & \text{otherwise}\end{cases}\\
\reflectbox{δ}\ v\ (x \in X) &= \begin{cases} x \in_\mathsf{T} X & \text{if } v[x] \wedge v[X]\\ x \in X & \text{otherwise}\end{cases}\\
\reflectbox{δ}\ v\ (x \in_\mathsf{T} X) &= \begin{cases} x \in X & \text{if } v[x] \wedge \neg v[X]\\ x \in_\mathsf{T} X & \text{otherwise}\end{cases}\\
\reflectbox{δ}\ v\ (\mathsf{FO}\ x) &= \begin{cases} \mathsf{T} & \text{if } v[x]\\ \mathsf{FO}\ x & \text{otherwise}\end{cases}
\end{aligned}$$

$$\begin{aligned}
o_<\ (x <_\mathsf{F} y) &= \mathsf{0}\\
o_<\ (x <_\mathsf{T} y) &= \mathsf{1}\\
o_<\ (x \in_\mathsf{T} X) &= \mathsf{1}\\
\delta\ v\ (x <_\mathsf{F} y) &= \begin{cases} x <_\mathsf{F} y & \text{if } \neg v[x] \wedge \neg v[y]\\ \mathsf{T} & \text{if } v[x] \wedge \neg v[y]\\ \mathsf{F} & \text{otherwise}\end{cases}\\
\delta\ v\ (x <_\mathsf{T} y) &= \begin{cases} x <_\mathsf{T} y & \text{if } \neg v[x] \wedge \neg v[y]\\ \mathsf{T} & \text{if } v[x] \wedge \neg v[y]\\ \mathsf{F} & \text{otherwise}\end{cases}\\
\delta\ v\ (x \in_\mathsf{T} X) &= \begin{cases} x \in_\mathsf{T} X & \text{if } \neg v[x]\\ \mathsf{T} & \text{if } v[x] \wedge v[X]\\ \mathsf{F} & \text{otherwise}\end{cases}
\end{aligned}$$

$$\begin{aligned}
\eth\, v\, (\varphi \vee \psi) &= \eth\, v\, \varphi \vee \eth\, v\, \psi \\
\eth\, v\, (\neg\varphi) &= \neg \eth\, v\, \varphi \\
\eth\, v\, (\exists_i\, \varphi) &= \exists_i\, (\eth\, (1 :: v)\, \varphi \vee \eth\, (0 :: v)\, \varphi) \quad \text{for } i \in \{1,2\}
\end{aligned}$$

A first thing worth noticing is that again the number of derivatives $\eth$ modulo ACI of $\vee$ is finite. The proof is analogous to the proof of Theorem 4.

▸ **Theorem 6.** *A formula φ has finitely many distinct right word derivatives modulo ACI of $\vee$.*

Next, we establish a correspondence between the M2L semantics and right derivatives, similarly to Theorem 3. Therefore, we introduce the operation SNOC, symmetric to CONS, on interpretations.

$$\begin{aligned}
&\mathsf{SNOC}\ [\,]\ [\,] = [\,] \\
&\mathsf{SNOC}\ (1 :: v)\ (X :: I) = (\{\#I\} \cup X) :: \mathsf{SNOC}\ v\ I \\
&\mathsf{SNOC}\ (0 :: v)\ (X :: I) = X :: \mathsf{SNOC}\ v\ I
\end{aligned}$$

Just as for CONS, we require $\#(\mathsf{SNOC}\ v\ I) = \#I + 1$. Another routine induction yields the fundamental property of right derivatives. Note that there is no equivalent theorem for the WS1S semantics.

▸ **Theorem 7.** $I \Vdash_< \eth\, v\, \varphi \Leftrightarrow \mathsf{SNOC}\ v\ I \Vdash_< \varphi$

Finally, we define a function futurize to repeatedly apply $\eth\ 0_\Sigma$, an operation $\lfloor - \rfloor$ to recursively apply futurize to all quantifiers in a formula, and the acceptance test o for WS1S formulas. The equations of $\lfloor - \rfloor$ are matched sequentially.

```
futurize $\varphi$ =
  let fut $\varphi$ $X$ =
    if $\varphi \in X$ then $\bigvee X$
    else fut $|\eth\ 0_\Sigma\ \varphi|_{\mathsf{ACI}}$ $(\{\varphi\} \cup X)$
  in fut $|\varphi|_{\mathsf{ACI}}$ $\{\}$
```

$$\begin{aligned}
\lfloor \varphi \vee \psi \rfloor &= \lfloor \varphi \rfloor \vee \lfloor \psi \rfloor \\
\lfloor \neg\varphi \rfloor &= \neg\lfloor \varphi \rfloor \\
\lfloor \exists_i\, \varphi \rfloor &= \mathsf{futurize}\ (\exists_i\, \lfloor \varphi \rfloor) \\
\lfloor \varphi \rfloor &= \varphi \\
o\, \varphi &= o_<\, \lfloor \varphi \rfloor
\end{aligned}$$

Due to Theorem 6 the function futurize does always terminate. Moreover, we can prove the expected properties of the acceptance tests.

▸ **Theorem 8.** *Let n be the number of free variables in φ and I an interpretation with $|I| = n$. Then*

1. $o_<\, \varphi \Leftrightarrow \{\}^n \Vdash_< \varphi$,
2. $I \Vdash_< \mathsf{futurize}\ \varphi \Leftrightarrow \exists k.\ ((\mathsf{SNOC}\ (0^n))^k\ I) \Vdash_< \varphi$,
3. $I \Vdash_< \lfloor \varphi \rfloor \Leftrightarrow I \Vdash \varphi$, *and*
4. $o\, \varphi \Leftrightarrow \{\}^n \Vdash \varphi$,

where $(\mathsf{SNOC}\ (0^n))^k$ *denotes the k-fold repeated application of* $\mathsf{SNOC}\ (0^n)$.

Proof. Prop. 1 follows by routine induction on φ. Prop. 2 follows from the fact that futurize φ is the disjunction of all right derivatives of φ by words of the form $(0^n)^k$ for some k and Theorem 7. Prop. 3 is again a routine induction using Prop. 2 in the existential quantifier cases. Finally, for Prop. 4 we calculate: $o\, \varphi \Leftrightarrow o_<\, \lfloor \varphi \rfloor \overset{\text{Prop. 1}}{\Leftrightarrow} \{\}^n \Vdash_< \lfloor \varphi \rfloor \overset{\text{Prop. 3}}{\Leftrightarrow} \{\}^n \Vdash \varphi$. ◂

6 Decision Procedure for WS1S

We are close to being able to instantiate the generic bisimulation computation with our notions to obtain a decision procedure for WS1S. The only missing jigsaw pieces are the treatment of first-order quantification and wellformedness checks on interpretations and formulas that were swept under the carpet so far. Let us make it more precise: in full detail the language of a formula is not just $\mathsf{L}\,\varphi = \{w_I \mid I \Vdash \varphi\}$, but rather has an additional explicit wellformedness condition $\hat{\mathsf{L}}\,\varphi = \{w_I \mid I \vDash \varphi \wedge (\forall x \in \mathsf{FOV}\,\varphi.\ I[x] \neq \{\})\}$ (similarly $\hat{\mathsf{L}}_<\,\varphi = \{w_I \mid I \vDash_< \varphi \wedge (\forall x \in \mathsf{FOV}\,\varphi.\ I[x] \neq \{\})\}$), where $\mathsf{FOV}\,\varphi$ is the set of free first-order variables of φ. Then, the language of the negated formula is

$$\hat{\mathsf{L}}\,(\neg\varphi) = \{w_I \mid \neg(I \vDash \varphi) \wedge (\forall x \in \mathsf{FOV}\,\varphi.\ I[x] \neq \{\})\} \neq \Sigma^* \setminus \hat{\mathsf{L}}\,\varphi.$$

We conclude that complementation on the language level does not correspond to the negation on the formula level. The solution is once again inspired by the treatment of this issue in MONA[3]: certain critical w.r.t. negation wellformedness annotations (critical here means: cannot be checked statically for a formula before starting to derive), called *restrictions*, are syntactically introduced in the formula by the initial transformation ι of bisim. Here, the formula $\mathsf{FO}\ x$ plays another important role, since in our case restrictions merely ensure that only non-empty sets are assigned to first-order variables.

For example, the acceptance test o would cast the formula $\forall_1\ \mathsf{FO}\ 0 = \neg\exists_1\ \neg\mathsf{FO}\ 0$ false without restrictions. However, if we conjoin the additional information that all first-order variables should be non-empty right under their introducing quantifiers and once globally for free first-order variables of a formula, we obtain the desired behavior: $o\ (\neg\exists_1\ \mathsf{FO}\ 0 \wedge \neg\mathsf{FO}\ 0) = 1$. The conjunction of restrictions is implemented by the function $\mathsf{restrict}$ (with equations of the helper function restr matched sequentially and the function FOV denoting the list of free first-order variables of a formula).

$$\begin{aligned}
\mathsf{restr}\ (\varphi \vee \psi) &= \mathsf{restr}\ \varphi \vee \mathsf{restr}\ \psi\\
\mathsf{restr}\ (\neg\varphi) &= \neg\mathsf{restr}\ \varphi\\
\mathsf{restr}\ (\exists_1\ \varphi) &= \exists_1\ (\mathsf{restr}\ \varphi \wedge \mathsf{FO}\ 0)\\
\mathsf{restr}\ (\exists_2\ \varphi) &= \exists_2\ (\mathsf{restr}\ \varphi)\\
\mathsf{restr}\ \varphi &= \varphi\\
\mathsf{restrict}\ \varphi &= \mathsf{fold}\ (\lambda i\ \varphi.\ \varphi \wedge \mathsf{FO}\ i)\ (\mathsf{FOV}\ \varphi)\ (\mathsf{restr}\ \varphi)
\end{aligned}$$

The function $\mathsf{restrict}$ translates between L and $\hat{\mathsf{L}}$:

▸ **Theorem 9.** $\mathsf{L}\,(\mathsf{restrict}\ \varphi) = \hat{\mathsf{L}}\,\varphi$ *and* $\mathsf{L}_<\,(\mathsf{restrict}\ \varphi) = \hat{\mathsf{L}}_<\,\varphi$.

Finally, we are ready to instantiate bisim to obtain decision procedures for the M2L and the WS1S semantics. Note that the argument formulas still have to be checked for simple wellformedness properties statically (e.g., by a type system that ensures that no variable is used as both first-order and second-order simultaneously). Also the alphabet Σ must be a list of vectors whose lengths correspond to the number of free variables.

$$\begin{aligned}
\mathsf{eqv}_<\ \Sigma\ \varphi\ \psi &= \mathsf{bisim}\ \Sigma\ (\lambda\varphi.\ |\mathsf{restrict}\ \varphi \wedge \mathsf{FO}\ z|_{\mathsf{ACI}})\ (\lambda a\ \varphi.\ |\delta\ a\ \varphi|_{\mathsf{ACI}})\ o_<\ \varphi\ \psi\\
\mathsf{eqv}\ \Sigma\ \varphi\ \psi &= \mathsf{bisim}\ \Sigma\ (\lambda\varphi.\ |\mathsf{restrict}\ \varphi|_{\mathsf{ACI}})\ (\lambda a\ \varphi.\ |\delta\ a\ \varphi|_{\mathsf{ACI}})\ o\ \varphi\ \psi
\end{aligned}$$

[3] Recent versions of MONA use a more efficient solution than the outlined simple approach [23].

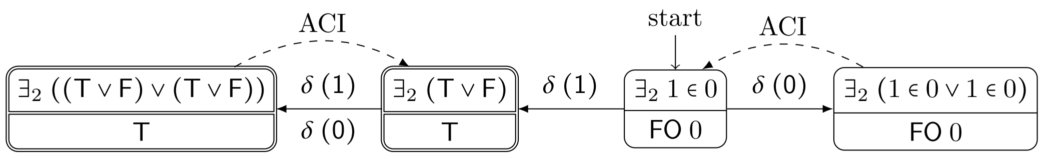

Figure 2 Bisimulation constructed by eqv for $\exists_2 1 \in 0 \equiv \mathsf{FO}\ 0$

The last unexpected thing is the conjunction of a fresh variable z (not occurring in φ and ψ) in the decision procedure for M2L. This is required to circumvent the situation that a M2L formula can only be true in the empty model if it does not quantify over first-order variables and has no free first-order variables. Because of this $\forall_1 \mathsf{FO}\ 0$ would be different from T but equivalent to $\mathsf{FO}\ 42$. Conjoining a fresh first-order variable restriction essentially means, that the first pair of a bisimulation must not be checked for acceptance. This admittedly surprising workaround is required due to the unusual quantification rules of M2L. In WS1S there is nothing special about empty models.

▸ **Theorem 10.** $\mathsf{eqv}_< \ \Sigma\ \varphi\ \psi \Leftrightarrow \hat{\mathsf{L}}_< \ \varphi = \hat{\mathsf{L}}_< \ \psi$ *and* $\mathsf{eqv}\ \Sigma\ \varphi\ \psi \Leftrightarrow \hat{\mathsf{L}}\ \varphi = \hat{\mathsf{L}}\ \psi$

Proof. Using Theorem 2 and discharging its assumptions using the Theorems 3, 4, and 8 together with the characteristic properties of CONS and $\mathsf{restrict}$. ◂

▸ **Example 11.** Figure 2 shows the bisimulation produced by our decision procedure for the equivalent formulas $\varphi = \exists_2 1 \in 0$ and $\psi = \mathsf{FO}\ 0$ over the alphabet $\Sigma = \mathit{bool}^1 = [(1), (0)]$. Graphically, a bisimulation can also be viewed as the product automaton of the automata whose states are derivatives of the formulas with transitions given by δ. Accepting states, i.e., pairs (φ, ψ) for which $o\ \varphi$ and $o\ \psi$ holds, are marked with a double margin. Previously seen ACI equivalent states, detected using the ACI normalization $|-|_{\mathsf{ACI}}$ and marked by a dashed back-edge, are not explored further (and not even checked for being both accepting or both not accepting).

The example suggests that a stronger normalization function than $|-|_{\mathsf{ACI}}$ might be useful. For example, trivial identities involving T or F (e.g., $\mathsf{T} \vee \varphi \equiv \varphi$) should be also simplified. Indeed in our tests we employ a much stronger normalization function, that additionally eliminates quantifiers $\exists_i\ \varphi \equiv \varphi$ provided that 0 is not free in φ. MONA performs a similar optimization, which according to its user manual [24] "can cause tremendous improvements". However, MONA can perform this optimization only on the initially entered formula, while for us it is available for every pair of formulas in the bisimulation because we keep the rich formula structure.

This is only one of a wealth of optimizations performed in MONA. Competing with this state-of-the-art tool is not our main objective yet. Our procedure does outperform MONA on specific examples, e.g., formulas of the form $\varphi \wedge \psi$ where the minimal automata for φ is small, the one for ψ is huge, and for almost all $w \in \Sigma^*$ either $\mathsf{fold}\ \delta\ w\ \varphi = \mathsf{F}$ or $\mathsf{fold}\ \delta\ w\ \psi = \mathsf{F}$. When extending the syntax with formulas of the form $x = n$ for constants n (and defining the left and right derivatives for them), such examples can be immediately constructed: $x = 10 \wedge x = 10000$ takes MONA roughly 40 minutes to disprove, while our procedure requires only a few seconds. MONA computes eagerly both minimal automata for φ and ψ, whereas our procedure is lazier, which pays off here. More generally, we can compete with and outperform by far another existing verified symbolic decision procedure for WS1S, namely the one translating formulas into regular expressions proposed by us earlier [39].

We have evaluated the performance of $\mathsf{eqv}_<$ and eqv for the family of formulas ψ_n from our earlier work [39] using the same hardware and compared it against the best timings

n	$\psi_n \equiv_{\mathsf{MSO}} \mathsf{T}$ [39]	$\psi_n \equiv_{\mathsf{WS1S}} \mathsf{T}$ [39]	$\mathsf{eqv}_< \ \psi_n \ \mathsf{T}$	$\mathsf{eqv} \ \psi_n \ \mathsf{T}$
0	0	0.4	0	0
1	0	14.4	0	0
2	3.9	–	0	0
10	–	–	0.07	0.07
15	–	–	3.43	3.46

Figure 3 Running times (in sec) of regular expression- and formula-based decision procedures

obtained for M2L and WS1S in there. The encouraging results are shown in Figure 3. A – means that the computation did not terminate within an hour. MONA solves all those examples instantly. We should note that the optimization $\exists_i \ \varphi \equiv \varphi$ provided that 0 is not free in φ is indeed a tremendous improvement—without its usage our decision procedure scales only slightly better than the regular expression-based one. The possibility to inspect the binder structure after several derivation steps, which is not available when translating to regular expressions, is the main advantage of the proposed decision procedure.

7 Related Work and Discussion

Much of the related work has been discussed earlier. We outline further references which yield some inspiration for generalizations and performance improvement of our algorithm.

For regular expression equivalences different variations of Brzozowski derivatives have been proposed. Most prominently, the partial derivatives have been introduced by Antimirov [1] and generalized by Caron et al. [9] to support complements and intersections. Partial derivatives capture ACI equivalence directly in the data structure of finite sets, eliminating the need for the subsequent ACI normalization after deriving. This idea immediately carries over to formulas and, since decision procedures based on partial derivatives tend to perform better in practice, is worth investigating. Further variations of derivatives exist, however a generalization to complements and intersections seems difficult. In earlier work [30], we provide an overview and a performance comparison.

Beyond regular expressions, there exist generalizations of Brzozowski derivatives for context-free grammars [12, 28] and Kleene algebra with tests (KAT) [26]. In the latter case, derivatives give rise to efficient coalgebraic decision procedures for KAT [31] and its network-targeted specialization NetKAT [15]. Our work can be seen in line with this work, albeit replacing “efficient” with “verified”. The ultimate goal is to develop procedures that fulfill both predicates. Below we outline how to possibly improve on efficiency for WS1S.

Our algorithm constructs a bisimulation. It is well known that a bisimulation up to congruence and context is often much smaller. Bonchi and Pous [6] successfully employ this technique in practice, outperforming state-of-the-art solvers for NFA equivalence. Fiedor et al. [14] employ the modern antichain technique for nondeterministic automata in the context of deciding WS1S. Their tool, dWiNA, outperforms MONA for certain classes of formulas (in particular for formulas in prefix normal form). Bonchi and Pous [6] showed that bisimulation up to congruence and context is an even further improvement over antichains for nondeterministic automata. Either of these techniques will improve the performance of our algorithm (extended to work nondeterministically using partial derivatives) at least for some classes of formulas.

MONA benefits a lot from the BDD representation of automata [25]. Pous [31] presents a fast symbolic decision procedure for KAT employing BDDs whose leafs are KAT expressions and lifting derivative operations to those BDDs. We expect this technique to be applicable to WS1S formulas.

An alternative to conjoining formulas FO 0 to every first-order quantifier, that turned out to be more efficient in MONA, is to work with a three-valued semantics [23] of acceptance (the third value indicates that no 1 for a first-order variable has been read yet). In our setting, this would require a change of the underlying final coalgebra (essentially working with Moore machines instead of DFAs), and therefore a complication of the syntactic coalgebra.

Ganzow and Kaiser [16,17] present a very general, model theoretic decision procedure for weak monadic second-order logic on inductive structures that is inspired by Shelah's composition method [33]. Although they tackle the problem from a completely different angle, their computation of the next-state function is also performed symbolically on formulas and therefore has a similar flavor as our left derivatives. However, after obtaining the next-state function in such a way, Ganzow and Kaiser escape the formula world by translating the problem into a finite reachability game. Their algorithm is implemented in the Toss tool. It outperforms MONA for certain formulas, by outsourcing parts of the computation to state-of-the-art SAT-solvers. In contrast, by employing the coalgebraic machinery, we do even more on formulas directly, e.g., by computing the acceptance test via right derivatives. Overall, the work by Ganzow and Kaiser can serve as a starting point on how to generalize our procedure beyond WS1S.

Since S1S, in which second-order variables quantify over potentially infinite sets, is also decidable, the question naturally arises whether our ideas are applicable there as well. Some hope for the affirmative answer can be drawn from the work by Esparza and Kretínský [13], who employ something similar to "derivatives of LTL formulas" (but in conjunction with a non-standard automaton model) for model checking. Furthermore, Ciancia and Venema [10] recast a more standard automaton model on infinite words into the coalgebraic setting.

Methodologically, Isabelle facilitated convenient reasoning about logic and in particular formal languages coalgebraically, not least due to recently introduced codatatype support [5]. The coinductive theory of languages is interesting in itself and formalized separately [37].

8 Conclusion

We have presented a simple coalgebraic algorithm for deciding WS1S. The algorithm operates directly on formulas by deriving them symbolically. The algorithm has been formalized and proved correct in Isabelle/HOL. Using Isabelle's code generator [18], we can extract a fully verified prover for WS1S from the formalization. The formalization, comprising 4000 lines of code, is available online [38]. For readers unfamiliar with Isabelle, a separate manual implementation of the algorithm in Standard ML is also available [38].

In future work, we plan to implement optimizations hinted at in the previous section and to explore derivatives for other weaker and stronger logics such as Presburger Arithmetic, LTL, S1S, WS2S, and S2S following Conway's motto "differentiation is a skill worth acquiring" [11, p. 43].

Acknowledgment

I am grateful to Tobias Nipkow for the continuous encouragement of this work. Damien Pous and several anonymous reviewers helped to improve the paper through a multitude of precious comments. This research is supported by the Deutsche Forschungsgemeinschaft (DFG) program Program and Model Analysis (PUMA, doctorate program 1480).

References

1 V. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, Mar. 1996.

2 A. Ayari and D. Basin. Bounded model construction for monadic second-order logics. In E. A. Emerson and A. P. Sistla, eds., *CAV 2000*, vol. 1855 of *LNCS*, pp. 99–112. Springer, 2000.

3 D. Basin and N. Klarlund. Automata based symbolic reasoning in hardware verification. *Formal Methods In System Design*, 13:255–288, 1998. Extended version of: “Hardware verification using monadic second-order logic," *CAV '95*, LNCS 939.

4 K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl. Abstracting WS1S systems to verify parameterized networks. In S. Graf and M. I. Schwartzbach, eds., *TACAS 2000*, vol. 1785 of *LNCS*, pp. 188–203. Springer, 2000.

5 J. C. Blanchette, J. Hölzl, A. Lochbihler, L. Panny, A. Popescu, and D. Traytel. Truly modular (co)datatypes for Isabelle/HOL. In G. Klein and R. Gamboa, eds., *ITP 2014*, vol. 8558 of *LNCS*, pp. 93–110. Springer, 2014.

6 F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In R. Giacobazzi and R. Cousot, eds., *POPL 2013*, pp. 457–468. ACM, 2013.

7 J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, Oct. 1964.

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

9 P. Caron, J.-M. Champarnaud, and L. Mignot. Partial derivatives of an extended regular expression. In A.-H. Dediu, S. Inenaga, and C. Martín-Vide, eds., *LATA 2011*, vol. 6638 of *LNCS*, pp. 179–191. Springer, 2011.

10 V. Ciancia and Y. Venema. Stream automata are coalgebras. In D. Pattinson and L. Schröder, eds., *CMCS 2012*, vol. 7399 of *LNCS*, pp. 90–108. Springer, 2012.

11 J. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall mathematics series. Dover Publications, Incorporated, 2012.

12 N. A. Danielsson. Total parser combinators. In P. Hudak and S. Weirich, eds., *ICFP 2010*, pp. 285–296. ACM, 2010.

13 J. Esparza and J. Kretínský. From LTL to deterministic automata: A Safraless compositional approach. In A. Biere and R. Bloem, eds., *CAV 2014*, vol. 8559 of *LNCS*, pp. 192–208. Springer, 2014.

14 T. Fiedor, L. Holík, O. Lengál, and T. Vojnar. Nested antichains for WS1S. In C. Baier and C. Tinelli, eds., *TACAS 2015*, LNCS. Springer, 2015. to appear.

15 N. Foster, D. Kozen, M. Milano, A. Silva, and L. Thompson. A coalgebraic decision procedure for NetKAT. In D. Walker, ed., *POPL 2015*, pp. 343–355. ACM, 2015.

16 T. Ganzow. *Definability and Model Checking: The Role of Orders and Compositionality*. PhD thesis, RWTH Aachen University, 2012.

17 T. Ganzow and L. Kaiser. New algorithm for weak monadic second-order logic on inductive structures. In A. Dawar and H. Veith, eds., *CSL 2010*, vol. 6247 of *LNCS*, pp. 366–380. Springer, 2010.

18 F. Haftmann and T. Nipkow. Code generation via higher-order rewrite systems. In M. Blume, N. Kobayashi, and G. Vidal, eds., *FLOPS 2010*, vol. 6009 of *LNCS*, pp. 103–117. Springer, 2010.

19 J. Hamza, B. Jobstmann, and V. Kuncak. Synthesis for regular specifications over unbounded domains. In R. Bloem and N. Sharygina, eds., *FMCAD 2010*, pp. 101–109. IEEE, 2010.

20 J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. MONA: Monadic second-order logic in practice. In E. Brinksma, R. Cleaveland, K. Larsen, T. Margaria, and B. Steffen, eds., *TACAS 1995*, vol. 1019 of *LNCS*, pp. 89–110. Springer, 1995.

21 B. Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In K. Futatsugi, J. Jouannaud, and J. Meseguer, eds., *Algebra, Meaning, and Computation*, vol. 4060 of *LNCS*, pp. 375–404. Springer, 2006.

22 J. L. Jensen, M. E. Jørgensen, N. Klarlund, and M. I. Schwartzbach. Automatic verification of pointer programs using monadic second-order logic. In M. C. Chen, R. K. Cytron, and A. M. Berman, eds., *PLDI 1997*, pp. 226–236. ACM, 1997.

23 N. Klarlund. Relativizations for the logic-automata connection. *Higher-Order and Symbolic Computation*, 18(1-2):79–120, 2005.

24 N. Klarlund and A. Møller. *MONA Version 1.4 User Manual.* BRICS, Department of Computer Science, Aarhus University, January 2001. Notes Series NS-01-1. Available from `http://www.brics.dk/mona/`. Revision of BRICS NS-98-3.

25 N. Klarlund, A. Møller, and M. I. Schwartzbach. MONA implementation secrets. *Int. J. Found. Comput. Sci.*, 13(4):571–586, 2002.

26 D. Kozen. On the coalgebraic theory of Kleene algebra with tests. Tech. report `http://hdl.handle.net/1813/10173`, Computing and Information Science, Cornell University, March 2008.

27 A. R. Meyer. Weak monadic second order theory of succesor is not elementary-recursive. In R. Parikh, ed., *Logic Colloquium*, vol. 453 of *LNM*, pp. 132–154. Springer, 1975.

28 M. Might, D. Darais, and D. Spiewak. Parsing with derivatives: A functional pearl. In M. M. T. Chakravarty, Z. Hu, and O. Danvy, eds., *ICFP 2011*, pp. 189–195. ACM, 2011.

29 T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, vol. 2283 of *LNCS*. Springer, 2002.

30 T. Nipkow and D. Traytel. Unified decision procedures for regular expression equivalence. In G. Klein and R. Gamboa, eds., *ITP 2014*, vol. 8558 of *LNCS*, pp. 450–466. Springer, 2014.

31 D. Pous. Symbolic algorithms for language equivalence and Kleene algebra with test. In D. Walker, ed., *POPL 2015*, pp. 357–368. ACM, 2015.

32 J. J. M. M. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorgi and R. de Simone, eds., *CONCUR 1998*, vol. 1466 of *LNCS*, pp. 194–218. Springer, 1998.

33 S. Shelah. The monadic theory of order. *Ann. Math.*, 102(3):379–419, 1975.

34 W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages*, pp. 389–455. Springer, 1997.

35 D. Traytel. [dWiNA Issue #1] Wrong results for simple formulas, 14 Jan 2015. Archived at `https://github.com/Raph-Stash/dWiNA/issues/1`.

36 D. Traytel. [Toss-devel] Toss deciding MSO, 14 Jan 2015. Archived at `http://sourceforge.net/p/toss/mailman/message/33232473/`.

37 D. Traytel. A codatatype of formal languages. In G. Klein, T. Nipkow, and L. Paulson, eds., *Archive of Formal Proofs*. `http://afp.sf.net/entries/Coinductive_Languages.shtml`, 2013.

38 D. Traytel. Supplementary material. `https://github.com/dtraytel/WS1S`, 2015.

39 D. Traytel and T. Nipkow. Verified decision procedures for MSO on words based on derivatives of regular expressions (extended version of [40]). `http://www21.in.tum.de/~traytel/mso.pdf`.

40 D. Traytel and T. Nipkow. Verified decision procedures for MSO on words based on derivatives of regular expressions. In G. Morrisett and T. Uustalu, eds., *ICFP 2013*, pp. 3–12. ACM, 2013.