Verified Decision Procedures for MSO on Words Based on Derivatives of Regular Expressions

Dmitriy Traytel Tobias Nipkow

Technische Universität München, Germany

traytel@in.tum.de www.in.tum.de/~nipkow

Abstract

Monadic second-order logic on finite words (MSO) is a decidable yet expressive logic into which many decision problems can be encoded. Since MSO formulas correspond to regular languages, equivalence of MSO formulas can be reduced to the equivalence of some regular structures (e.g. automata). This paper presents a verified functional decision procedure for MSO formulas that is not based on automata but on regular expressions. Functional languages are ideally suited for this task: regular expressions are data types and functions on them are defined by pattern matching and recursion and are verified by structural induction.

Decision procedures for regular expression equivalence have been formalized before, usually based on Brzozowski derivatives. Yet, for a straightforward embedding of MSO formulas into regular expressions an extension of regular expressions with a projection operation is required. We prove total correctness and completeness of an equivalence checker for regular expressions extended in that way. We also define a language-preserving translation of formulas into regular expressions with respect to two different semantics of MSO. Our results have been formalized and verified in the theorem prover Isabelle. Using Isabelle's code generation facility, this yields purely functional, formally verified programs that decide equivalence of MSO formulas.

General Terms Algorithms, Theory, Verification

Keywords MSO, WS1S, decision procedure, regular expressions, Brzozowski derivatives, interactive theorem proving, Isabelle

1. Introduction

Many decision procedures for logical theories are based on the famous logic-automaton connection. That is, they reduce the decision problem for some logical theory to a decidable question about some class of automata. Automata are usually implemented with the help of imperative data structures for efficiency reasons.

In functional languages, automata are not an ideal abstraction because they are graphs rather than trees. In contrast, regular expressions are perfect for functional languages and they are equally expressive. In fact, Brzozowski [8] showed how automata-based algorithms can be recast as recursive algebraic manipulations of regular expressions. His *derivatives* can be seen as a way of simulating automaton states with regular expressions and computing the next-state function symbolically.

Recently Brzozowski's derivatives were discovered by functional programmers and theorem provers. Owens *et al.* [23] realized that regular expressions and their derivatives fit perfectly with data types and recursive functions. Their paper explores regular expression matching based directly on regular expressions rather than automata. Fischer *et al.* [13] also explore regular expression matching, but by means of marked regular expressions rather than derivatives. Slightly later, the interactive theorem proving community woke up to the beauty of derivatives, too. This resulted in four papers about verified decision procedures for the equivalence of regular expressions based on derivatives and on marked regular expressions (see Related Work below). In one of these four papers, Coquand and Siles [10] state that "A more ambitious project will be to use this work for writing a decision procedure for WS1S", a monadic second-order logic. Our paper does just that (and more).

Monadic second-order logic on finite words (MSO) is a decidable yet expressive logic into which many decision problems can be encoded [26]. MSO allows only monadic predicates but quantification both over numbers and finite sets of numbers. Two closely related but subtly different semantics can be found in the literature. One of the two, WS1S—the Weak monadic Second-order logic of 1 Successor, is based on arithmetic. The other, M2L(Str) [16], is more closely related to formal languages. There seems to be some disagreement as to which semantics is the more appropriate one for verification purposes [3, 17]. Hence we cover both.

Essentially, MSO formulas describe regular languages. Therefore MSO formulas can be decided by translating them into automata. This is the basis of the highly successful MONA tool [12] for deciding WS1S. MONA's success is due to its (in practical terms) highly efficient implementation and to the ease with which very different verification problems can be encoded in monadic second-order logic, for example Presburger arithmetic and Hoare logic for pointer programs.

The contribution of this paper is the presentation of the first purely functional decision procedures for two interpretations of MSO based on derivatives of regular expressions. These decision procedures have been verified in Isabelle/HOL and we sketch their correctness proofs. We are not aware of any previous decision procedure for MSO based on regular expressions (as opposed to automata), let alone a verified program.

It is instructive to compare our decision procedure for WS1S with MONA. MONA is a highly tuned implementation using cache-conscious data structures including a BDD-based automaton representation. Ours is a (by comparison tiny) purely functional program that operates on regular expressions and can only cope with small examples. MONA is not verified (and the prospect of doing so is daunting), whereas our code is.

In this paper we distinguish ordinary regular expressions that contain only concatenation, union, and iteration from *extended* regular expressions that also provide complement and intersection.

The rest of the paper is organized as follows. Section 2 gives an overview of related work. Section 3 introduces some basic notations. Sections 4 and 5 constitute the main contribution of our paper—the first shows how to decide equivalence of extended regular expressions with an additional projection operation, the second reduces equivalence of MSO formulas to equivalence of exactly those regular expressions with respect to both semantics, M2L and WS1S. In total this yields a decision procedure for MSO on words. A short case study of the decision procedure is given in Section 6.

2. Related Work

Brzozowski [8] introduced derivatives of extended regular expressions. Antimirov [1] introduced partial derivatives of regular expressions. Caron *et al.* [9] extended Antimirov to extended regular expressions. The concept of derivatives as means to compute the next state symbolically goes beyond regular expressions—as witnessed by libraries for parsing developed by Danielsson [11] in Agda and by Might *et al.* [19] in Lisp using lazily evaluated variations of Brzozowski derivatives for parser combinators.

MONA was linked to Isabelle by Basin and Friedrich [5] and to PVS by Owre and Rueß [24]. In both cases, MONA is used as an oracle for deciding formulas in the respective theorem prover.

Now we discuss work on verified decision procedures for regular expressions. The first verified equivalence checker for regular expressions was published by Braibant and Pous [7]. They worked with automata, not regular expressions, their theory was large and their algorithm efficient. In response, Krauss and Nipkow [18] gave a much simpler partial correctness proof for an equivalence checker for regular expressions based on derivatives. Coquand and Siles [10] showed total correctness of their equivalence checker for extended regular expressions based on derivatives. Asperti [2] presented an equivalence checker for regular expressions via marked regular expressions (as previously used by [13]) and showed total correctness. Moreira *et al.* [20] presented an equivalence checker for regular expressions based on partial derivatives and showed its total correctness. Berghofer and Reiter [6] formalized a decision procedure for Presburger arithmetic via automata in Isabelle/HOL.

Outside of the application area of equivalence checking, Wu *et al.* [28] benefited from the inductive structure of regular expressions to formally verify the Myhill-Nerode theorem.

3. Preliminaries

Although we formalized everything in this paper in the theorem prover Isabelle/HOL [21, 22], no knowledge of theorem provers or Isabelle/HOL is required because we employ mostly ordinary mathematical notation in our presentation. Some specific notations are summarized below.

The symbol $\mathbb{B}$ represents the type of Booleans, where $\top$ and $\bot$ represent true and false. The type of sets and the type of lists over some type τ are written $\tau\,\mathsf{set}$ and $\tau\,\mathsf{list}$. In general, type constructors follow their arguments. The letters α and β represent type variables. The notation $t :: \tau$ means that term t has type τ.

Many of our functions are curried. In some cases we write the first argument as an index: instead of $f\ a\ b$ we write $f_a(b)$ (in preference to just $f_a\ b$). The projection functions on pairs are called fst and snd.

The image of a function f over a set S is written $f \bullet S$.

Lists are built up from the empty list $[\,]$ via the infix # operator that prepends an element x to a list xs: $x \# xs$. Two lists are concatenated with the infix @ operator. Accessing the nth element of a list xs is denoted by $xs[n]$; the indexing is zero-based. The length of the list xs is written $|xs|$.

Finite words as in formal language theory are modelled as finite lists, i.e. type $\alpha\,\mathsf{list}$. The empty word is the empty list. As is customary, concatenation of two words u and v is denoted by their juxtaposition uv; similarly for a single letter a of the alphabet and a word w: aw. That is, the operators # and @ remain implicit (for words, not for arbitrary lists).

4. Extended Regular Expressions

In Section 5, MSO formulas are translated into regular expressions such that encodings of models of a formula correspond exactly to words in the regular language. Thereby, equivalence of formulas is reduced to the equivalence of regular expressions.

Decision procedures for equivalence of regular expression have been formalized earlier in theorem provers. Here, we extend the existing formalization and the soundness proof in Isabelle/HOL by Krauss and Nipkow [18] with negation and intersection operation on regular expressions, as well as with a nonstandard projection operation. Additionally, we provide proofs of termination and completeness.

4.1 Syntax and Semantics

Regular expressions extended with intersection and complement allow us to encode Boolean operators on formulas in a straightforward fashion. A further operation—the *projection* Π—plays the crucial role of encoding existential quantifiers. These Π-*extended* regular expressions (to distinguish them from mere extended regular expressions) are defined as a recursive data type $\alpha\,\mathsf{RE}$, where α is the type of the underlying alphabet. In conventional concrete syntax, $\alpha\,\mathsf{RE}$ is defined by the grammar

$$
\begin{array}{rcccccc}
r & = & \mathbf{0} & | & \mathbf{1} & | & a \\
 & | & r+s & | & r\cdot s & | & r^* \\
 & | & r\cap s & | & \neg r & | & \Pi\, r
\end{array}
$$

where $r, s :: \alpha\,\mathsf{RE}$ and $a :: \alpha$. Note that much of the time we will omit the "Π-extended" and simply speak of regular expressions if there is no danger of confusion.

We assume that type α is partitioned into a family of alphabets Σ_n that depend on a natural number n. In our application, n will represent the number of free variables of the translated MSO formula. For now Σ_n is just a parameter of our setup.

We focus on wellformed regular expressions where all atoms come from the same alphabet Σ_n. This will guarantee that the language of such a wellformed expression is a subset of Σ_n^*. The projection operation complicates wellformedness a little. Because projection is meant to encode existential quantifiers, projection should transform a regular expression over Σ_{n+1} into a regular expression over Σ_n, just as the existential quantifier transforms a formula with $n+1$ free variables into a formula with n free variables. Thus projection changes the alphabet.

Wellformedness is defined as the recursive predicate $\mathsf{wf} :: \mathbb{N} \to \alpha\,\mathsf{RE} \to \mathbb{B}$.

$$
\begin{array}{llll}
\mathsf{wf}_n(\mathbf{0}) & = \top & \mathsf{wf}_n(\mathbf{1}) & = \top \\
\mathsf{wf}_n(a) & = a \in \Sigma_n & \mathsf{wf}_n(r+s) & = \mathsf{wf}_n(r) \wedge \mathsf{wf}_n(s) \\
\mathsf{wf}_n(r\cdot s) & = \mathsf{wf}_n(r) \wedge \mathsf{wf}_n(s) & \mathsf{wf}_n(r^*) & = \mathsf{wf}_n(r) \\
\mathsf{wf}_n(r\cap s) & = \mathsf{wf}_n(r) \wedge \mathsf{wf}_n(s) & \mathsf{wf}_n(\neg r) & = \mathsf{wf}_n(r) \\
\mathsf{wf}_n(\Pi\, r) & = \mathsf{wf}_{n+1}(r) & &
\end{array}
$$

We call a regular expression r *n-wellformed* if $\mathsf{wf}_n(r)$ holds.

The *language* $\mathcal{L} :: \mathbb{N} \to \alpha\,\mathsf{RE} \to (\alpha\,\mathsf{list})\,\mathsf{set}$ of a regular expression is defined as usual, except for the equations for complement and projection. For an n-wellformed regular expression the definition yields a subset of Σ_n^*.

$$\begin{array}{llll}
\mathcal{L}_n(\mathbf{0}) & = \{\} & \mathcal{L}_n(\mathbf{1}) & = \{[\,]\} \\
\mathcal{L}_n(a) & = \{a\} & \mathcal{L}_n(r+s) & = \mathcal{L}_n(r) \cup \mathcal{L}_n(s) \\
\mathcal{L}_n(r \cdot s) & = \mathcal{L}_n(r) \cdot \mathcal{L}_n(s) & \mathcal{L}_n(r^*) & = \mathcal{L}_n(r)^* \\
\mathcal{L}_n(r \cap s) & = \mathcal{L}_n(r) \cap \mathcal{L}_n(s) & \mathcal{L}_n(\neg r) & = \Sigma_n^* \setminus \mathcal{L}_n(r) \\
\mathcal{L}_n(\Pi\, r) & = (\mathsf{map}\, \pi) \bullet \mathcal{L}_{n+1}(r) & &
\end{array}$$

The first unusual point is the parametrization with n. It expresses that we expect a regular expression over Σ_n and is necessary for the definition $\mathcal{L}_n(\neg r) = \Sigma_n^* \setminus \mathcal{L}_n(r)$.

The definition $\mathcal{L}_n(\Pi\, r) = \mathsf{map}\, \pi \bullet \mathcal{L}_{n+1}(r)$ is parameterized by a function $\pi :: \Sigma_{n+1} \to \Sigma_n$[1]. The projection Π denotes the homomorphic image under this fixed π. In more detail: map lifts π homomorphically to words (lists), and $\bullet$ lifts it to sets of words. Therefore Π transforms a language over Σ_{n+1} into a language over Σ_n.

To understand the "projection" terminology, it is helpful to think of elements of Σ_n as lists of fixed length n over some alphabet Σ and of π as the tail function on lists that drops the first element of the list. A word over Σ_n is then a list of lists. Though this is a good intuition, the actual encoding of formulas later on will be slightly more complicated. Fortunately we can ignore these complications for now by working with arbitrary but fixed Σ_n and π in the current section. Specific instantiations for them are given in Section 5.

4.2 Deciding Language Equivalence

Now we turn our attention to deciding equivalence of Π-extended regular expressions. The key concepts required for this are finality and derivatives. We call a regular expression *final* if its language contains the empty word $[\,]$. Finality can be easily checked syntactically by the following recursive function $\varepsilon :: \alpha\, \mathsf{RE} \to \mathbb{B}$.

$$\begin{array}{llll}
\varepsilon(\mathbf{0}) & = \bot & \varepsilon(\mathbf{1}) & = \top \\
\varepsilon(a) & = \bot & \varepsilon(r+s) & = \varepsilon(r) \vee \varepsilon(s) \\
\varepsilon(r \cdot s) & = \varepsilon(r) \wedge \varepsilon(s) & \varepsilon(r^*) & = \top \\
\varepsilon(r \cap s) & = \varepsilon(r) \wedge \varepsilon(s) & \varepsilon(\neg r) & = \neg \varepsilon(r) \\
\varepsilon(\Pi\, r) & = \varepsilon(r) & &
\end{array}$$

The characteristic property—$\varepsilon(r)$ iff $[\,] \in \mathcal{L}_n(r)$ for any regular expression r and $n \in \mathbb{N}$—follows by structural induction on r.

The second key concept—the *derivative* of a regular expression $\mathcal{D} :: \alpha \to \alpha\, \mathsf{RE} \to \alpha\, \mathsf{RE}$ and its lifting to words $\mathcal{D}^* :: \alpha\, \mathsf{list} \to \alpha\, \mathsf{RE} \to \alpha\, \mathsf{RE}$—semantically corresponds to left quotients of regular languages with respect to a fixed letter or word. Just as before, the recursive definition is purely syntactic and the semantic correspondence is established by induction.

$$\begin{array}{llll}
\mathcal{D}_b(\mathbf{0}) & = \mathbf{0} & \mathcal{D}_b(\mathbf{1}) & = \mathbf{0} \\
\mathcal{D}_b(a) & = \textbf{if } a = b \textbf{ then } \mathbf{1} \textbf{ else } \mathbf{0} & \mathcal{D}_b(r+s) & = \mathcal{D}_b(r) + \mathcal{D}_b(s) \\
\mathcal{D}_b(r \cdot s) & = & \mathcal{D}_b(r^*) & = \mathcal{D}_b(r) \cdot r^* \\
\multicolumn{4}{l}{\quad \textbf{if } \varepsilon(r) \textbf{ then } \mathcal{D}_b(r) \cdot s + \mathcal{D}_b(s)} \\
\multicolumn{4}{l}{\quad \textbf{else } \mathcal{D}_b(r) \cdot s} \\
\mathcal{D}_b(r \cap s) & = \mathcal{D}_b(r) \cap \mathcal{D}_b(s) & \mathcal{D}_b(\neg r) & = \neg \mathcal{D}_b(r) \\
\mathcal{D}_b(\Pi\, r) & = \Pi \left(\bigoplus_{c \in \pi^- b} \mathcal{D}_c(r) \right) & & \\
\mathcal{D}^*_{[\,]}(r) & = r & \mathcal{D}^*_{bw}(r) & = \mathcal{D}^*_w(\mathcal{D}_b(r))
\end{array}$$

Lemma 1. *Assume $b \in \Sigma_n$, $v \in \Sigma_n^*$ and let r be an n-wellformed regular expression. Then $\mathcal{L}_n(\mathcal{D}_b(r)) = \{w \mid bw \in \mathcal{L}_n(r)\}$ and $\mathsf{wf}_n(\mathcal{D}_b(r))$, and consequently $\mathcal{L}_n(\mathcal{D}^*_v(r)) = \{w \mid vw \in \mathcal{L}_n(r)\}$ and $\mathsf{wf}_n(\mathcal{D}^*_v(r))$.*

[1] Due to Isabelle's lack of dependent types the actual type of π is $\alpha \to \alpha$. The more refined dependent type $\Sigma_{n+1} \to \Sigma_n$ is realized via Isabelle's tool for modeling parameterized systems with additional assumptions: locales [4]. A locale fixes parameters and states assumptions about them. Hence, we use the locale assumption $\pi \bullet \Sigma_{n+1} \subseteq \Sigma_n$ to relate locale parameters π and Σ.

The projection case introduced some new syntax that deserves some explanation. The preimage π^- applied to a letter $b \in \Sigma_n$ denotes the set $\{c \in \Sigma_{n+1} \mid \pi\, c = b\}$. Our alphabets Σ_n are finite for each n, hence so is the preimage of a letter. The summation $\bigoplus$ over a finite set denotes the iterated application of the $+$-constructor of regular expressions. The summation over the empty set is defined as $\mathbf{0}$.

Derivatives of extended regular expressions were introduced by Brzozowski [8] almost fifty years ago. Our contribution is the extension of the concept to handle the projection operation. Since the projection acts homomorphically on words, it is clear that the derivative of $\Pi\, r$ with respect to a letter b can be expressed as a projection of derivatives of r. The concrete definition is a consequence of the following identity of left quotients for $b \in \Sigma_n$ and $A \subseteq \Sigma_{n+1}^*$:

$$\{w \mid bw \in \mathsf{map}\, \pi \bullet A\} = \mathsf{map}\, \pi \bullet \bigcup_{c \in \pi^- b} \{w \mid cw \in A\}$$

Although we completely avoid automata in the formalization, a derivative with respect to the letter b can be seen as a transition labelled by b in a deterministic automaton, the states of which are labelled by regular expressions. The automaton accepting the language of a regular expression r can be thus constructed iteratively by exploring all derivatives of r and defining exactly those states as accepting, which are labelled by a final regular expression. However, the set $\{\mathcal{D}^*_w(r) \mid w :: \alpha\, \mathsf{list}\}$ of states reachable in this manner is infinite in general. To obtain a finite automaton, the states must be partitioned into classes of regular expressions that are *ACI-equivalent*, i.e. syntactically equal modulo associativity, commutativity and idempotence of the $+$-constructor. Brzozowski showed that the number of such classes for a fixed regular expression r is finite by structural induction on r. The inductive steps require proving finiteness by representing equivalence classes of derivatives of the expression in terms of equivalence classes of derivatives of subexpressions. This is technically complicated, especially for concatenation, iteration and projection, since it requires a careful choice representatives of equivalence classes to reason about them, and Isabelle's automation can not help much with the finiteness arguments—indeed the verification of Theorem 2 constitutes the most intricate proof in the present work.

Theorem 2. *$\{\langle\!\langle \mathcal{D}^*_w(r) \rangle\!\rangle \mid w :: \alpha\, \mathsf{list}\}$ is finite for any regular expression r.*

The function $\langle\!\langle - \rangle\!\rangle :: \alpha\, \mathsf{RE} \to \alpha\, \mathsf{RE}$ is the ACI normalization function, which maps ACI-equivalent regular expressions to the same representative. It is defined by means of a normalizing constructor $\oplus :: \alpha\, \mathsf{RE} \to \alpha\, \mathsf{RE} \to \alpha\, \mathsf{RE}$ and an arbitrary linear order $\leq$ on regular expressions.

$$\begin{array}{llll}
\langle\!\langle \mathbf{0} \rangle\!\rangle & = \mathbf{0} & \langle\!\langle \mathbf{1} \rangle\!\rangle & = \mathbf{1} \\
\langle\!\langle a \rangle\!\rangle & = a & \langle\!\langle r+s \rangle\!\rangle & = \langle\!\langle r \rangle\!\rangle \oplus \langle\!\langle s \rangle\!\rangle \\
\langle\!\langle r \cdot s \rangle\!\rangle & = \langle\!\langle r \rangle\!\rangle \cdot \langle\!\langle s \rangle\!\rangle & \langle\!\langle r^* \rangle\!\rangle & = \langle\!\langle r \rangle\!\rangle^* \\
\langle\!\langle r \cap s \rangle\!\rangle & = \langle\!\langle r \rangle\!\rangle \cap \langle\!\langle s \rangle\!\rangle & \langle\!\langle \neg r \rangle\!\rangle & = \neg \langle\!\langle r \rangle\!\rangle \\
\langle\!\langle \Pi\, r \rangle\!\rangle & = \Pi \langle\!\langle r \rangle\!\rangle & &
\end{array}$$

$$\begin{array}{lcl}
(r+s) \oplus t & = & r \oplus (s \oplus t) \\
r \oplus (s+t) & = & \textbf{if } r = s \textbf{ then } s + t \\
 & & \textbf{else if } r \leq s \textbf{ then } r + (s+t) \\
 & & \quad \textbf{else } s + (r \oplus t) \\
r \oplus s & = & \textbf{if } r = s \textbf{ then } r \\
 & & \textbf{else if } r \leq s \textbf{ then } r + s \\
 & & \quad \textbf{else } s + r
\end{array}$$

The equations for $\oplus$ are matched sequentially.

After the application of $\langle\!\langle - \rangle\!\rangle$ all sums in the expression are associated to the right and the summands are sorted with respect to $\le$ and duplicated summands are removed. From this, further later on useful properties of $\langle\!\langle - \rangle\!\rangle$ can be derived:

Lemma 3. *Let r be a regular expression, $n \in \mathbb{N}$ and $b \in \Sigma_n$. Then $\mathcal{L}_n \langle\!\langle r \rangle\!\rangle = \mathcal{L}_n(r)$, $\langle\!\langle \langle\!\langle r \rangle\!\rangle \rangle\!\rangle = \langle\!\langle r \rangle\!\rangle$ and $\langle\!\langle \mathcal{D}_b \langle\!\langle r \rangle\!\rangle \rangle\!\rangle = \langle\!\langle \mathcal{D}_b(r) \rangle\!\rangle$.*

So far, ACI normalization only connects Brzozowski derivatives to deterministic finite automata. Furthermore, it will ensure termination of our decision procedure even without ever entering the world of automata. Instead we follow Rutten [25], who gives an alternative view on deterministic automata as coalgebras. In the coalgebraic setting the function $\lambda r.\ (\varepsilon(r), \lambda b.\ \mathcal{D}_b(r)) :: \alpha\,\mathsf{RE} \to \mathbb{B} \times (\alpha \to \alpha\,\mathsf{RE})$ is a D-coalgebra for the functor $D(S) = \mathbb{B} \times (\alpha \to S)$. The final coalgebra of D exists and corresponds exactly to the set of all languages. Therefore, we obtain the powerful coinduction principle, reducing language equality to bisimilarity. We phrase this general theorem instantiated to our concrete setting. The formalized proof itself does not require any category theory; it resembles the reasoning in Rutten [25, §4].

Theorem 4 (Coinduction)**.** *Let $R :: (\alpha\,\mathsf{RE} \times \alpha\,\mathsf{RE})\,\mathsf{set}$ be a relation, such that for all $(r, s) \in R$ we have:*

1. $\mathsf{wf}_n(r) \wedge \mathsf{wf}_n(s)$;
2. $\varepsilon(r) \leftrightarrow \varepsilon(s)$;
3. $(\langle\!\langle \mathcal{D}_b(r) \rangle\!\rangle, \langle\!\langle \mathcal{D}_b(s) \rangle\!\rangle) \in R$ *for all* $b \in \Sigma_n$.

Then for all $(r, s) \in R$, $\mathcal{L}_n(r) = \mathcal{L}_n(s)$ holds.

From Lemma 1 and Lemma 3, we know that the relation

$$\mathcal{B} = \{(\langle\!\langle \mathcal{D}^*_w(r) \rangle\!\rangle, \langle\!\langle \mathcal{D}^*_w(s) \rangle\!\rangle) \mid w \in \Sigma^*_n\}$$

contains $(\langle\!\langle r \rangle\!\rangle, \langle\!\langle s \rangle\!\rangle)$ and fulfils the assumptions 1 and 3 of the coinduction theorem, assuming that r and s are both n-wellformed. Moreover, using Theorem 2 it follows that this relation is finite. Thus, checking assumption 2 for every pair of this finite relation is sufficient to prove language equality of r and s by coinduction. We obtain the following abstract specification of a language equivalence checking algorithm.

Theorem 5. *Let r and s be n-wellformed regular expressions. Then $\mathcal{L}_n(r) = \mathcal{L}_n(s)$ iff we have $\varepsilon(r') \leftrightarrow \varepsilon(s')$ for all $(r', s') \in \mathcal{B}$.*

4.3 Executable Algorithm from a Theorem

Our goal is not only to prove some abstract theorems about a decision procedure, but also to extract executable code in some functional programming language (e.g. Standard ML, Haskell, OCaml) using the code generation facility of Isabelle/HOL [15]. Theorem 5 is not enough to do so: it contains a set comprehension ranging over the infinite set Σ^*_n, which is not executable as such. We need to instruct the system how to enumerate $\mathcal{B}$.

We start with the pair $(\langle\!\langle r \rangle\!\rangle, \langle\!\langle s \rangle\!\rangle)$ and compute its pairwise derivatives for all letters of the alphabet. For the computed pairs of regular expressions we proceed by computing their derivatives and so on. This of course does not terminate. However, if we stop our exploration at pairs that we have seen before it does, since we are exploring a finite set.

In more detail, we use a worklist algorithm that iteratively adds not yet inspected pairs of regular expressions while exhausting words of increasing length until no new pairs are generated. Saturation is reached by means of the executable combinator $\mathsf{while} :: (\alpha \to \mathbb{B}) \to (\alpha \to \alpha) \to \alpha \to \alpha\,\mathsf{option}$ from the Isabelle/HOL library. The option type $\alpha\,\mathsf{option}$ has two constructors $\mathsf{None} :: \alpha\,\mathsf{option}$ and $\mathsf{Some} :: \alpha \to \alpha\,\mathsf{option}$. Some lifts elements from the base type α to the option type, while None is usually used to indicate some exceptional behaviour. The definition of while

$$\mathsf{while}\ b\ c\ s = \textbf{if}\ \exists k. \neg b(c^k(s))\ \textbf{then}\ \mathsf{Some}\ (c^{\mathsf{Least}\ k. \neg b(c^k(s))}(s))\ \textbf{else}\ \mathsf{None}$$

is not executable, but the following key lemma is:

$$\mathsf{while}\ b\ c\ s\ =\ \textbf{if}\ b\ s\ \textbf{then}\ \mathsf{while}\ b\ c\ (c\ s)\ \textbf{else}\ \mathsf{Some}\ s$$

The code generated from this recursive equation will return $\mathsf{Some}\ s$ in case the definition of while says so, but instead of returning None, it will not terminate. Thus we can prove termination if we can show that the result is $\neq \mathsf{None}$.

In our algorithm, the state s of the while loop consists of a worklist $ws :: (\alpha\,\mathsf{RE} \times \alpha\,\mathsf{RE})\,\mathsf{list}$ of unprocessed pairs of regular expressions together with a set $N :: (\gamma \times \gamma)\,\mathsf{set}$ of already seen pairs modulo a normalization function $\mathsf{norm} :: \alpha\,\mathsf{RE} \to \gamma$. This normalization function (which is a parameter of our setup) is applied to already ACI-normalized expressions, to syntactically identify further language equivalent expressions. This makes the bisimulation relation that must be exhausted smaller, thus saturation is reached faster. The range type of the normalization is not fixed, but we require a notion of languages $\mathcal{L}^\gamma :: \mathbb{N} \to \gamma \to (\alpha\,\mathsf{list})\,\mathsf{set}$ to be available for it, such that $\mathcal{L}^\gamma_n(\mathsf{norm}\ r) = \mathcal{L}_n(r)$ holds. In the simplest case norm can be the identity function and $\mathcal{L}^\gamma = \mathcal{L}$. More interesting is a function on regular expressions that eliminates $\mathbf{0}$ from unions, concatenations and intersections and $\mathbf{1}$ from concatenations. Not fixing the range type allows us to use different regular structures such as automata or different types of regular expressions, on which further simplifications might be easier.

Finally, the functions $\mathsf{b} :: (\alpha\,\mathsf{RE} \times \alpha\,\mathsf{RE})\,\mathsf{list} \times (\gamma \times \gamma)\,\mathsf{set} \to \mathbb{B}$ and $\mathsf{c} :: \mathbb{N} \to (\alpha\,\mathsf{RE} \times \alpha\,\mathsf{RE})\,\mathsf{list} \times (\gamma \times \gamma)\,\mathsf{set} \to (\alpha\,\mathsf{RE} \times \alpha\,\mathsf{RE})\,\mathsf{list} \times (\gamma \times \gamma)\,\mathsf{set}$ defined below are given as arguments to while. A wellformedness check completes the now executable algorithm $\mathsf{eqv}^{\mathsf{RE}} :: \mathbb{N} \to \alpha\,\mathsf{RE} \to \alpha\,\mathsf{RE} \to \mathbb{B}$.

$$
\begin{aligned}
&\mathsf{b}\ ([\,], \_) && = \bot \\
&\mathsf{b}\ ((r, s) \# \_, \_) && = \varepsilon(r) \leftrightarrow \varepsilon(s)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{c}_n\ ((r, s) \# ws, N)\ = \\
&\quad \textbf{let} \\
&\qquad succs = \mathsf{map}\ (\lambda b. \\
&\qquad\qquad \textbf{let} \\
&\qquad\qquad\quad r' = \langle\!\langle \mathcal{D}_b(r) \rangle\!\rangle \\
&\qquad\qquad\quad s' = \langle\!\langle \mathcal{D}_b(s) \rangle\!\rangle \\
&\qquad\qquad \textbf{in}\ ((r', s'), (\mathsf{norm}\ r', \mathsf{norm}\ s')))\ \Sigma_n \\
&\qquad new = \mathsf{remdups}\ \mathsf{snd}\ (\mathsf{filter}\ (\lambda(\_, rs).\ rs \notin N)\ succs) \\
&\quad \textbf{in}\ (ws\ @\ \mathsf{map}\ \mathsf{fst}\ new, \mathsf{set}\ (\mathsf{map}\ \mathsf{snd}\ new) \cup N)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{eqv}^{\mathsf{RE}}_n\ r\ s\ = \\
&\quad \mathsf{wf}_n(r) \wedge \mathsf{wf}_n(s) \wedge \\
&\quad (\textbf{case}\ \mathsf{while}\ \mathsf{b}\ \mathsf{c}_n\ ([(\langle\!\langle r \rangle\!\rangle, \langle\!\langle s \rangle\!\rangle)], \{(\mathsf{norm} \langle\!\langle r \rangle\!\rangle, \mathsf{norm} \langle\!\langle s \rangle\!\rangle)\})\ \textbf{of} \\
&\qquad\quad ([\,], \_) \Rightarrow \top \\
&\quad \mid\ \ (\_ \# \_, \_) \Rightarrow \bot)
\end{aligned}
$$

The function $\mathsf{set} :: \alpha\,\mathsf{list} \to \alpha\,\mathsf{set}$ maps a list to the set of its elements, $\mathsf{filter} :: (\alpha \to \mathbb{B}) \to \alpha\,\mathsf{list} \to \alpha\,\mathsf{list}$ removes elements that do not fulfil the given predicate, while $\mathsf{remdups} :: (\alpha \to \beta) \to \alpha\,\mathsf{list} \to \alpha\,\mathsf{list}$ is used to keep the worklist as small as possible. $\mathsf{remdups}\ f\ xs$ removes duplicates from xs modulo the function f, e.g. $\mathsf{remdups}\ \mathsf{snd}\ [(\mathbf{0}, \mathbf{0}), (\mathbf{1}, \mathbf{0})] = [(\mathbf{1}, \mathbf{0})]$ (which element is actually kept is irrelevant; the result $[(\mathbf{0}, \mathbf{0})]$ would also be valid).

The termination of $\mathsf{eqv}^{\mathsf{RE}}$ for any input is guaranteed by two facts: (1) all recursively defined functions in Isabelle/HOL terminate by their definitional principle (either primitive or wellfounded recursion) and (2) the termination of while follows from Theorem 2 and the fact that the set N of already seen pairs in the state is a subset of $\mathsf{norm} \bullet \{(\langle\!\langle \mathcal{D}^*_w(r) \rangle\!\rangle, \langle\!\langle \mathcal{D}^*_w(s) \rangle\!\rangle) \mid w \in \Sigma^*_n\}$.

Theorem 6 (Termination)**.** *Let r and s be n-wellformed regular expressions. Then*

$$\mathsf{while}\ \mathsf{b}\ \mathsf{c}_n\,([(\langle\!\langle r\rangle\!\rangle, \langle\!\langle s\rangle\!\rangle)], \{(\mathsf{norm}\langle\!\langle r\rangle\!\rangle, \mathsf{norm}\langle\!\langle s\rangle\!\rangle)\}) \neq \mathsf{None}.$$

Function $\mathsf{eqv}^{\mathsf{RE}}$ deserves the name decision procedure since it constitutes the refinement of the algorithm abstractly stated in Theorem 5, and is therefore sound and complete.

Theorem 7 (Soundness)**.** *Let r and s be regular expressions such that $\mathsf{eqv}_n^{\mathsf{RE}}\ r\ s$. Then $\mathcal{L}_n(r) = \mathcal{L}_n(s)$.*

Theorem 8 (Completeness)**.** *Let r and s be n-wellformed regular expressions such that $\mathcal{L}_n(r) = \mathcal{L}_n(s)$. Then $\mathsf{eqv}_n^{\mathsf{RE}}\ r\ s$.*

Let us observe the decision procedure at work by looking at the regular expressions a^* and $\mathbf{1} + a \cdot a^*$ for some $a \in \Sigma_n = \{a, b\}$ for some n. For presentation purposes, the correspondence of derivatives to automata is useful. Figure 1 shows two automata, the states of which are equivalence classes of pairs of regular expressions indicated by a dashed fringe (which is omitted for singleton classes). The equivalence classes of automaton (a) are modulo plain ACI normalization, while those of automaton (b) are modulo a stronger normalization function, making the automaton smaller. Transitions correspond to pairwise derivatives and doubled margins denote states for which the associated pairs of regular expressions are pairwise final. Both automata are the result of our decision procedure performing a breadth-first exploration starting with the initially given pair and ignoring states that are in the equivalence class of already visited states. The absence of pairs (r, s) for which r is final and s is not final (or vice versa) proves the equivalence of all pairs in the automaton, including the pair $(a^*, \mathbf{1} + a \cdot a^*)$.

5. MSO on Finite Words

Logics on finite words consider formulas in the context of a formal word, with variables representing positions in the word. In the first-order logic on words a variable always denotes a single position, while in monadic second-order logic (MSO) variables come in two flavours: first-order variables for single positions and second-order variables for finite sets of positions.

In the next subsections we first define the syntax of formulas and give them a semantics that is related to formal languages: M2L(Str). The second semantics, WS1S, is then introduced as a relaxation of M2L (we drop the "(Str)" from now on). Both semantics are equally expressive and deciding both is of nonelementary complexity. The benefits and drawbacks of the two semantics are discussed elsewhere [3, 17].

5.1 Syntax and M2L Semantics

MSO formulas are syntactically represented by the recursive data type $\alpha\,\Phi$ using de Bruijn indices for variable bindings. Terms of $\alpha\,\Phi$ are generated by the grammar

$$\begin{array}{rcccccc} \varphi & = & \mathsf{Q}\,a\,m & \mid & m_1 < m_2 & \mid & m \in M \\ & \mid & \varphi \wedge \psi & \mid & \varphi \vee \psi & \mid & \neg\varphi \\ & \mid & \exists\,\varphi & \mid & \boldsymbol{\exists}\,\varphi & & \end{array}$$

where $\varphi, \psi :: \alpha\,\Phi$, $m, m_1, m_2, M \in \mathbb{N}$ and $a \in \alpha$. Lower-case variables m, m_1, m_2 denote first-order variables, M denotes a second order variable. The atomic formula $\mathsf{Q}\,a\,m$ requires the letter of the word at the position represented by variable m to be a; the constructors $<$ and $\in$ compare positions; Boolean operators are interpreted as usual.

The bold existential quantifier $\boldsymbol{\exists}$ binds second-order variables, $\exists$ binds first-order variables. Occurrences of bound variables represented as de Bruijn indices refer to their binders by counting the number of nested existential quantifier between the binder and the occurrence. For example, the formula $\exists\,(\mathsf{Q}\,a\,0 \wedge (\boldsymbol{\exists}\,1 \in 0))$ corresponds to $\exists x. (\mathsf{Q}\,a\,x \wedge (\exists X.\ x \in X))$ when using names. The first 0 in the nameless formula refers to the outermost first-order quantifier. Inside of the inner second-order quantifier, index 1 refers to the outermost quantifier and index 0 to the inner quantifier. The nameless representation simplifies reasoning by implicitly capturing α-equivalence of formulas. On the downside, de Bruijn indices are less readable and must be manipulated with care.

Formulas may have free variables. The functions $\mathcal{V}_1 :: \alpha\,\Phi \to \mathbb{N}\,\mathsf{set}$ and $\mathcal{V}_2 :: \alpha\,\Phi \to \mathbb{N}\,\mathsf{set}$ collect the free first-order and second-order variables:

$$\begin{array}{ll@{\quad}ll} \mathcal{V}_1(\mathsf{Q}\,a\,m) & = \{m\} & \mathcal{V}_2(\mathsf{Q}\,a\,m) & = \{\} \\ \mathcal{V}_1(m_1 < m_2) & = \{m_1, m_2\} & \mathcal{V}_2(m_1 < m_2) & = \{\} \\ \mathcal{V}_1(m \in M) & = \{m\} & \mathcal{V}_2(m \in M) & = \{M\} \\ \mathcal{V}_1(\varphi \wedge \psi) & = \mathcal{V}_1(\varphi) \cup \mathcal{V}_1(\psi) & \mathcal{V}_2(\varphi \wedge \psi) & = \mathcal{V}_2(\varphi) \cup \mathcal{V}_2(\psi) \\ \mathcal{V}_1(\varphi \vee \psi) & = \mathcal{V}_1(\varphi) \cup \mathcal{V}_1(\psi) & \mathcal{V}_2(\varphi \vee \psi) & = \mathcal{V}_2(\varphi) \cup \mathcal{V}_2(\psi) \\ \mathcal{V}_1(\neg\varphi) & = \mathcal{V}_1(\varphi) & \mathcal{V}_2(\neg\varphi) & = \mathcal{V}_2(\varphi) \\ \mathcal{V}_1(\exists\,\varphi) & = \lfloor \mathcal{V}_1(\varphi) \setminus \{0\} \rfloor & \mathcal{V}_2(\exists\,\varphi) & = \lfloor \mathcal{V}_2(\varphi) \rfloor \\ \mathcal{V}_1(\boldsymbol{\exists}\,\varphi) & = \lfloor \mathcal{V}_1(\varphi) \rfloor & \mathcal{V}_2(\boldsymbol{\exists}\,\varphi) & = \lfloor \mathcal{V}_2(\varphi) \setminus \{0\} \rfloor \end{array}$$

The notation $\lfloor X \rfloor$ is shorthand for $(\lambda x.\ x - 1) \bullet X$, which reverts the increasing effect of an existential quantifier on previously bound or free variables. To obtain only free variables, bound variables are removed when their quantifier is processed, at which point the bound variable has index 0.

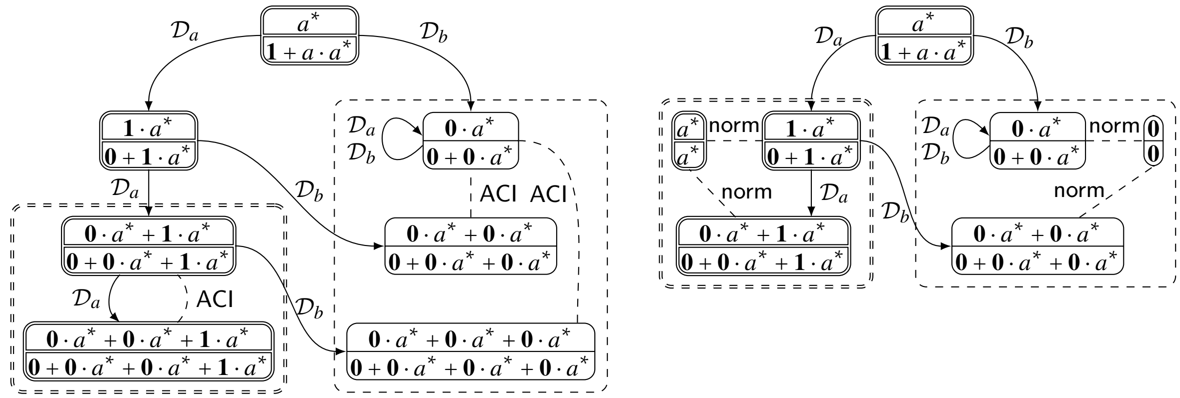
Just as for Π-extended regular expressions, not all formulas in $\alpha\,\Phi$ are meaningful. Consider $0 \in 0$, where 0 is both a first-order and a second-order variable. To exclude such formulas, we define the predicate $\mathsf{wf}^{\Phi} :: \mathbb{N} \to \alpha\,\Phi \to \mathbb{B}$ as $\mathsf{wf}_n^{\Phi}(\varphi) = (\mathcal{V}_1(\varphi) \cap \mathcal{V}_2(\varphi) = \{\}) \wedge \mathsf{pre\_wf}_n^{\Phi}(\varphi)$ and call a formula φ *n-wellformed* if $\mathsf{wf}_n^{\Phi}(\varphi)$ holds. The recursively defined predicate $\mathsf{pre\_wf}^{\Phi} :: \mathbb{N} \to \alpha\,\Phi \to \mathbb{B}$ is used for further assumptions on the structure of n-wellformed formulas, which will simplify our proofs:

$$\begin{array}{ll} \mathsf{pre\_wf}_n^{\Phi}(\mathsf{Q}\,a\,m) & = a \in \Sigma \wedge m < n \\ \mathsf{pre\_wf}_n^{\Phi}(m_1 < m_2) & = m_1 < n \wedge m_2 < n \\ \mathsf{pre\_wf}_n^{\Phi}(m \in M) & = m < n \wedge M < n \\ \mathsf{pre\_wf}_n^{\Phi}(\varphi \wedge \psi) & = \mathsf{pre\_wf}_n^{\Phi}(\varphi) \wedge \mathsf{pre\_wf}_n^{\Phi}(\psi) \\ \mathsf{pre\_wf}_n^{\Phi}(\varphi \vee \psi) & = \mathsf{pre\_wf}_n^{\Phi}(\varphi) \wedge \mathsf{pre\_wf}_n^{\Phi}(\psi) \\ \mathsf{pre\_wf}_n^{\Phi}(\neg\varphi) & = \mathsf{pre\_wf}_n^{\Phi}(\varphi) \\ \mathsf{pre\_wf}_n^{\Phi}(\exists\,\varphi) & = \mathsf{pre\_wf}_{n+1}^{\Phi}(\varphi) \wedge 0 \in \mathcal{V}_1(\varphi) \wedge 0 \notin \mathcal{V}_2(\varphi) \\ \mathsf{pre\_wf}_n^{\Phi}(\boldsymbol{\exists}\,\varphi) & = \mathsf{pre\_wf}_{n+1}^{\Phi}(\varphi) \wedge 0 \notin \mathcal{V}_1(\varphi) \wedge 0 \in \mathcal{V}_2(\varphi) \end{array}$$

$\mathsf{pre\_wf}_n^{\Phi}(\varphi)$ ensures that the index of every free variable in φ is below n and the values of type α come from a fixed alphabet Σ. Note that Σ is really just a fixed set of letters of type α, independent of any n and is a parameter of our setup. Moreover, $\mathsf{pre\_wf}^{\Phi}$ checks that bound variables are correctly used as first-order or second-order with respect to their binders and excludes formulas with unused binders; unused binders are obviously superfluous.

An *interpretation* of an MSO formula is a pair of a word $w :: \alpha\,\mathsf{list}$ from Σ^* and an assignment $\mathcal{I} :: (\mathbb{N} + \mathbb{N}\,\mathsf{set})\,\mathsf{list}$ for free variables. The latter essentially consists of two functions with finite domain: one from first-order variables to positions and the other from second-order variables to sets of positions. We represent those two functions by a list, once again benefiting from de Bruijn indices—the value lookup for a variable with de Bruijn index i corresponds to inspecting the assignment $\mathcal{I}$ at position i, i.e. $\mathcal{I}[i]$. The range of $\mathcal{I}$ is a sum type, denoting the disjoint union of its two argument types. The sum type has two constructors $\mathsf{Inl} :: \alpha \to \alpha + \beta$ and $\mathsf{Inr} :: \beta \to \alpha + \beta$, such that for a first-order variable m there is a position p with $\mathcal{I}[m] = \mathsf{Inl}\ p$ and for a second-order variable M there is a finite set of positions P with $\mathcal{I}[M] = \mathsf{Inr}\ P$.

An interpretation that *satisfies* a formula is called a model. Satisfiability for M2L, denoted by infix $\vDash :: \alpha\,\mathsf{list} \times (\mathbb{N} + \mathbb{N}\,\mathsf{set})\,\mathsf{list} \to$



(a) norm is identity function

(b) norm unfolds $\mathbf{0} + r = r$, $\mathbf{0} \cdot r = \mathbf{0}$ and $\mathbf{1} \cdot r = r$

Figure 1. Checking the equivalence of a^* and $\mathbf{1} + a \cdot a^*$ for $\Sigma_n = \{a, b\}$

$\alpha\,\Phi \to \mathbb{B}$, is defined recursively on $\alpha\,\Phi$. To simplify the notation, the sum constructors Inl and Inr are stripped implicitly in the definition.

$$\begin{aligned}
(w,\mathcal{I}) &\vDash \mathsf{Q}\,a\,m &&\leftrightarrow w[\mathcal{I}[m]] = a\\
(w,\mathcal{I}) &\vDash m_1 < m_2 &&\leftrightarrow \mathcal{I}[m_1] < \mathcal{I}[m_2]\\
(w,\mathcal{I}) &\vDash m \in M &&\leftrightarrow \mathcal{I}[m] \in \mathcal{I}[M]\\
(w,\mathcal{I}) &\vDash \varphi \wedge \psi &&\leftrightarrow (w,\mathcal{I}) \vDash \varphi \wedge (w,\mathcal{I}) \vDash \psi\\
(w,\mathcal{I}) &\vDash \varphi \vee \psi &&\leftrightarrow (w,\mathcal{I}) \vDash \varphi \vee (w,\mathcal{I}) \vDash \psi\\
(w,\mathcal{I}) &\vDash \neg \varphi &&\leftrightarrow (w,\mathcal{I}) \nvDash \varphi\\
(w,\mathcal{I}) &\vDash \exists \varphi &&\leftrightarrow \exists p \in \{0,\dots,|w|-1\}.\ (w,\ \mathsf{Inl}\ p \mathbin{\#} \mathcal{I}) \vDash \varphi\\
(w,\mathcal{I}) &\vDash \boldsymbol{\exists} \varphi &&\leftrightarrow \exists P \subseteq \{0,\dots,|w|-1\}.\ (w,\ \mathsf{Inr}\ P \mathbin{\#} \mathcal{I}) \vDash \varphi
\end{aligned}$$

For the definition to make sense, $\mathcal{I}$ must correctly map first-order variables to positions (i.e. $\mathcal{I}[m] = \mathsf{Inl}\ p$) and second-order variables to sets of positions (i.e. $\mathcal{I}[M] = \mathsf{Inr}\ P$). Furthermore, all positions in $\mathcal{I}$ should be below the length of the word, and for technical reasons the word should not be empty. We formalize these assumptions by the predicate $\mathsf{wf}^{\mathsf{M2L}} :: \alpha\,\Phi \to \alpha\ \mathsf{list} \times (\mathbb{N} + \mathbb{N}\ \mathsf{set})\ \mathsf{list} \to \mathbb{B}$ and call an interpretation *M2L-wellformed* for φ if $\mathsf{wf}^{\mathsf{M2L}}_{\varphi}(w,\mathcal{I})$ holds:

$$\begin{aligned}
&\mathsf{wf}^{\mathsf{M2L}}_{\varphi}(w,\mathcal{I}) =\\
&\quad w \neq [\,] \wedge w \in \Sigma^* \wedge\\
&\quad \forall\, \mathsf{Inl}\ p \in \mathsf{set}\ \mathcal{I}.\ p < |w| \wedge\\
&\quad \forall\, \mathsf{Inr}\ P \in \mathsf{set}\ \mathcal{I}.\ (\forall p \in P.\ p < |w|) \wedge\\
&\quad \forall\, m \in \mathcal{V}_1(\varphi)\ (\exists p.\ \mathcal{I}[m] = \mathsf{Inl}\ p) \wedge\\
&\quad \forall\, M \in \mathcal{V}_2(\varphi)\ (\exists P.\ \mathcal{I}[M] = \mathsf{Inr}\ P)
\end{aligned}$$

5.2 WS1S Semantics

In an M2L-wellformed model, positions are restricted by the length of the word. This is the key difference compared to WS1S. In WS1S no a priori restrictions on the variable ranges are made, although all second-order variables still represent finite sets. The subtle difference is illustrated by the formula $\boldsymbol{\exists}\,(\forall\, 0 \in 1)$ (with names: $\exists X.\, \forall x.\ x \in X$), where $\forall \varphi$ is just an abbreviation for $\neg \exists \neg \varphi$. In the M2L semantics $\boldsymbol{\exists}\,(\forall\, 0 \in 1)$ is satisfied by all wellformed interpretations—the witness set for the outer existential quantifier is for a wellformed interpretation $(w,\mathcal{I})$ just the set $\{0,\dots,|w|-1\}$. In contrast, in WS1S, there is no finite set which contains all arbitrarily large positions, thus $\boldsymbol{\exists}\,(\forall\, 0 \in 1)$ is unsatisfiable.

Formally, satisfiability for WS1S, denoted by infix $\stackrel{\wedge}{=} :: \alpha\ \mathsf{list} \times (\mathbb{N} + \mathbb{N}\ \mathsf{set})\ \mathsf{list} \to \alpha\,\Phi \to \mathbb{B}$, is defined just as for M2L (replacing $\vDash$ by $\stackrel{\wedge}{=}$) except for the following equations.

$$\begin{aligned}
(w,\mathcal{I}) &\stackrel{\wedge}{=} \mathsf{Q}\,a\,m &&\leftrightarrow (\textbf{if}\ \mathcal{I}[m] < |w|\ \textbf{then}\ w[\mathcal{I}[m]]\ \textbf{else}\ \mathsf{z}) = a\\
(w,\mathcal{I}) &\stackrel{\wedge}{=} \exists \varphi &&\leftrightarrow \exists p.\ (w,\ \mathsf{Inl}\ p \mathbin{\#} \mathcal{I}) \stackrel{\wedge}{=} \varphi\\
(w,\mathcal{I}) &\stackrel{\wedge}{=} \boldsymbol{\exists} \varphi &&\leftrightarrow \exists P.\ (w,\ \mathsf{Inr}\ P \mathbin{\#} \mathcal{I}) \stackrel{\wedge}{=} \varphi \wedge \mathsf{finite}\ P
\end{aligned}$$

Here, z is a distinguished letter from Σ. WS1S as defined in the literature does not handle the $\mathsf{Q}\,a\,m$ case at all, usually interpreting formulas only with respect to the assignment $\mathcal{I}$. In order to be able to use the same syntax and the same type of interpretations for both semantics, we have made the above choice. This also allows us to translate $\mathsf{Q}\,a\,m$ into the same regular expression irrespective of the intended semantics.

Besides the mentioned relaxation of *WS1S-wellformedness* with respect to variable ranges, the empty word does not impose technical complications as in M2L. Therefore, the predicate $\mathsf{wf}^{\mathsf{WS1S}} :: \alpha\,\Phi \to \alpha\ \mathsf{list} \times (\mathbb{N} + \mathbb{N}\ \mathsf{set})\ \mathsf{list} \to \mathbb{B}$ is defined as follows.

$$\begin{aligned}
&\mathsf{wf}^{\mathsf{WS1S}}_{\varphi}(w,\mathcal{I}) =\\
&\quad w \in \Sigma^* \wedge\\
&\quad \forall\, \mathsf{Inr}\ P \in \mathsf{set}\ \mathcal{I}.\ \mathsf{finite}\ P \wedge\\
&\quad \forall\, m \in \mathcal{V}_1(\varphi)\ (\exists p.\ \mathcal{I}[m] = \mathsf{Inl}\ p) \wedge\\
&\quad \forall\, M \in \mathcal{V}_2(\varphi)\ (\exists P.\ \mathcal{I}[M] = \mathsf{Inr}\ P)
\end{aligned}$$

5.3 Encoding Interpretations as Words

Formulas are equivalent if they have the same set of wellformed models. To relate equivalent formulas with language equivalent regular expressions, the set of wellformed models must be represented as a formal language by encoding interpretations as words. As before, we cover the encoding of the M2L semantics first.

To simplify the formalization, we choose a very simple encoding using Boolean vectors. For an interpretation $(w,\mathcal{I})$, we associate with every position p in the word w a Boolean vector bs of length $|\mathcal{I}|$, such that $bs[m] = \top$ iff the mth variable in $\mathcal{I}$ is first-order and its value is p or it is second-order and its value contains p. For example, for $\Sigma = \{a, b\}$ the interpretation $(w,\mathcal{I}) = (aba,\ \mathsf{Inl}\ 0 \mathbin{\#} \mathsf{Inr}\ \{1,2\} \mathbin{\#} \mathsf{Inl}\ 2 \mathbin{\#} [\,])$ can be written in two dimensions as follows:

	a	b	a
$\mathsf{Inl}\ 0$	$\top$	$\bot$	$\bot$
$\mathsf{Inr}\ \{1,2\}$	$\bot$	$\top$	$\top$
$\mathsf{Inl}\ 2$	$\bot$	$\bot$	$\top$

In the first row, the value $\top$ is placed only in the first column because the first variable of $\mathcal{I}$ is the first-order position 0. In general, the columns correspond to the Boolean vectors associated with positions in the word, while every row corresponds to one variable. For first-order variables there must be exactly one $\top$ per row. The first row encodes the value of the most recently bound variable. Now, we consider every column as a letter of a new alphabet, which is the underlying alphabet $\Sigma_n = \Sigma \times \mathbb{B}^n$ of regular expressions of Section 4. This transformation of interpretations into words over Σ_n is performed by the function $\mathsf{enc}^{\mathsf{M2L}} :: \alpha\,\mathsf{list} \times (\mathbb{N} + \mathbb{N}\,\mathsf{set})\,\mathsf{list} \to (\alpha \times \mathbb{B}\,\mathsf{list})\,\mathsf{list}$; we omit its obvious definition.

Furthermore, the second parameter $\pi :: \Sigma_{n+1} \to \Sigma_n$ of our decision procedure for regular expressions can now be instantiated as the function that maps $(a, b \# bs)$ to (a, bs). Thus, the projection Π operates on words by removing the first row from words in the language of the body expression, reflecting the semantics of an existential quantifier.

Below we use a more visually appealing notation for elements of Σ_n. E.g. $(a, \top \# \bot \# \bot \# [\,])$ is written as

$$\begin{pmatrix} a \\ \top\bot\bot \end{pmatrix}.$$

Finally, the *M2L-language* $\mathcal{L}^{\mathsf{M2L}} :: \mathbb{N} \to \alpha\,\Phi \to (\alpha \times \mathbb{B}\,\mathsf{list})\,\mathsf{set}$ of an MSO formula is the set of encodings of its wellformed models, i.e. $\mathcal{L}^{\mathsf{M2L}}_n(\varphi) = \{\mathsf{enc}^{\mathsf{M2L}}(w,\mathcal{I}) \mid \mathsf{wf}^{\mathsf{M2L}}_\varphi(w,\mathcal{I}) \wedge |\mathcal{I}| = n \wedge (w,\mathcal{I}) \vDash \varphi\}$.

Concerning WS1S, the encoding is slightly more complicated due to the following observation: Interpretations $(w,\mathcal{I})$ and $(w\mathsf{z}^n,\mathcal{I})$ for all $n \in \mathbb{N}$ behave the same when considering satisfiability and wellformedness with respect to a formula (z^n denotes n-fold repetition of the letter z as a word). That suggests that the example interpretation $(w,\mathcal{I}) = (aba,\ \mathsf{Inl}\,0 \# \mathsf{Inr}\,\{1,2\} \# \mathsf{Inl}\,2 \# [\,])$ from above can be encoded as

	a	b	a	z^m
$\mathsf{Inl}\,0$	$\top$	$\bot$	$\bot$	$\bot^m$
$\mathsf{Inr}\,\{1,2\}$	$\bot$	$\top$	$\top$	$\bot^m$
$\mathsf{Inl}\,2$	$\bot$	$\bot$	$\top$	$\bot^m$

for every $m \in \mathbb{N}$. Hence, the a single WS1S interpretation is translated into a countably infinite set of words by a function $\mathsf{enc}^{\mathsf{WS1S}} :: \alpha\,\mathsf{list} \times (\mathbb{N} + \mathbb{N}\,\mathsf{set})\,\mathsf{list} \to (\alpha \times \mathbb{B}\,\mathsf{list})\,\mathsf{list}\,\mathsf{set}$; we again omit its formal definition. Accordingly, the *WS1S-language* $\mathcal{L}^{\mathsf{WS1S}} :: \mathbb{N} \to \alpha\,\Phi \to (\alpha \times \mathbb{B}\,\mathsf{list})\,\mathsf{set}$ of an MSO formula is defined by taking the union of all encodings of its wellformed models: $\mathcal{L}^{\mathsf{WS1S}}_n(\varphi) = \bigcup\{\mathsf{enc}^{\mathsf{M2L}}(w,\mathcal{I}) \mid \mathsf{wf}^{\mathsf{WS1S}}_\varphi(w,\mathcal{I}) \wedge |\mathcal{I}| = n \wedge (w,\mathcal{I}) \mathrel{\hat{\vDash}} \varphi\}$.

5.4 From Formulas to Regular Expressions

MSO formulas interpreted in M2L are translated into regular expressions by means of the primitive recursive function $\mathsf{mkRE}^{\mathsf{M2L}} :: \mathbb{N} \to \alpha\,\Phi \to \alpha\,\mathsf{RE}$ (see Figure 2). The natural number parameter of $\mathsf{mkRE}^{\mathsf{M2L}}$ indicates the number for free variables for the processed formula. The parameter is increased when entering recursively the scope of an existential quantifier. In general, the abbreviation

$$\begin{pmatrix} & X & \\ \underbrace{\bot/\top \cdots \bot/\top}_{m} & \top & \underbrace{\bot/\top \cdots \bot/\top}_{n-m-1} \end{pmatrix}$$

actually denotes the huge summation

$$\bigoplus_{\substack{a \in X \\ b_i \in \{\top,\bot\} \\ i \in \{0,\ldots,m-1,m+1,\ldots,n-1\}}} \begin{pmatrix} a \\ b_0 \cdots b_{m-1}\ \top\ b_{m+1} \cdots b_{n-1} \end{pmatrix}.$$

The intuition behind the translation is demonstrated by the case $\mathsf{Q}\,a\,m$. We fix a wellformed model $(w,\mathcal{I})$ of $\mathsf{Q}\,a\,m$. $(w,\mathcal{I})$ must satisfy $w[\mathcal{I}[m]] = a$, or equivalently the fact that there exists a Boolean vector bs of length n such that $\mathsf{enc}^{\mathsf{M2L}}(w,\mathcal{I})[\mathcal{I}[m]] = (a, bs)$ and $bs[m] = \top$. Therefore, the letter at position $\mathcal{I}[m]$ of $\mathsf{enc}^{\mathsf{M2L}}(w,\mathcal{I})$ is matched by the "middle" part of $\mathsf{mkRE}^{\mathsf{M2L}}_n(\mathsf{Q}\,a\,m)$, while the subexpressions $\neg\,\mathbf{0}$ (which denotes Σ_n^*) match the first $\mathcal{I}[m]$ and the last $n - \mathcal{I}[m]$ letters of $\mathsf{enc}^{\mathsf{M2L}}(w,\mathcal{I})$.

Conversely, if we fix a word from $\mathsf{mkRE}^{\mathsf{M2L}}_n(\mathsf{Q}\,a\,m)$, it will be equal to an encoding of an interpretation that satisfies $\mathsf{Q}\,a\,m$ by a similar argument. However, the interpretation might be not wellformed for $\mathsf{Q}\,a\,m$. This happens because the regular expression $\mathsf{mkRE}^{\mathsf{M2L}}_n(\mathsf{Q}\,a\,m)$ does not capture the distinction between first-order and second-order variables, such that it accepts encodings of interpretations that have the value $\top$ more than once at different positions representing the same first-order variable. This indicates that the subexpressions $\neg\,\mathbf{0}$ in the base cases are not precise enough, but also in the case of Boolean operators similar issues arise. So instead of tinkering with the base cases, it is better to separate the generation a regular expression that encodes models from the one that encodes wellformed interpretations.

To rule out not wellformed interpretations is exactly the purpose of the $\mathsf{WF} :: \mathbb{N} \to \alpha\,\Phi \to \alpha\,\mathsf{RE}$ function. The regular expression $\mathsf{WF}_n(\varphi)$ (see Figure 2) accepts exactly the encodings of wellformed interpretations (both models and non-models) for φ by ensuring that first-order variables are encoded correctly.

Lemma 9. *Let φ be an n-wellformed formula. Then*

- $\mathcal{L}_n(\mathsf{WF}_n(\varphi)) = \{\mathsf{enc}^{\mathsf{WS1S}}(w,\mathcal{I}) \mid \mathsf{wf}^{\mathsf{WS1S}}_\varphi(w,\mathcal{I}) \wedge |\mathcal{I}| = n\}$*, and*
- $\mathcal{L}_n(\mathsf{WF}_n(\varphi)) \setminus \{[\,]\} = \{\mathsf{enc}^{\mathsf{M2L}}(w,\mathcal{I}) \mid \mathsf{wf}^{\mathsf{M2L}}_\varphi(w,\mathcal{I}) \wedge |\mathcal{I}| = n\}$.

Using WF in every case of the recursive definition of $\mathsf{mkRE}^{\mathsf{M2L}}$ is very redundant—it is enough to perform the intersection once globally for the entire formula and additionally for every existential quantifier.

MSO formulas interpreted in WS1S are translated into regular expressions by means of the function $\mathsf{mkRE}^{\mathsf{WS1S}} :: \mathbb{N} \to \alpha\,\Phi \to \alpha\,\mathsf{RE}$.

The definition of $\mathsf{mkRE}^{\mathsf{WS1S}}$ coincides with the one of $\mathsf{mkRE}^{\mathsf{M2L}}$ except for the existential quantifier cases:

$$\mathsf{mkRE}^{\mathsf{WS1S}}_n(\exists\,\varphi) = \mathcal{Q}\begin{pmatrix}\mathsf{z}\\ \bot^n\end{pmatrix}\left(\Pi\left(\mathsf{mkRE}^{\mathsf{WS1S}}_{n+1}(\varphi) \cap \mathsf{WF}_{n+1}(\varphi)\right)\right)$$

$$\mathsf{mkRE}^{\mathsf{WS1S}}_n(\boldsymbol{\exists}\,\varphi) = \mathcal{Q}\begin{pmatrix}\mathsf{z}\\ \bot^n\end{pmatrix}\left(\Pi\left(\mathsf{mkRE}^{\mathsf{WS1S}}_{n+1}(\varphi) \cap \mathsf{WF}_{n+1}(\varphi)\right)\right)$$

The regular operation $\mathcal{Q} :: \alpha \times \mathbb{B}\,\mathsf{list} \to \alpha\,\mathsf{RE} \to \alpha\,\mathsf{RE}$ reestablishes the invariant of having all words terminated with a suffix $\begin{pmatrix}\mathsf{z}\\ \bot^n\end{pmatrix}^m$ for every $m \in \mathbb{N}$ in the WS1S language encoding of a formula as required by definition of $\mathsf{enc}^{\mathsf{WS1S}}$ (this invariant might be violated by the projection). More precisely, the following language identity holds for an n-wellformed regular expression r:

$$\mathcal{L}_n(\mathcal{Q}\,a\,r) = \left\{xa^m \mid m \in \mathbb{N} \wedge \exists l.\ xa^l \in \mathcal{L}_n(r)\right\}$$

We do not show the concrete executable definition of $\mathcal{Q}$ which can be found in our formalization. On a high-level, $\mathcal{Q}$ is computed by repeatedly deriving from the right by a (dual to $\mathcal{D}_a$ which derives from the left). The termination of the repeated derivation is established by the dual of Theorem 2 for ACI-equivalent "right derivatives".

Finally, we can establish the language correspondence between formulas and generated regular expressions.

Theorem 10. *Let φ be an n-wellformed formula. Then*

- $\mathcal{L}^{\mathsf{WS1S}}_n(\varphi) = \mathcal{L}_n(\mathsf{mkRE}^{\mathsf{WS1S}}_n(\varphi) \cap \mathsf{WF}_n(\varphi))$*, and*
- $\mathcal{L}^{\mathsf{M2L}}_n(\varphi) = \mathcal{L}_n(\mathsf{mkRE}^{\mathsf{M2L}}_n(\varphi) \cap \mathsf{WF}_n(\varphi)) \setminus \{[\,]\}$.

The proof is by structural induction on φ. Above we have seen the argument for the base case $\mathsf{Q}\,a\,m$, other base cases follow similarly. The cases $\exists\,\varphi$ and $\boldsymbol{\exists}\,\varphi$ follow easily from the semantics of Π given by our concrete instantiation for π and Σ_n and the induc-

$$\mathsf{mkRE}^{\mathsf{M2L}}_n(\mathsf{Q}\,a\,m) \;=\; \neg\mathbf{0}\cdot\begin{pmatrix} & \{a\} & \\ \underbrace{\bot/\top\cdots\bot/\top}_{m} & \top & \underbrace{\bot/\top\cdots\bot/\top}_{n-m-1}\end{pmatrix}\cdot\neg\mathbf{0}$$

$$\mathsf{mkRE}^{\mathsf{M2L}}_n(m_1<m_2) \;=\; \neg\mathbf{0}\cdot\begin{pmatrix} & \Sigma & \\ \underbrace{\bot/\top\cdots\bot/\top}_{m_1} & \top & \underbrace{\bot/\top\cdots\bot/\top}_{n-m_1-1}\end{pmatrix}\cdot\neg\mathbf{0}\cdot\begin{pmatrix} & \Sigma & \\ \underbrace{\bot/\top\cdots\bot/\top}_{m_2} & \top & \underbrace{\bot/\top\cdots\bot/\top}_{n-m_2-1}\end{pmatrix}\cdot\neg\mathbf{0}$$

$$\mathsf{mkRE}^{\mathsf{M2L}}_n(m\in M) \;=\; \neg\mathbf{0}\cdot\begin{pmatrix} & & \Sigma & & \\ \underbrace{\bot/\top\cdots\bot/\top}_{\mathsf{min}\,m\,M} & \top & \underbrace{\bot/\top\cdots\bot/\top}_{\mathsf{max}\,m\,M-\mathsf{min}\,m\,M-1} & \top & \underbrace{\bot/\top\cdots\bot/\top}_{n-\mathsf{max}\,m\,M-1}\end{pmatrix}\cdot\neg\mathbf{0}$$

$$\begin{aligned}
\mathsf{mkRE}^{\mathsf{M2L}}_n(\varphi\wedge\psi) &= \mathsf{mkRE}^{\mathsf{M2L}}_n(\varphi)\cap\mathsf{mkRE}^{\mathsf{M2L}}_n(\psi)\\
\mathsf{mkRE}^{\mathsf{M2L}}_n(\varphi\vee\psi) &= \mathsf{mkRE}^{\mathsf{M2L}}_n(\varphi)+\mathsf{mkRE}^{\mathsf{M2L}}_n(\psi)\\
\mathsf{mkRE}^{\mathsf{M2L}}_n(\neg\varphi) &= \neg\,\mathsf{mkRE}^{\mathsf{M2L}}_n(r)\\
\mathsf{mkRE}^{\mathsf{M2L}}_n(\exists\,\varphi) &= \Pi\big(\mathsf{mkRE}^{\mathsf{M2L}}_{n+1}(\varphi)\cap\mathsf{WF}_{n+1}(\varphi)\big)\\
\mathsf{mkRE}^{\mathsf{M2L}}_n(\boldsymbol{\exists}\,\varphi) &= \Pi\big(\mathsf{mkRE}^{\mathsf{M2L}}_{n+1}(\varphi)\cap\mathsf{WF}_{n+1}(\varphi)\big)
\end{aligned}$$

$$\mathsf{WF}_n(\varphi) \;=\; \bigcap_{m\in\mathcal{V}_1(\varphi)}\begin{pmatrix} & \Sigma & \\ \underbrace{\bot/\top\cdots\bot/\top}_{m} & \bot & \underbrace{\bot/\top\cdots\bot/\top}_{n-m-1}\end{pmatrix}^{*}\cdot\begin{pmatrix} & \Sigma & \\ \underbrace{\bot/\top\cdots\bot/\top}_{m} & \top & \underbrace{\bot/\top\cdots\bot/\top}_{n-m-1}\end{pmatrix}\cdot\begin{pmatrix} & \Sigma & \\ \underbrace{\bot/\top\cdots\bot/\top}_{m} & \bot & \underbrace{\bot/\top\cdots\bot/\top}_{n-m-1}\end{pmatrix}^{*}$$

Figure 2. Definition of $\mathsf{mkRE}^{\mathsf{M2L}}$ and WF

tion hypothesis. The most interesting cases are, somehow unexpectedly, those for Boolean operators. Although the definitions are purely structural, sets of encodings of models must be composed or, even worse, complemented in the inductive steps. The key property required here is that $\mathsf{enc}^{\mathsf{M2L}}$ (and $\mathsf{enc}^{\mathsf{WS1S}}$) do not collapse models and non-models: two different wellformed interpretations for a formula—one being a model, the other being a non-model—are encoded into different words (sets of words). This is again established by structural induction on formulas for both semantics.

Lemma 11. *Let $(w_1,\mathcal{I}_1)$ and $(w_2,\mathcal{I}_2)$ be two M2L-wellformed interpretations for φ such that $\mathsf{enc}^{\mathsf{M2L}}(w_1,\mathcal{I}_1)=\mathsf{enc}^{\mathsf{M2L}}(w_2,\mathcal{I}_2)$. Then $(w_1,\mathcal{I}_1)\models\varphi\leftrightarrow(w_2,\mathcal{I}_2)\models\varphi$.*

Lemma 12. *Let $(w_1,\mathcal{I}_1)$ and $(w_2,\mathcal{I}_2)$ be two WS1S-wellformed interpretations for φ such that $\mathsf{enc}^{\mathsf{WS1S}}(w_1,\mathcal{I}_1)=\mathsf{enc}^{\mathsf{WS1S}}(w_2,\mathcal{I}_2)$. Then $(w_1,\mathcal{I}_1)\mathrel{\dot{\models}}\varphi\leftrightarrow(w_2,\mathcal{I}_2)\mathrel{\dot{\models}}\varphi$.*

5.5 Deciding Language Equivalence of Formulas

The algorithms $\mathsf{eqv}^{\mathsf{M2L}} :: \mathbb{N}\to\alpha\,\Phi\to\alpha\,\Phi\to\mathbb{B}$ and $\mathsf{eqv}^{\mathsf{WS1S}} :: \mathbb{N}\to\alpha\,\Phi\to\alpha\,\Phi\to\mathbb{B}$ that decide language equivalence of MSO formulas check wellformedness of the input formulas, translate the formulas into regular expressions and let $\mathsf{eqv}^{\mathsf{RE}}$ do the work:

$$\begin{aligned}
&\mathsf{eqv}^{\mathsf{M2L}}_n\,\varphi\,\psi \;=\\
&\quad \mathsf{wf}^{\Phi}_n(\varphi\vee\psi)\,\wedge\\
&\quad \mathsf{eqv}^{\mathsf{RE}}_n\,(\mathsf{mkRE}^{\mathsf{M2L}}_n(\varphi)+\mathbf{1})\,(\mathsf{mkRE}^{\mathsf{M2L}}_n(\psi)+\mathbf{1})
\end{aligned}$$

$$\begin{aligned}
&\mathsf{eqv}^{\mathsf{WS1S}}_n\,\varphi\,\psi \;=\\
&\quad \mathsf{wf}^{\Phi}_n(\varphi\vee\psi)\,\wedge\\
&\quad \mathsf{eqv}^{\mathsf{RE}}_n\,(\mathsf{mkRE}^{\mathsf{WS1S}}_n(\varphi))\,(\mathsf{mkRE}^{\mathsf{WS1S}}_n(\psi))
\end{aligned}$$

Note that wellformedness is checked on the disjunction of both formulas to ensure that they agree on free variables (i.e. no first-order free variable of φ is used as a second-order free variable in ψ and vice versa). Further, we add the empty word into both regular expression when working with the M2L semantics. This is allowed, since [] is not a valid encoding of an interpretation, and necessary because Theorem 10 does not give us any information whether the empty word is contained in the output of $\mathsf{mkRE}^{\mathsf{M2L}}$ or not.

Termination of $\mathsf{eqv}^{\mathsf{RE}}$ is ensured by Theorem 6 and the definition principle of primitive recursion for wf^{Φ}, $\mathsf{mkRE}^{\mathsf{M2L}}$ and $\mathsf{mkRE}^{\mathsf{WS1S}}$. Soundness and completeness follow easily from Theorems 7, 8 and 10.

Theorem 13 (Soundness)**.** *Let φ and ψ be MSO formulas.*

- *If $\mathsf{eqv}^{\mathsf{M2L}}_n\,\varphi\,\psi$, then $\mathcal{L}^{\mathsf{M2L}}_n(\varphi)=\mathcal{L}^{\mathsf{M2L}}_n(\psi)$.*
- *If $\mathsf{eqv}^{\mathsf{WS1S}}_n\,\varphi\,\psi$, then $\mathcal{L}^{\mathsf{WS1S}}_n(\varphi)=\mathcal{L}^{\mathsf{WS1S}}_n(\psi)$.*

Theorem 14 (Completeness)**.** *Let $\varphi\vee\psi$ be an n-wellformed MSO formula.*

- *If $\mathcal{L}^{\mathsf{M2L}}_n(\varphi)=\mathcal{L}^{\mathsf{M2L}}_n(\psi)$, then $\mathsf{eqv}^{\mathsf{M2L}}_n\,\varphi\,\psi$.*
- *If $\mathcal{L}^{\mathsf{WS1S}}_n(\varphi)=\mathcal{L}^{\mathsf{WS1S}}_n(\psi)$, then $\mathsf{eqv}^{\mathsf{WS1S}}_n\,\varphi\,\psi$.*

6. Application: Finite-Word LTL

We want to execute the code generated by Isabelle/HOL for our decision procedures on some larger examples. For simplicity, we focus on M2L.

In order to create larger formulas, it is helpful to introduce some syntactic abbreviations. We define the unsatisfiable formula $\bot$ as $\exists\,0<0$ and the valid formula $\top$ as $\neg\bot$. Now, checking that a formula is valid amounts to checking its equivalence to $\top$. Implication $\varphi\to\psi$ is defined as $(\neg\varphi)\vee\psi$ and universal quantification $\forall\,\varphi$ as before as $\neg\exists\neg\varphi$. Next, we introduce temporal logical operators *always* $\Box P :: \mathbb{N}\to\alpha\,\Phi$ and *eventually* $\Diamond P :: \mathbb{N}\to\alpha\,\Phi$ depending on $P :: \mathbb{N}\to\alpha\,\Phi$—a formula parameterized by a single variable indicating the time. The operators have their usual meaning except that with the given M2L semantics the time variable ranges over a fixed set determined by the interpretation. Additionally, we lift the disjunction and implication to time-parameterized formulas.

$$\begin{array}{lcl} \Box P\, t & = & \forall\,(\neg\, t+1<0 \rightarrow P\, 0) \\ \Diamond P\, t & = & \exists\,(\neg\, t+1<0 \wedge P\, 0) \\ (P \Rightarrow Q)\, t & = & P\, t \rightarrow Q\, t \\ (P \textcircled{\vee} Q)\, t & = & P\, t \vee Q\, t \end{array}$$

Note that $t+1$ has nothing to do with the next time step. It is just the lifting of the de Bruijn index under a single quantifier.

Further, formulas of linear temporal logic contain atomic predicates for which the interpretation must specify at which points in time they are true. This information can be encoded in two ways, which we compare in the following.

The first possibility is to encode atomic predicates in the word of the interpretation. This is done by identifying Σ with the powerset $\mathcal{P}$ of atomic predicates. For every point in time, that is for every position in the word, the letter is the set of predicates that are true at this point. Using this encoding we can prove the validity of the following closed formulas over the alphabet $\mathcal{P}\{P\} = \{\{P\}, \{\}\}$ automatically within a few milliseconds.

$$\begin{array}{c} \forall\,(\Box(\mathsf{Q}\{P\}) \Rightarrow \Diamond(\mathsf{Q}\{P\}))\ 0 \\ \forall\,(\Box(\mathsf{Q}\{P\}) \Rightarrow \Box\Diamond(\mathsf{Q}\{P\}))\ 0 \end{array}$$

Alternatively, a free second-order variable can be used to encode an atomic predicate directly. The variable denotes the set of points in time for which the atomic predicate holds. The alphabet Σ can then be trivial, i.e. $\Sigma = \{a\}$ for an arbitrary a. Using this encoding the above two formulas correspond to

$$\begin{array}{c} \forall\,(\Box(\lambda t.\, t\in 2) \Rightarrow \Diamond(\lambda t.\, t\in 2))\ 0 \\ \forall\,(\Box(\lambda t.\, t\in 2) \Rightarrow \Box\Diamond(\lambda t.\, t\in 3))\ 0 \end{array}$$

Both formulas have one free second-order variable 0 that is lifted when passing two or three quantifiers. The generated algorithm shows the equivalence to $\top$ again within milliseconds.

In order to explore the limits of our decision procedure, formulas over more atomic predicates are required. Therefore, we consider the distributivity theorems of $\Box$ over implication for both representations of atomic predicates as shown in Figure 3. When the number of predicates n is increased, the size of φ_n grows exponentially: to express that a predicate P holds at some position we need the disjunction of all atoms containing P. In contrast, the size of ψ_n grows linearly. The complexity of ψ_n is hidden in its encoding—the latter also grows exponentially with increasing n. The running times of the decision procedure are summarized in Figure 4. Thereby, ψ_1, ψ_2 and ψ_3 were processed over $\Sigma = \{a\}$, φ_1 was processed over $\Sigma = \mathcal{P}\{P\}$, φ_2 over $\Sigma = \mathcal{P}\{P_1, P_2\}$ and finally φ_3 over $\Sigma = \mathcal{P}\{P_1, P_2, P_3\}$. Figure 4 also shows the sizes of the generated regular expressions. Both size and $\mathsf{size}_{\mathsf{len}}$ count the constructors in a regular expression. The difference is that the size of an atom is 1 whereas the $\mathsf{size}_{\mathsf{len}}$ of an atom is the length of its Boolean vector.

The attentive reader will have noticed that we have said nothing about how sets are represented in the code generated from our mathematical definitions. We have chosen an existing verified red–black tree implementation for our measurements. Isabelle's code generator supports the transparent replacement of sets by some verified implementation [14].

The performance of our automatically generated code may appear disappointing but that would be a misunderstanding of our intentions. We see our work primarily as a succinct and elegant functional program that may pave the way towards verified and efficient decision procedures. As a bonus, the generated code is applicable to small examples. In the context of interactive theorem proving, this is primarily what one encounters: small formulas. Any automation is welcome here because it saves the user time and effort. Automatic verification of larger systems is the domain of highly tuned implementations such as MONA.

7. Conclusion

We have presented functional programs that decide equivalence of MSO formulas for two different semantics in Isabelle/HOL. They come with formal proofs of termination, soundness and completeness. The programs operate by translating formulas into Π-extended regular expressions and deciding the language equivalence of the latter using Brzozowski derivatives. Although formalized in Isabelle/HOL's functional programming language, we can automatically generate code from them in different functional target languages. The development amounts to roughly 350 lines of functional programs and 5000 lines of proofs, of which 2100 lines are devoted to deciding equivalence of Π-extended regular expressions. For M2L, the program is completely contained in this paper. The Isabelle scripts are publicly available [27].

Our work can be continued in two dimensions. First, the algorithm is not optimized. Especially the encoding of interpretations as Boolean vectors leaves room for improvement.

Second, several related decidable logics can be formalized and verified using similar technology. A related logic is MSO on infinite words (also called S1S). S1S formulas can be translated into ω-regular expressions representing ω-regular languages. A verified decision procedure for deciding equivalence of ω-regular expressions without constructing ω-automata is an interesting challenge. An even more distant goal is to move from words to trees (or even from ω-words to ω-trees) and decide equivalence of MSO formulas on (in)finite trees (or alternatively (W)S2S formulas) by translating them into (ω-)regular tree expressions.

Acknowledgments We thank Alexander Krauss for inspiring discussions and Jasmin Christian Blanchette for numerous comments on the presentation. Comments and questions of four anonymous reviewers helped to clarify certain fine points. The first author's research is supported by the doctorate program 1480 (PUMA) of the Deutsche Forschungsgemeinschaft (DFG).

References

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

[2] A. Asperti. A compact proof of decidability for regular expression equivalence. In L. Beringer and A. Felty, eds., *Interactive Theorem Proving, ITP 2012*, vol. 7406 of *LNCS*, pp. 283–298. Springer, 2012.

[3] A. Ayari and D. Basin. Bounded model construction for monadic second-order logics. In E. A. Emerson and A. P. Sistla, eds., *Proc. Int. Conf. Computer Aided Verification, CAV 2000*, vol. 1855 of *LNCS*, pp. 99–112. Springer, 2000.

[4] C. Ballarin. Interpretation of locales in isabelle: Theories and proof contexts. In J. M. Borwein and W. M. Farmer, eds., *Mathematical Knowledge Management, MKM 2006*, vol. 4108 of *LNCS*, pp. 31–43. Springer, 2006.

[5] D. Basin and S. Friedrich. Combining WS1S and HOL. In D. Gabbay and M. de Rijke, eds., *Frontiers of Combining Systems 2*, vol. 7 of *Studies in Logic and Computation*, pp. 39–56. Research Studies Press/Wiley, 2000.

[6] S. Berghofer and M. Reiter. Formalizing the logic-automaton connection. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, eds., *Theorem Proving in Higher Order Logics, TPHOLs 2009*, vol. 5674 of *LNCS*, pp. 147–163. Springer, 2009.

[7] T. Braibant and D. Pous. An efficient Coq tactic for deciding Kleene algebras. In M. Kaufmann and L. Paulson, eds., *Interactive Theorem Proving, ITP 2010*, vol. 6172 of *LNCS*, pp. 163–178. Springer, 2010.

$$\begin{aligned}
\varphi_1 &= \forall\,(\Box\,(\mathsf{Q}\{P\}) \Rightarrow \Box\,(\mathsf{Q}\{P\}))\ 0\\
\varphi_2 &= \forall\,(\Box\,(\mathsf{Q}\{P_1\} \circledvee \mathsf{Q}\{P_1,P_2\} \Rightarrow \mathsf{Q}\{P_2\} \circledvee \mathsf{Q}\{P_1,P_2\}) \Rightarrow \Box\,(\mathsf{Q}\{P_1\} \circledvee \mathsf{Q}\{P_1,P_2\}) \Rightarrow \Box\,(\mathsf{Q}\{P_2\} \circledvee \mathsf{Q}\{P_1,P_2\}))\ 0\\
\varphi_3 &= \forall\,(\Box\,(\mathsf{Q}\{P_1\} \circledvee \mathsf{Q}\{P_1,P_2\} \circledvee \mathsf{Q}\{P_1,P_3\} \circledvee \mathsf{Q}\{P_1,P_2,P_3\} \Rightarrow\\
&\qquad\qquad \mathsf{Q}\{P_2\} \circledvee \mathsf{Q}\{P_1,P_2\} \circledvee \mathsf{Q}\{P_2,P_3\} \circledvee \mathsf{Q}\{P_1,P_2,P_3\} \Rightarrow \mathsf{Q}\{P_3\} \circledvee \mathsf{Q}\{P_1,P_3\} \circledvee \mathsf{Q}\{P_2,P_3\} \circledvee \mathsf{Q}\{P_1,P_2,P_3\}) \Rightarrow\\
&\qquad \Box\,(\mathsf{Q}\{P_1\} \circledvee \mathsf{Q}\{P_1,P_2\} \circledvee \mathsf{Q}\{P_1,P_3\} \circledvee \mathsf{Q}\{P_1,P_2,P_3\}) \Rightarrow\\
&\qquad \Box\,(\mathsf{Q}\{P_2\} \circledvee \mathsf{Q}\{P_1,P_2\} \circledvee \mathsf{Q}\{P_2,P_3\} \circledvee \mathsf{Q}\{P_1,P_2,P_3\}) \Rightarrow\\
&\qquad \Box\,(\mathsf{Q}\{P_3\} \circledvee \mathsf{Q}\{P_1,P_3\} \circledvee \mathsf{Q}\{P_2,P_3\} \circledvee \mathsf{Q}\{P_1,P_2,P_3\}))\ 0\\
\psi_1 &= \forall\,(\Box\,(\lambda t.\,t \in 2) \Rightarrow \Box\,(\lambda t.\,t \in 2))\ 0\\
\psi_2 &= \forall\,(\Box\,(\lambda t.\,t \in 2 \to t \in 3) \Rightarrow \Box\,(\lambda t.\,t \in 2) \Rightarrow \Box\,(\lambda t.\,t \in 3))\ 0\\
\psi_3 &= \forall\,(\Box\,(\lambda t.\,t \in 2 \to t \in 3 \to t \in 4) \Rightarrow \Box\,(\lambda t.\,t \in 2) \Rightarrow \Box\,(\lambda t.\,t \in 3) \Rightarrow \Box\,(\lambda t.\,t \in 4))\ 0
\end{aligned}$$

Figure 3. Definition of φ_n and ψ_n

	Time to prove φ_n	Time to prove ψ_n	$\mathsf{size}\,(\mathsf{mkRE}_0^{\mathsf{M2L}}(\varphi_n))$	$\mathsf{size}\,(\mathsf{mkRE}_n^{\mathsf{M2L}}(\psi_n))$	$\mathsf{size}_{\mathsf{len}}\,(\mathsf{mkRE}_0^{\mathsf{M2L}}(\varphi_n))$	$\mathsf{size}_{\mathsf{len}}\,(\mathsf{mkRE}_n^{\mathsf{M2L}}(\psi_n))$
n=1	2ms	2ms	262	262	330	404
n=2	2s	2s	741	960	949	1370
n=3	81min	44min	1920	1836	2480	4148

Figure 4. Comparison of φ_n and ψ_n in the M2L semantics

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

[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., *Proc. Int. Conf. Language and Automata Theory and Applications, LATA 2011*, vol. 6638 of *LNCS*, pp. 179–191. Springer, 2011.

[10] T. Coquand and V. Siles. A decision procedure for regular expression equivalence in type theory. In J.-P. Jouannaud and Z. Shao, eds., *Proc. Int. Conf. Certified Programs and Proofs, CPP 2011*, vol. 7086 of *LNCS*, pp. 119–134. Springer, 2011.

[11] N. A. Danielsson. Total parser combinators. In P. Hudak and S. Weirich, eds., *Proc. Int. Conf. Functional Programming, ICFP 2010*, pp. 285–296. ACM, 2010.

[12] J. Elgaard, N. Klarlund, and A. Møller. MONA 1.x: new techniques for WS1S and WS2S. In A. J. Hu and M. Y. Vardi, eds., *Proc. Int. Conf. Computer Aided Verification, CAV 1998*, vol. 1427 of *LNCS*, pp. 516–520. Springer, 1998.

[13] S. Fischer, F. Huch, and T. Wilke. A play on regular expressions: functional pearl. In P. Hudak and S. Weirich, eds., *Proc. Int. Conf. Functional Programming, ICFP 2010*, pp. 357–368. ACM, 2010.

[14] F. Haftmann, A. Krauss, O. Kunčar, and T. Nipkow. Data refinement in Isabelle/HOL. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, eds., *Interactive Theorem Proving, ITP 2013*, vol. 7998 of *LNCS*. Springer, 2013.

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

[16] 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., *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 1995*, vol. 1019 of *LNCS*, pp. 89–110. Springer, 1995.

[17] N. Klarlund. A theory of restrictions for logics and automata. In N. Halbwachs and D. Peled, eds., *Proc. Int. Conf. Computer Aided Verification, CAV 1999*, vol. 1633 of *LNCS*, pp. 406–417. Springer, 1999.

[18] A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *J. Automated Reasoning*, 49:95–106, 2012. published online March 2011.

[19] M. Might, D. Darais, and D. Spiewak. Parsing with derivatives: A functional pearl. In M. M. T. Chakravarty, Z. Hu, and O. Danvy, eds., *Proc. Int. Conf. Functional Programming, ICFP 2011*, pp. 189–195. ACM, 2011.

[20] N. Moreira, D. Pereira, and S. M. de Sousa. Deciding regular expressions (in-)equivalence in Coq. In W. Kahl and T. Griffin, eds., *Relational and Algebraic Methods in Computer Science, RAMiCS 2012*, vol. 7560 of *LNCS*, pp. 98–113. Springer, 2012.

[21] T. Nipkow. Programming and proving in Isabelle/HOL. `http://isabelle.in.tum.de/doc/prog-prove.pdf`.

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

[23] S. Owens, J. H. Reppy, and A. Turon. Regular-expression derivatives re-examined. *J. Funct. Program.*, 19(2):173–190, 2009.

[24] S. Owre and H. Rueß. Integrating WS1S with PVS. In E. A. Emerson and A. P. Sistla, eds., *Proc. Int. Conf. Computer Aided Verification, CAV 2000*, vol. 1855 of *LNCS*, pp. 548–551. Springer, 2000.

[25] J. J. M. M. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorgi and R. de Simone, eds., *Proc. Int. Conf. Concurrency Theory, CONCUR 1998*, vol. 1466 of *LNCS*, pp. 194–218. Springer, 1998.

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

[27] D. Traytel and T. Nipkow. Formal development associated with this paper. `http://www21.in.tum.de/~traytel/icfp13_mso.tar.gz`.

[28] C. Wu, X. Zhang, and C. Urban. A formalisation of the Myhill–Nerode theorem based on regular expressions (Proof pearl). In M. Eekelen, H. Geuvers, J. Schmaltz, and F. Wiedijk, eds., *Interactive Theorem Proving, ITP 2011*, vol. 6898 of *LNCS*, pp. 341–356. Springer, 2011.