HAL
open science

Regular Language Representations in the Constructive Type Theory of Coq

Christian Doczkal, Gert Smolka

To cite this version:

Christian Doczkal, Gert Smolka. Regular Language Representations in the Constructive Type Theory of Coq. Journal of Automated Reasoning, 2018, Special Issue: Milestones in Interactive Theorem Proving, 61 (1-4), pp.521-553. 10.1007/s10817-018-9460-x . hal-01832031

HAL Id: hal-01832031

https://hal.science/hal-01832031v1

Submitted on 6 Jul 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Regular Language Representations in the Constructive Type Theory of Coq

Christian Doczkal[†] · **Gert Smolka**

March 5, 2018

Abstract We explore the theory of regular language representations in the constructive type theory of Coq. We cover various forms of automata (deterministic, nondeterministic, one-way, two-way), regular expressions, and the logic WS1S. We give translations between all representations, show decidability results, and provide operations for various closure properties. Our results include a constructive decidability proof for the logic WS1S, a constructive refinement of the Myhill-Nerode characterization of regularity, and translations from two-way automata to one-way automata with verified upper bounds for the increase in size. All results are verified with an accompanying Coq development of about 3000 lines.

Keywords Regular Languages · Two-Way Automata · WS1S · Constructive Type Theory · Interactive Theorem Proving · Coq · Ssreflect

1 Introduction

Regular languages and their representations [20, 24, 22] are an important topic in Computer Science. The representations of regular languages range from various forms of automata (deterministic, nondeterministic, one-way, two-way) to regular expressions and formulas in monadic second-order logic. The core of the theory consists of algorithms that translate between different representations, decide language properties like membership and emptiness, and realize language operations such as union, complement, concatenation, iteration, word reversal, right-quotient, and image and preimage under word homomorphisms. The

[†]This author has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157).

This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

C. Doczkal
Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, France
E-mail: christian.doczkal@ens-lyon.fr

G. Smolka
Saarland University, Saarbrücken, Germany
E-mail: smolka@ps.uni-saarland.de

translation of formulas to automata yields a decision method for WS1S, a logic with non-elementary complexity interesting in its own right. Minimal DFAs provide a normal form for DFAs that is unique up to renaming. There is also an abstract characterization of regular languages based on finite partitions (Myhill-Nerode relations).

We present a formal development of the aforementioned results in the constructive type theory of Coq [35]. We observe that constructive type theory provides for an elegant presentation of the theory of regular languages, improving on the usual set-theoretic presentation by addressing computational issues explicitly and by providing an expressive facility for inductive definitions. One aspect where the constructive approach leads to new insight is the Myhill-Nerode theorem, which in the literature is formulated as a nonconstructive characterization of regular languages. We refine the theorem such that we obtain a constructive characterization of regular languages with a class of finite partitions we call classifiers. Classifiers provide a big-step representation of DFAs useful for establishing decidability results and as intermediate representation for the translation of 2NFAs to DFAs.

First-class finite types and dependent types are essential for our development. Languages are formalized as predicates on words, words are formalized as lists over alphabets, and alphabets are formalized as finite types of symbols. Automata are formalized as structures involving an alphabet and a finite type of states. Classifiers and finite partitions are formalized as functions mapping words to elements of a finite type. Given a two-way automaton and an input word, we represent the possible configurations as a finite type. Here, dependent types are used to represent the finitely many possible head positions on the input word. In order to be able to quantify over automata, (e.g., in the definition of regularity) it is essential that finite types are first-class objects, which is not the case in HOL-based provers [42].

The results we prove are known in the literature. Our efforts concern constructive proofs and the formalization of the results and definitions in constructive type theory. Considerable refinement was needed for the constructive formulation of the Myhill-Nerode theorem. Two of our main results, the characterization of regular languages with WS1S formulas [6, 13, 37] and with two-way automata [31, 34, 39], have not been formalized before, although their technical complexity certainly calls for machined-checked proofs. In the case of two-way automata, we adapt the proofs of Vardi [39] and Shepherdson [34] to tapes with endmarkers and generalize Shepherdson's proof from 2DFAs to 2NFAs.

Our main goal is an elegant presentation of the results in constructive type theory with machine-checked proofs. Constructive type theory ensures that all constructed functions are computable. Our focus is on clarity and simplicity, concrete executability is not a concern.

The paper develops the theory and the proofs in an informal mathematical language based on constructive type theory and is accompanied by a Coq development [10]. The paper and the Coq development are aligned such that it is easy to go back and forth. All numbered theorems in the paper correspond to theorems in the Coq development, and the proofs in the paper outline the Coq proofs. When appropriate, we discuss issues concerning the Coq development in the paper, but we do not give Coq code in the paper.

The Coq development employs the Ssreflect extension and makes essential use of the basic modules of the Mathematical Components library [36]. The development consists of about 3000 lines of code. The two biggest parts are the results about WS1S (870 lines) and two-way automata (550 lines).

As it comes to related work, we distinguish between work concerned with the theory of regular languages and work concerned with the formalization of regular language theory using interactive theorem provers. Papers from the first group will be mentioned in the technical sections of the paper they relate to, while papers from the second group will be discussed in Section 14.

The present paper extends and revises the material of two previous conference papers [11,12] of the authors.

The paper is organized as follows. We start with type-theoretic preliminaries (Section 2) and fix notations for languages (Section 3). We then define the notion of regularity using DFAs (Section 4). Afterwards, we consider classifiers (Section 5), NFAs (Section 6), and regular regular expressions (Section 7) and show that they yield representations of of regular languages. The remaining part of the paper studies DFA minimization (Section 8), two-way automata (Sections 9 to 11), and the logic WS1S (Sections 12 and 13).

2 Type Theory Preliminaries

We formalize our results in the constructive and intensional type theory of the proof assistant Coq. In this setting, decidability properties are of great importance. A *decider* for a predicate $P : X \to \mathsf{Prop}$ is a boolean predicate $p : X \to \mathbb{B}$ satisfying

$$\forall x. P x \leftrightarrow (p x = \mathsf{true})$$

We call a predicate *decidable* if it has a decider. Similar to predicates, a decidable proposition is a proposition that is equivalent to some boolean expression. We will mostly use the same notation for decidable propositions and the associated boolean expressions. In particular, if a boolean b appears as a proposition, it is to be read as $b = \mathsf{true}$.

In Coq, operations such as boolean equality tests and choice operators are not available for all types. Nevertheless, there are certain classes of types for which these operations are definable. For our purposes, three classes of types are of particular importance. These are discrete types, countable types, and finite types [16].

We call a type X *discrete* if equality on (elements of) X is decidable. The type of booleans $\mathbb{B}$ and the type of natural numbers $\mathbb{N}$ are both discrete.

We call a type X *countable* if there are functions $f : X \to \mathbb{N}$ and $g : \mathbb{N} \to X_\perp$ ($X_\perp$ being the option type over X) such that $g(f x) = \mathsf{Some} x$ for all $x : X$. Since $\mathbb{N}$ is discrete, all countable types are also discrete. Moreover, every countable type X comes equipped with a choice operator

$$\mathsf{xchoose}_X : \forall p : X \to \mathbb{B}. (\exists x. p x) \to X$$

satisfying $p(\mathsf{xchoose}_X\, p\, E)$ for all $E : (\exists x. p x)$. That is, $\mathsf{xchoose}_X$ turns abstract[1] existence proofs into concrete witnesses. If X is countable, the type X^* of lists over X is also countable.

We will repeatedly make use of the fact that surjective functions from countable types to discrete types have right inverses.

Lemma 2.1 *Let X be countable, let Y be discrete, and let $f : X \to Y$ be surjective. Then one can define a function $\overline{f} : Y \to X$ such that $f(\overline{f} y) = y$ for all y.*

Proof By surjectivity, we have a proof $E : (\forall y \exists x. f x = y)$. It is easy to verify that $\overline{f} y := \mathsf{xchoose}_X (E\, y)$ is a right inverse as required. □

A *finite type* is a discrete type X together with a duplicate free list enumerating all elements of X. If X is finite, we write $|X|$ for the number of elements of X and $\mathsf{rank}\, x$ for the position of an element $x : X$ in the list enumerating the type. For our purposes, the most important property of finite types is that quantification over finite types preserves decidability.

[1] Here, abstract means that no witness can be extracted from the existence proof. The function $\mathsf{xchoose}_X$ computes a witness by enumerating elements of X. The proof argument is only used to guarantee termination.

Discrete, countable, and finite types are closed under forming product types $X \times Y$, sum types $X + Y$, and option types $X_\perp$. Moreover, all three classes of types are closed under building subtypes with respect to decidable predicates. Let $p : X \to \mathbb{B}$. The Σ-type $\{x : X \mid px\}$, whose elements are dependent pairs of elements $x : X$ and proofs of $px = \mathsf{true}$, can be treated as a subtype of X. In particular, the first projection yields an injection from $\{x : X \mid px\}$ to X since $px = \mathsf{true}$ is proof irrelevant, i.e., has at most one proof [18].

Finite types also come with a *power operator*. That is, if X and Y are finite types then there is a finite type Y^X whose $|Y|^{|X|}$ elements represent the functions from X to Y up to extensionality. We write 2^X for the finite type of (extensional) finite sets with decidable membership represented as $\mathbb{B}^X$. If a finite type X appears as a set, it is to be read as the full set over X.

3 Languages in Constructive Type Theory

An *alphabet* is a finite type. A *word over an alphabet* is a list over the alphabet. We shall use the letters Σ and Γ for alphabets. Since we write X^* for the type of lists over X, Σ^* denotes the type of words over Σ. The elements of an alphabet are called *symbols*. We shall use the letters a and b for symbols and the letters x, y, and z for words. As usual, we write ε for the *empty word* (i.e., the empty list), xy or $x \cdot y$ for the concatenation of words, a for the singleton word $[a]$, a^n for the word consisting of n occurrences of the symbol a, and $|x|$ for the length of words. We also write $x[n,m]$ for the subword from position n (inclusive) to m (exclusive).

A *language over* Σ is a unary predicate on Σ^*. We shall write $x \in L$ for Lx when convenient. Given that we work in an intensional type theory, languages containing the same words are not necessarily equal. We call two languages *equivalent* and write $L_1 \equiv L_2$ if they contain the same words. The absence of extensionality will not cause difficulties since all our constructions respect language equivalence. Note that in contrast to languages, which are predicates on the infinite type Σ^*, sets over finite types (e.g., sets of states) are represented extensionally.

We employ the usual notations for languages. We write $\emptyset$ for the *empty language*, Σ^* for the language of all words, $\overline{L} := \Sigma^* \setminus L$ for *complements* of languages, $L_1 \cdot L_2 := \{xy \mid x \in L_1, y \in L_2\}$ for *concatenations* of languages, $L^0 := \{\varepsilon\}$ and $L^{n+1} := L \cdot L^n$ for *powers* of languages, and $L^* := \bigcup_n L^n$ for *Kleene stars* of languages. Further, we write $L_1/L_2 := \{x \mid \exists y \in L_2 . xy \in L_1\}$ for *right quotients* and $L_1 \backslash L_2 := \{y \mid \exists x \in L_1 . xy \in L_2\}$ for *left quotients* of languages. Moreover, we write $\mathscr{R}_L(x) := \{y \mid xy \in L\}$ for the *residual language* of L with respect to x. Note that residual languages are a special case of left quotients. That is, we have $\mathscr{R}_L(x) \equiv \{x\}\backslash L$ (but not $\mathscr{R}_L(x) = \{x\}\backslash L$ due to the absence of extensionality).

A function $h : \Sigma^* \to \Gamma^*$ from words to words is a *(word) homomorphism* if $h(x \cdot y) = h(x) \cdot h(y)$. We write $h^{-1}(L) := \{x \mid hx \in L\}$ for the *preimage* of L under h and $h(L) := \{y \mid \exists x \in L . hx = y\}$ for the *image* of L under h.

Lemma 3.1 *Let $x \in \Sigma^*$, let L and L' be decidable languages, and let h be a homomorphism. Then the languages Σ^*, $\emptyset$, $\{x\}$, $\overline{L}$, $L \cup L'$, $L \cap L'$, $L \cdot L'$, L^*, $\mathscr{R}_L(x)$, and $h^{-1}(L)$ are decidable.*

Proof Let $x : \Sigma^*$. Then $x \in L \cdot L'$ is decidable since there are only finitely many ways to split x. Decidability of $x \in L^*$ then follows by complete induction on $|x|$ using the equivalence $ax \in L^* \leftrightarrow x \in \mathscr{R}_L(a) \cdot L^*$. All other claims are trivial. □

Note that the language $h(L)$ is generally not decidable even for decidable L.[2] The languages associated to the various representations of regular languages will always be decidable.

4 Deterministic Finite Automata

Deterministic finite automata (DFAs) can be seen as the most basic operational representation of regular languages. We take DFAs to be the representation defining the notion of regularity.

Definition 4.1 A *deterministic finite automaton (DFA)* is a structure (Q, s, F, δ) where
- Q is a finite type of *states*.
- $s : Q$ is the *starting state*.
- $F : 2^Q$ is the set of *final states*.
- $\delta : Q \to \Sigma \to Q$ is the *transition function*.

In Coq, we represent DFAs using dependent records:

$$\begin{aligned} \mathsf{dfa} := \{\ & \mathsf{state} : \mathsf{finType} \\ & \mathsf{start} : \mathsf{state} \\ & \mathsf{final} : \mathsf{set\ state} \\ & \mathsf{trans} : \mathsf{state} \to \Sigma \to \mathsf{state}\} \end{aligned}$$

Here, $\mathsf{state} : \mathsf{finType}$ restricts the type of states to be a finite type. The combination of dependent records and finite types provides for a formalization of finite automata that is very convenient to work with. In particular, there are no well-formedness conditions.

Let $A = (Q, s, F, \delta)$ be a DFA. We extend δ to a function $\hat{\delta} : Q \to \Sigma^* \to Q$ by recursion on words:[3]

$$\begin{aligned} \hat{\delta}\, q\, \varepsilon &:= q \\ \hat{\delta}\, q(ax) &:= \hat{\delta}\,(\delta\, q\, a)\, x \end{aligned}$$

If $\hat{\delta}\, q\, x \in F$ for some state q and some word x, we say that q *accepts* x. The *language of* A is then defined as the language accepted by the starting state:

$$\mathscr{L}(A) := \{\, x \in \Sigma^* \mid \hat{\delta}\, s\, x \in F \,\}$$

Note that $\mathscr{L}(A)$ is a decidable language.

Definition 4.2 A DFA A *accepts* a language L if $L \equiv \mathscr{L}(A)$.

Definition 4.3 A language is *regular* if it is accepted by some DFA.

Regular languages enjoy a number of closure properties. The closure under boolean operations (e.g., complement and intersection) and preimages of homomorphisms can easily be established using DFAs. For other closure properties we will use different representations.

[2] The language $L := \{\, a^n b^m \mid \text{The } n\text{-th Turing machine holds within } m \text{ steps on input } \varepsilon \,\}$ is decidable, but the image of L under the homomorphism mapping a to a and b to ε is undecidable.

[3] In [24], $\hat{\delta}$ is defined recursively starting from the right end of the word. In Coq, structural recursion is more natural and the impact on the proofs appears to be minimal.

Lemma 4.4 *Let A_1 and A_2 be DFAs, and let $\odot : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$. Then one can construct DFAs accepting $\overline{\mathscr{L}(A_1)}$ and $\{\, x \mid x \in \mathscr{L}(A_1) \odot x \in \mathscr{L}(A_2) \,\}$.*

Proof We show the case for the binary boolean operation "$\odot$". Let $A_1 = (Q_1, s_1, F_1, \delta_1)$ and $A_2 = (Q_2, s_2, F_2, \delta_2)$. It is easy to verify that the DFA (Q, s, F, δ) where

$$Q := Q_1 \times Q_2 \qquad F := \{\, q \mid q \in F_1 \odot q \in F_2 \,\}$$
$$s := (s_1, s_2) \qquad \delta(p,q)\,a := (\delta_1\, p\, a, \delta_2\, q\, a)$$

accepts $\{\, x \mid x \in \mathscr{L}(A_1) \odot x \in \mathscr{L}(A_2) \,\}$. □

Lemma 4.5 *Let $L \subseteq \Gamma^*$ be regular and let $h : \Sigma^* \to \Gamma^*$ be a homomorphism. Then $h^{-1}(L)$ is regular.*

Proof Let $A = (Q, s, F, \delta)$ be a DFA accepting L. Then the DFA (Q, s, F, δ') with $\delta' p\, a := \hat{\delta}\, p\, (h\, a)$ accepts $h^{-1}(L)$. □

We now prove two criteria for nonregularity. The first criterion is based on residual languages. The reachable states of a DFA accept, up to equivalence, exactly the residual languages of its language. Consequently, a language with infinitely many nonequivalent residual languages cannot be regular.

Theorem 4.6 (Residual Criterion) *Let L be a language and let $f : \mathbb{N} \to \Sigma^*$ such that $n = m$ whenever $\mathscr{R}_L(f\, n) \equiv \mathscr{R}_L(f\, m)$. Then L is not regular.*

Proof Assume that L is regular and let $A = (Q, s, F, \delta)$ be a DFA accepting L. We define a function $g : \mathbb{N} \to Q$ as $g\, n := \hat{\delta}\, s\, (f\, n)$. Using the assumption on f, we obtain that g is injective. This is a contradiction since Q is finite. □

Example 4.7 Let $L = \{\, a^n b^n \mid n \in \mathbb{N} \,\}$. Then $f\, n := a^n$ satisfies the residual criterion.

The contrapositive of Theorem 4.6 is referred to as "Continuation Lemma" in [33].

Another criterion for nonregularity is the Pumping Lemma [31]. The intuition is that for every DFA the runs on sufficiently long words must include cycles that may be repeated by repeating the corresponding part of the input word.

Lemma 4.8 *Let A be a DFA and let $x, z, y : \Sigma^*$. If $xyz \in \mathscr{L}(A)$ and $|A| < |y|$, then there exist u, v, and w such that $y = uvw$, $v \neq \varepsilon$, and $xuv^iwz \in \mathscr{L}(A)$ for all $i : \mathbb{N}$.*

Proof Let $A = (Q, s, F, \delta)$ and $xyz \in \mathscr{L}(A)$ with $|A| < |y|$. Let $f(n) := \hat{\delta}\, (\hat{\delta}\, s\, x)\, (y[0, n])$. Since $|A| < |y|$, we obtain $j < k < |y|$ such that $f(j) = f(k)$. It is easy to verify that $u := y[0, j]$, $v := y[j, k]$ and $w := y[k, |y|]$ satisfy the required properties. □

Note that the argument in the proof of Lemma 4.8 does not make use of the notion of "runs" employed in the informal argument. While the notion of a run provides useful intuitions about DFAs, the formalization of our results for DFAs would not profit from making the concept explicit.

Lemma 4.9 (Pumping) *Let $L \subseteq \Sigma^*$. Then L is nonregular if for all $k \in \mathbb{N}$ there exist words $x, y, z : \Sigma^*$ such that $k \leq |y|$, $xyz \in L$, and for all u, v, w such that $y = uvw$ and $v \neq \varepsilon$ there exists some i such that $xuv^iwz \notin L$.*

We include the Pumping Lemma because of its ubiquity in the literature. For proofs of nonregularity, the Residual Criterion (Theorem 4.6) is usually easier to use. In particular, the Pumping Lemma is incomplete in the sense that there are examples of decidable nonregular languages (cf. [21]) that do not satisfy the precondition of Lemma 4.9.

5 Classifiers

We now introduce the notion of classifier. A classifier is a function that can be seen as representing a right-congruent partition of Σ^* with finite index. Right-congruence provides for a cut-off property that is useful for establishing decidabiliy properties. Moreover, classifiers refining a given language correspond to Myhill-Nerode relations [20,24]. Based on classifiers, we will establish a constructive Myhill-Nerode characterization of regularity.

Definition 5.1 We call a function $f : \Sigma^* \to Q$ *classifier* if Q is a finite type and f is *right-congruent*, i.e., $f(xa) = f(ya)$ whenever $fx = fy$.

Lemma 5.2 *Let $f : \Sigma^* \to Q$ be a classifier and let x be a word such that $|Q| < |x|$. Then there exists some word y such that $|y| < |x|$ and $f\,x = f\,y$.*

Proof Since $|Q| < |x|$, there exist i, j such that $i < j < |x|$ and $f(x[0,i]) = f(x[0,j])$. The claim then follows with right-congruence by setting $y := x[0,i] \cdot x[j,|x|]$. □

Theorem 5.3 (Cut-Off) *Let $f : \Sigma^* \to Q$ be a classifier and let $P : Q \to \mathsf{Prop}$. Then*

$$(\exists x.\, P(f\,x)) \;\leftrightarrow\; \exists x.\, |x| \leq |Q| \wedge P(f\,x)$$

Proof By complete induction on $|x|$ using Lemma 5.2. □

Corollary 5.4 *Let $f : \Sigma^* \to Q$ be a classifier. Then $\exists x.\, p(f\,x)$ and $\forall x.\, p(f\,x)$ are decidable for all decidable predicates $p : Q \to \mathbb{B}$.*

Proof Decidability of $\exists x.\, p(f\,x)$ follows with Theorem 5.3, since there are only finitely many words of length at most $|Q|$. Decidability of $\forall x.\, p(f\,x)$ then follows from decidability of $\exists x.\, \neg p(f\,x)$. □

Corollary 5.5 *Let $f : \Sigma^* \to Q$ be a classifier. Then the image of f can be constructed as a subtype of Q.*

Proof By Corollary 5.4, we have that $\exists x.\, f\,x = q$ is decidable for all q. Hence, we can construct the subtype $\{\, q : Q \mid \exists x.\, f\,x = q \,\}$. □

If $f : \Sigma^* \to Q$ is a classifier, we write $f(\Sigma^*)$ for the subtype of Q corresponding to the image of f.

We now extend the notion of classifier to classifiers for a given language and show that this provides another representation for regular languages.

Definition 5.6 Let L be a language. A *classifier for L* is a classifier that *refines L*, i.e., a classifier such that $(x \in L \leftrightarrow y \in L)$ whenever $f\,x = f\,y$.

Fact 5.7 *Let $A = (Q, s, F, \delta)$ be a DFA. Then $\hat{\delta} q$ is a classifier for the language accepted by q. In particular, $\hat{\delta} s$ is a classifier for $\mathscr{L}(A)$.*

Conversely, we can turn a classifier-decider pair for a language into a DFA.

Lemma 5.8 *Let L be decidable and let $f : \Sigma^* \to Q$ be a classifier for L. Then one can construct a DFA accepting L that has at most $|Q|$ states.*

Proof By casting the results of f from Q to $f(\Sigma^*)$, we obtain a surjective classifier $g : \Sigma^* \to f(\Sigma^*)$ for L (Corollary 5.5). Since g is surjective, it has a right inverse $\overline{g}$ (Lemma 2.1). It is straightforward to verify that the DFA (Q', s, F, δ) where

$$\begin{aligned} Q' &:= f(\Sigma^*) \\ s &:= g\,\varepsilon \\ F &:= \{\, q \mid \overline{g}\,q \in L \,\} \\ \delta\,q\,a &:= g((\overline{g}\,q) \cdot a) \end{aligned}$$

accepts the language L and has at most $|Q|$ states. □

Theorem 5.9 *A language is regular if and only if it has both a decider and a classifier.*

Remark 5.10 Classifiers can be seen as computational representations of Myhill-Nerode relations (i.e., right congruent relations of finite index that refine the language). Lemma 5.8 corresponds to the observation in [34] that the construction of DFAs from Myhil-Nerode relations is "effective" (or computational) whenever language membership is decidable and the number of equivalence classes is bounded. In addition to the upper bound on the number of equivalence classes (i.e., the size of Q), Lemma 5.8 includes (via the decidable equality on the finite type Q) the redundant assumption that the Myhill-Nerode relation is decidable. This simplifies both the statement and the proof and is sufficient for our purposes.

The cut-off property for classifiers (Theorem 5.3) allows us to easily establish decidability of language emptiness for DFAs.

Theorem 5.11 *1. Language (non-)emptiness for DFAs is decidable.*
2. Language inclusion and equivalence for DFAs are decidable.

Proof For (1), let $A = (Q, s, F, \delta)$ be a DFA. Then $\mathscr{L}(A)$ is nonempty iff $\exists x.\, \hat{\delta}\, s\, x \in F$ which is decidable since $\hat{\delta}\, s$ is a classifier (Corollary 5.4). Claim (2) follows with (1) and Lemma 4.4. □

Note that, constructively, decidability of non-emptiness (i.e., $\exists x. x \in L$) implies decidability of language emptiness (i.e., $L \equiv \emptyset$), but not the other way around. We remark that our proof of decidability of language emptiness is essentially the proof given in [31] with the cut-off property made explicit.

With decidability of language emptiness in place, we can show the closure of regular languages under quotients. For DFAs $A = (Q, s, F, \delta)$ and $q : Q$, we define $A[q] := (Q, q, F, \delta)$.

Lemma 5.12 *Let L_1 and L_2 be regular. Then L_1/L_2 is regular.*

Proof Let $A = (Q, s, F, \delta)$ be a DFA accepting L_1. We want to define a DFA $B = (Q, s, F', \delta)$ where $F' := \{\, q \mid \exists y \in L_2.\, \hat{\delta}\, q\, y \in F \,\}$. For this to be well-defined we need to be able to decide which states are final states. We have $\exists y \in L_2.\, \hat{\delta}\, q\, y \in F$ iff $L_2 \cap \mathscr{L}(A[q])$ is nonempty which is decidable (Lemma 4.4 and Theorem 5.11). □

Lemma 5.13 *Let L_1 and L_2 be regular. Then $L_1 \backslash L_2$ is regular.*

Proof Let $A = (Q, s, F, \delta)$ be a DFA accepting L_2 and let $S := \{\, q \mid \exists x \in L_1.\, \hat{\delta}\, s\, x = q \,\}$. Similar to the proof of Lemma 5.12, membership in S is decidable since $\exists x \in L_1.\, \hat{\delta}\, s\, x = q$ is equivalent to the non-emptiness of $L_1 \cap \mathscr{L}(B)$ where $B := (Q, s, \{q\}, \delta)$. It is straightforward to verify that $L_1 \backslash L_2 \equiv \bigcup_{q \in S} \mathscr{L}(A[q])$. Thus, $L_1 \backslash L_2$ is regular since regular languages are closed under union (Lemma 4.4). □

We remark that there are languages that can be shown regular using excluded middle but cannot be shown regular constructively. To see this, consider the constant language $L := \{x \mid P\}$ where P is some independent proposition (i.e., a proposition for which $P \vee \neg P$ is not provable constructively). Assuming $\forall P. P \vee \neg P$, one can easily show the existence of a DFA accepting L. However, even without excluded middle, the existence of a DFA accepting L would allow us to prove $\varepsilon \in L \vee \varepsilon \notin L$ and therefore also $P \vee \neg P$. Hence, we have:

Fact 5.14 $(\forall P : \mathsf{Prop}.\ P \vee \neg P) \leftrightarrow \forall P : \mathsf{Prop}.\ \exists A : \mathsf{dfa}. \mathscr{L}(A) \equiv \{w \mid P\}$.

Since the constant function from Σ^* to the unit type is a classifier for $\{x \mid P\}$, this also shows that the restriction to decidable languages in Lemma 5.8 is unavoidable in a constructive setting. For our use of Lemma 5.8 in the translation from 2NFAs to DFAs (cf. Section 11), the decidability condition is easily satisfied.

Similar to constant languages, one can show using excluded middle that L_1/L_2 is regular whenever L_1 is regular (for any language L_2). However, without a proper computational representation of L_2 it is impossible to determine which of the finitely many possible sets of final states is the correct one (cf. proof of Lemma 5.12). In contrast to Lemma 5.8, assuming decidabiliy of L_2 is not sufficient. To see this, consider the regular language $L_1 := \Sigma^*$ and the class of decidable languages $L_2(m) := \{a^n \mid \text{the } m\text{-th Turing machine holds within } n \text{ steps on input } \varepsilon\}$. Then $L_1/L_2(m)$ is nonempty iff the m-th Turing machine halts on input ε. If we could constructively show the existence of automata for all languages $L_1/L_2(m)$, this would give us, via the constructive interpretation of the logic, a method for deciding the halting problem. A sufficient requirement on the representation of L_2 is that non-emptiness of $L_2 \cap \mathscr{L}(A_q)$ is decidable. We formalize the case where L_2 is given by a DFA and remark that that a context-free grammar for L_2 would suffice.

6 Nondeterministic Finite Automata

Nondeterministic finite automata (NFAs) are another prominent representation of regular languages. Nondeterminism often allows for constructions that are simpler than directly constructing DFAs. We give conversions between DFAs and NFAs and then use NFAs to establish the closure of regular languages under concatenation and Kleene star. We will also use NFAs as one possible target for the reduction from two-way automata to one-way automata (Section 10) and to show decidability of WS1S (Section 12).

Nondeterministic finite automata differ from DFAs in that the transition function is replaced with a relation. Moreover, we allow multiple starting states.

Definition 6.1 A *nondeterministic finite automation (NFA)* is a structure $(Q, S, F, \rightarrow)$ where:
- Q is a finite type of states.
- $S : 2^Q$ is the set of *starting* states.
- $F : 2^Q$ is the set of *final* states.
- $\rightarrow : Q \rightarrow \Sigma \rightarrow Q \rightarrow \mathbb{B}$ is the *transition relation*.

We write $p \xrightarrow{a} q$ for the application of $\rightarrow$ to p, a, and q.

Let $A = (Q, S, F, \rightarrow)$ be an NFA. Similar to DFAs, we define acceptance for every state of an NFA by structural recursion on the input word.

$$\mathsf{accept}\, p\, \varepsilon := p \in F$$

$$\mathsf{accept}\, p\, (ax) := \exists q.\ p \xrightarrow{a} q \wedge \mathsf{accept}\, q\, x$$

The *language* of an NFA is the union of the languages accepted by its starting states.

$$\mathscr{L}(A) := \{x \in \Sigma^* \mid \exists s \in S.\, \mathsf{accept}\, s\, x\}$$

Note that since Q is finite, $\mathscr{L}(A)$ is a decidable language. As with DFAs, acceptance of languages is defined up to language equivalence.

Since we allow multiple starting states, the disjoint union of two NFAs yields an NFA for the union of the respective languages.

Lemma 6.2 *Let N be an n-state NFA and let M be an m-state NFA. Then one can construct an NFA with $n+m$ states that accepts $\mathscr{L}(N) \cup \mathscr{L}(M)$.*

Proof Let $N = (Q_1, S_1, F_1, \to_1)$ and $M = (Q_2, S_2, F_2, \to_2)$. The NFA for $\mathscr{L}(N) \cup \mathscr{L}(M)$ uses the sum type $Q_1 + Q_2$ as the type of states and $\{\mathsf{inl}\, q \mid q \in S_1\} \cup \{\mathsf{inr}\, q \mid q \in S_2\}$ as starting states. Final states and transition relation are defined analogously. □

We establish the equivalence between DFAs and NFAs using the powerset construction.

Lemma 6.3 *For every n-state NFA A, one can construct a DFA with 2^n states that accepts $\mathscr{L}(A)$.*

Proof Let $A = (Q, S, F, \to)$ be an NFA. Then the DFA (Q', s', F', δ') where

$$Q' := 2^Q \qquad F' := \{X \mid X \cap F \neq \emptyset\}$$
$$s' := S \qquad \delta'\, X\, a := \{q \mid \exists p \in X.\, p \xrightarrow{a} q\}$$

is a DFA accepting $\mathscr{L}(A)$. □

Since DFAs can easily be transformed to NFAs, we obtain:

Theorem 6.4 *A language is regular if and only if it is accepted by some NFA.*

One can work directly with NFAs to show the closure of regular languages under concatenation and iteration. Indeed, this is what we did in [11]. However, the proofs become simpler if one first extends NFAs with ε-transitions.

We define εNFAs like NFAs with additional transitions of the form $p \xrightarrow{\varepsilon} q$, i.e., transitions that do not consume a symbol from the input word. In contrast to NFAs, acceptance for εNFAs cannot be defined using a simple recursion on the input word due to possible sequences of ε-transitions of arbitrary length. Thus, we define acceptance for εNFAs as an inductive predicate. Let $N = (Q, S, F, \to)$ be an εNFA. The *acceptance relation* for N is defined inductively as follows:

$$\frac{p \in F}{\mathsf{accept}_\varepsilon\, p\, \varepsilon} \qquad \frac{p \xrightarrow{a} q \qquad \mathsf{accept}_\varepsilon\, q\, x}{\mathsf{accept}_\varepsilon\, p\, (a :: x)} \qquad \frac{p \xrightarrow{\varepsilon} q \qquad \mathsf{accept}_\varepsilon\, q\, x}{\mathsf{accept}_\varepsilon\, p\, x}$$

Note that, in contrast to NFAs, the language of an εNFA is – a priori – not a decidable language. However, decidability (and equivalence to NFAs) are easily established by removing ε-edges as shown below.

Lemma 6.5 *For every n-state εNFA N one can construct an n-state NFA accepting $\mathscr{L}(N)$.*

Proof Let $N = (Q, S, F, \to)$ be some εNFA. We define an NFA $N' = (Q, S', F, \rightsquigarrow)$ where

$$S' := \{\, q \mid \exists s \in S.\, s \xrightarrow{\varepsilon}{}^{*}\, q \,\}$$
$$p \overset{a}{\rightsquigarrow} q := \exists q'.\, p \xrightarrow{a} q' \wedge q' \xrightarrow{\varepsilon}{}^{*}\, q$$

The inclusion $\mathscr{L}(N') \subseteq \mathscr{L}(N)$ follows by induction on words. The converse inclusion follows by induction on the acceptance relation. □

Lemma 6.6 *Let N_1 and N_2 be NFAs. Then one can construct NFAs accepting $\mathscr{L}(N_1)^*$ and $\mathscr{L}(N_1) \cdot \mathscr{L}(N_2)$.*

Proof We show the case for $\mathscr{L}(N_1) \cdot \mathscr{L}(N_2)$. Let $N_1 = (Q_1, S_1, F_1, \to_1)$ and $N_2 = (Q_2, S_2, F_2, \to_2)$. By Lemma 6.5, it suffices to construct an εNFA. We define an εNFA $M = (Q, S, F, \to)$ where

$$\begin{aligned} Q &:= Q_1 + Q_2 & \mathsf{inl}\, p \xrightarrow{a} \mathsf{inl}\, q &:= p \xrightarrow{a}_1 q \\ S &:= \{\, \mathsf{inl}\, p \mid p \in S_1 \,\} & \mathsf{inr}\, p \xrightarrow{a} \mathsf{inr}\, q &:= p \xrightarrow{a}_2 q \\ F &:= \{\, \mathsf{inr}\, p \mid p \in F_2 \,\} & \mathsf{inl}\, p \xrightarrow{\varepsilon} \mathsf{inr}\, q &:= p \in F_1 \wedge q \in S_2 \end{aligned}$$

and no other transitions are possible. To show that M accepts $\mathscr{L}(N_1) \cdot \mathscr{L}(N_2)$ it suffices to show the following three properties:

1. $\forall p : Q_2.\, \mathsf{accept}\, p\, x \to \mathsf{accept}_\varepsilon\, (\mathsf{inr}\, p)\, x$
2. $\forall p : Q_1.\, \mathsf{accept}\, p\, x \to y \in \mathscr{L}(N_2) \to \mathsf{accept}_\varepsilon\, (\mathsf{inl}\, p)\, xy$
3. $\forall p : Q_1 + Q_2.$
$$\mathsf{accept}_\varepsilon\, p\, x \to \begin{cases} \mathsf{accept}\, q\, x & \text{if } p = \mathsf{inr}\, q \\ \exists yz.\, x = yz \wedge \mathsf{accept}\, q\, y \wedge z \in \mathscr{L}(N_2) & \text{if } p = \mathsf{inl}\, q \end{cases}$$

Claims (1) and (2) follow by induction on x where (1) provides the base case for (2). Claim (3) follows by induction on $\mathsf{accept}_\varepsilon\, p\, x$. □

The proof the lemma above avoids the need for the notion of a run through a simple generalization of the statement. For the translations from 2NFAs to NFAs (Section 10) and MSO formulas to NFAs (Section 13) we were unable to find such generalizations. For this reason, we also formalize the notion of (accepting) runs on NFAs.

Let $N = (Q, S, F, \to)$ be an NFA. We define a relation $\mathsf{run} : \Sigma^* \to Q \to Q^* \to \mathsf{Prop}$ relating words and nonempty sequences of states inductively as follows:

$$\frac{q \in F}{\mathsf{run}\, \varepsilon\, q\, []} \qquad\qquad \frac{p \xrightarrow{a} q \qquad \mathsf{run}\, x\, q\, l}{\mathsf{run}\, (ax)\, p\, (q :: l)}$$

An *accepting run for x* is a sequence of states $(s :: l)$ such that $s \in S$ and $\mathsf{run}\, x\, s\, l$. Note that accepting runs for x must have length $|x| + 1$. We write $(r_i)_{i \le |x|}$ for runs of length $|x| + 1$ and r_i for the i-th element (counting from 0).

Lemma 6.7 *$x \in \mathscr{L}(N)$ iff there exists an accepting run for x.*

7 Regular Expressions

In this section we establish the equivalence between finite automata and regular expressions, another prominent representation of regular languages. While finite automata can be seen as operational characterizations of regular languages, regular expressions provide a compositional (or algebraic) characterization of regular languages. We consider *regular expressions* with the following syntax:

$$e := \emptyset \mid \varepsilon \mid a \mid e \cdot e \mid e + e \mid e^* \qquad (a : \Sigma)$$

As with words, we usually omit the "·" operator. We write $|e|$ for the tree size of e. The *language* of a regular expression e, written $\mathscr{L}(e)$, is defined as follows:

$$\begin{aligned} \mathscr{L}(\emptyset) &:= \emptyset & \mathscr{L}(e_1 e_2) &:= \mathscr{L}(e_1) \cdot \mathscr{L}(e_2) \\ \mathscr{L}(\varepsilon) &:= \{\varepsilon\} & \mathscr{L}(e_1 + e_2) &:= \mathscr{L}(e_1) \cup \mathscr{L}(e_2) \\ \mathscr{L}(a) &:= \{a\} & \mathscr{L}(e^*) &:= \mathscr{L}(e)^* \end{aligned}$$

Lemma 7.1 *$\mathscr{L}(e)$ is a decidable language for every regular expression e.*

Proof Immediate with Lemma 3.1. □

Remark 7.2 In the Coq development, we directly assign decidable languages to regular expressions using the constructions underlying the proof of Lemma 3.1.

The main result of this section is that regular expressions describe exactly the regular languages. One direction is fairly straightforward.

Lemma 7.3 *For every regular expression e one can construct an NFA accepting $\mathscr{L}(e)$ that has at most $2 \cdot |e|$ states.*

Proof By induction on e. The base cases are straightforward, the remaining cases follow with Lemma 6.2 and the constructions underlying Lemma 6.6. □

We now translate DFAs to regular expressions. There are a number of algorithms for constructing regular expressions from automata, e.g., Brzozowski's algebraic method [5] and Kleene's algorithm [23,24]. We employ a variant of Kleene's algorithm for DFAs, because we deem it easiest to formalize.

For the rest of this section, we fix some DFA $A = (Q, s, F, \delta)$. We define an indexed collection of *transition languages* between states $p, q : Q$ as follows:

$$L^X_{p,q} := \{\, x \in \Sigma^* \mid \hat{\delta}\, p x = q, \forall 0 < i < |x|.\, \hat{\delta}\, p(x[0,i]) \in X \,\}$$

That is, $x \in L^X_{p,q}$ if starting from state p and reading x the automaton A ends in state q and visits only states from X in between. Note that in the particular case where X is Q, the requirement on the intermediate states is trivially fulfilled. Consequently, the language of A can be expressed as the union of the transition languages between the starting state and the final states.

Lemma 7.4 $\mathscr{L}(A) \equiv \bigcup_{q \in F} L^Q_{s,q}$

That is, to obtain a regular expression for $\mathscr{L}(A)$, it suffices to construct regular expressions for $L^{Q}_{s,q}$ where $q \in F$. We recursively solve this problem for all languages $L^{X}_{p,q}$ by successively removing states from X. For the case where $X = \emptyset$, one can directly give a regular expression. Let $p, q : Q$. We define:

$$R^{\emptyset}_{p,q} := (\text{if } p = q \text{ then } \varepsilon \text{ else } \emptyset) + \sum_{\substack{a \in \Sigma \\ \delta(p,a)=q}} a$$

Lemma 7.5 $\mathscr{L}\left(R^{\emptyset}_{p,q}\right) \equiv L^{\emptyset}_{p,q}$.

For nonempty sets of allowed intermediate states, consider some word x from $L^{\{r\}\cup X}_{p,q}$. Reading x starting from p, the automaton will either not visit r at all, or reach r, go through a number of cycles starting and ending at r, and then continue to q without visiting r again. This motivates the following recursive equivalence:

Lemma 7.6 *Let $p, q, r : Q$ and $X \subseteq Q$, then*

$$L^{\{r\}\cup X}_{p,q} \equiv L^{X}_{p,r} \cdot \left(L^{X}_{r,r}\right)^{*} \cdot L^{X}_{r,q} \cup L^{X}_{p,q}$$

Proof The inclusion from right to left is easy to show. For the other direction, we first show

$$\forall x.\, x \in L^{\{r\}\cup X}_{p,q} \to x \in L^{X}_{p,q} \vee \exists yz.\, x = yz \wedge y \in L^{X}_{p,r} \wedge z \in L^{\{r\}\cup X}_{r,q} \wedge |z| < |x| \qquad (*)$$

We consider two cases:

$\hat{\delta}\, p\, (x[0,i]) \neq r$ for all i such that $0 < i < |x|$. In this case, we have $x \in L^{X}_{p,q}$.

$\hat{\delta}\, p\, (x[0,i]) = r$ for some i with $0 < i < |x|$. Let i be the minimal i satisfying the condition. It is straightforward to verify that $x[0,i] \in L^{X}_{p,q}$ and $x[i,|x|] \in L^{\{r\}\cup X}_{r,q}$. Moreover, $|x[i,|x|]| < |x|$ since $0 < i$.

The inclusion from left to right then follows by complete induction on word length using (*). □

The construction of the regular expression accepting $\mathscr{L}(A)$ then follows the recursive structure given by Lemma 7.6. We define a function $\mathsf{reg} : Q^{*} \to Q \to Q \to \mathsf{regexp}$ by recursion on the list of states:

$$\begin{aligned} \mathsf{reg}\, []\, p\, q &:= R^{\emptyset}_{p,q} \\ \mathsf{reg}\, (r :: l)\, p\, q &:= (\mathsf{reg}\, l\, p\, r)(\mathsf{reg}\, l\, r\, r)^{*}(\mathsf{reg}\, l\, r\, q) + (\mathsf{reg}\, l\, p\, q) \end{aligned}$$

Lemma 7.7 *Let $l : Q^{*}$ and $p, q : Q$. Then $\mathscr{L}(\mathsf{reg}\, l\, p\, q) \equiv L^{\{r \mid r \in l\}}_{p,q}$.*

Theorem 7.8 *A language is regular iff it is accepted by some regular expression.*

Proof Let L be a regular language. Since A was chosen arbitrarily, we can assume that A accepts L. Hence, the claim follows with Lemmas 7.4 and 7.7 (taking l to be the list enumerating the finite type Q). The converse follows with Lemma 7.3 and Theorem 6.4. □

The equivalence between DFAs and regular expressions yields two closure properties of regular expressions that are difficult to prove without automata.[4]

[4] Wu et al. [42] derive the closure of regular expressions under complement by proving the Myhill-Nerode theorem using regular expressions. The proof is significantly more complex than the automata constructions.

Corollary 7.9 *Let e_1 and e_2 be regular expressions. Then one can construct regular expressions accepting $\mathscr{L}(e_1) \cap \mathscr{L}(e_2)$ and $\overline{\mathscr{L}(e_1)}$.*

Remark 7.10 The constructions presented in this section allow the definition of a complementation operation for regular expressions that causes a doubly-exponential increase in the size of the expression. The DFA for the complement language of some expression e can have up to $2^{2\cdot|e|}$ states and the construction underlying Theorem 7.8 yields expressions of size $O(|\Sigma| \cdot |Q| \cdot 4^{2\cdot|Q|})$ (see the Coq development for a precise bound). While this may seem expensive, it matches the doubly exponential lower bound for the complementation of regular expressions [15].

Using regular expressions, it is straightforward to show that regular languages are closed under images of homomorphisms and under word reversal.

Lemma 7.11 *Let $L \subseteq \Sigma^*$ be regular and let $h : \Sigma^* \to \Gamma^*$ be a homomorphism. Then $h(L)$ is regular.*

Proof Let e be a regular expression for L. Replacing all atoms a occurring in e with an expression accepting $\{h(a)\}$ yields a regular expression for $h(L)$. □

Lemma 7.12 *Let L be regular. Then $\{x \mid \mathsf{rev}\, x \in L\}$ is regular.*

Proof Let e such that $\mathscr{L}(e) \equiv L$. It is straightforward to compute a regular expression for $\{x \mid \mathsf{rev}\, x \in L\}$ by reversing all sequential compositions $e_1 e_2$ in e. □

Remark 7.13 Closure under word reversal can also be shown using NFAs. However, both the direct structural acceptance criterion and the notion of accepting run (Section 6) are asymmetric in that they always need to reach final states. To show closure under word reversal using NFAs, we would have to generalize the notion of run to runs between any pair of states. The proof above requires no such generalization.

Together with the closure properties established in Sections 4 and 6, we obtain:

Theorem 7.14 (Closure Properties) *Let L_1 and L_2 be regular. Then $\overline{L_1}$, L_1^*, $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 \cdot L_2$, $h(L)$, $h^{-1}(L)$, L_1/L_2, $L_1 \backslash L_2$ and $\{x \mid \mathsf{rev}\, x \in L\}$ are regular.*

Note that, all closure properties are established as operations on the underlying representations of the input languages.

8 DFA Minimization

In this section, we show that for every regular language there exists a unique minimal automaton (cf. [24]). The proof is based on a simple minimization function that prunes unreachable states and collapses states that accept the same language.

Definition 8.1 Let $A = (Q, s, F, \delta)$ be a DFA. We call A

- *connected* if every state is reachable, i.e. $\forall q : Q\, \exists x : \Sigma^*.\, \hat{\delta}\, s\, x = q$.
- *collapsed* if no distinct states $p, q : Q$ are related by the *collapsing relation*

$$p \approx_A q := \forall x.\ \hat{\delta}\, p\, x \in F \leftrightarrow \hat{\delta}\, q\, x \in F$$

- *minimal* if every DFA accepting $\mathscr{L}(A)$ has at least $|Q|$ states.

The collapsing relation relates states that accept the same language and can therefore be merged. It turns out that an automaton is minimal iff if it is connected and collapsed. We first show that all connected and collapsed automata for a given language are isomorphic and therefore have the same size.

Definition 8.2 Let $A_1 = (Q_1, s_1, F_1, \delta_1)$ and $A_2 = (Q_2, s_2, F_2, \delta_2)$ be DFAs. We call A_1 and A_2 *isomorphic*, if there exists a a bijection $i : Q_1 \to Q_2$ satisfying

$$\begin{aligned} \forall q : Q_1 .\, i(\delta_1\, q\, a) &= \delta_2\, (i\, q)\, a \\ \forall q : Q_1 .\, i\, q \in F_2 &\leftrightarrow q \in F_1 \\ i\, s_1 &= s_2 \end{aligned}$$

Theorem 8.3 *Let A_1 and A_2 be connected and collapsed DFAs accepting the same language. Then A_1 and A_2 are isomorphic.*

Proof Let $A_1 = (Q_1, s_1, F_1, \delta_1)$ and $A_2 = (Q_2, s_2, F_2, \delta_2)$. Since A_1 and A_2 are connected, $\hat{\delta}_1 s : \Sigma^* \to Q_1$ and $\hat{\delta}_2 s : \Sigma^* \to Q_2$ are both surjective. By Lemma 2.1, we can define

$$i := \left(\hat{\delta}_2\, s\right) \circ \left(\overline{\hat{\delta}_1\, s}\right) \qquad\qquad j := \left(\hat{\delta}_1\, s\right) \circ \left(\overline{\hat{\delta}_2\, s}\right)$$

Since A and B are also collapsed, we have $i(j\, p) = p$ and $j(i\, q) = q$ for all p and q. Thus, $i : Q_1 \to Q_2$ is a bijection. The remaining properties are easy to verify. □

A straightforward consequence of Theorem 8.3 is the fact that every function that establishes connectedness and collapsedness without increasing the number of states is a correct minimization function.

Lemma 8.4 *Let $F : \mathsf{dfa} \to \mathsf{dfa}$ such that for all A we have $\mathscr{L}(FA) \equiv \mathscr{L}(A)$, $|FA| \leq |A|$, and FA is connected and collapsed. Then for all A, FA is minimal (and accepts $\mathscr{L}(A)$).*

Proof Let B be some DFA accepting $\mathscr{L}(F\, A)$. Then $|FA| = |FB| \leq |B|$. □

We now construct a minimization function by defining two operations on DFAs, called prune and collapse, that preserve the language and respectively establish connectedness and collapsedness.

For prune we construct the sub-automaton whose type of states is the subtype of reachable states. Recall that for every DFA (Q, s, F, δ) the function $\hat{\delta}\, s$ is a classifier. Consequently, the pruning function can be defined as follows (cf. Corollary 5.5).

$$\begin{aligned} \mathsf{prune}\,(Q, s, F, \delta) &:= (Q', s, F', \delta) \qquad \text{where} \\ Q' &:= (\hat{\delta}\, s)(\Sigma^*) \\ F' &:= \{\, q \in Q' \mid q \in F \,\} \end{aligned}$$

Remark 8.5 For the definition in Coq we also need to cast s from Q to Q' and lift δ to a function of type $Q' \to \Sigma \to Q'$. This is straightforward since s is clearly reachable and transitions preserve reachability.

Lemma 8.6 prune$A$ *is connected, accepts $\mathscr{L}(A)$ and has at most $|A|$ states.*

To define the function collapse, we make use of quotient types. A quotient type for some type Q and an equivalence relation $\approx$ on Q is a type $Q/{\approx}$ together with a surjective function $\pi : Q \to Q/{\approx}$ satisfying $\pi p = \pi q \leftrightarrow p \approx q$. If Q is countable and $\approx$ is decidable, then $Q/{\approx}$ can be constructed as a subtype of Q [7]. In this case π also has a right inverse $\overline{\pi}$ (Lemma 2.1).

To show decidability of the collapsing relation, we again make use of Corollary 5.4.

Lemma 8.7 *Let A be a DFA. Then $\approx_A$ is decidable.*

Proof Let $A = (Q, s, F, \delta)$ and let $q, r : Q$. Decidability of $q \approx_A r$ follows with Corollary 5.4 by taking $f\,x := (\hat{\delta}\,q\,x, \hat{\delta}\,r\,x)$ and $p := \lambda(q,r).\, q \in F \leftrightarrow r \in F$. □

The function collapse takes as input some automaton A and returns the quotient automaton with respect to the collapsing relation.

$$\begin{aligned}
\mathsf{collapse}A &:= (Q', s', F', \delta) \qquad \text{where } A = (Q, s, F, \delta) \text{ and} \\
Q' &:= Q/{\approx_A} \\
s' &:= \pi\, s \\
F' &:= \{\, q \mid \overline{\pi}\, q \in F \,\} \\
\delta' q\, a &:= \pi(\delta\,(\overline{\pi}\, q)\, a)
\end{aligned}$$

Lemma 8.8 *Let A be a connected DFA. Then* collapse$A$ *accepts $\mathscr{L}(A)$, is collapsed and connected, and has at most $|A|$ states.*

We obtain a minimization function by defining minimize := collapse ∘ prune.

Theorem 8.9 *Let A be a DFA, then* minimize$A$ *accepts $\mathscr{L}(A)$ and is connected, collapsed, and minimal.*

Proof Immediate with Lemmas 8.6, 8.8, and 8.4. □

Theorem 8.10 *Let A be a DFA. Then A is minimal iff A is connected and collapsed.*

Proof Assume A is is minimal. Then both prune and collapse preserve the number of states. Hence, all states must be reachable and $\approx_A$ must be an identity relation. For the converse direction, assume A is connected and collapsed and fix some DFA B with $\mathscr{L}(A) = \mathscr{L}(B)$. Then $|A| = |\mathsf{minimize}B| \leq |B|$ by Theorems 8.3 and 8.9 □

Corollary 8.11 *All minimal automata accepting a given language are isomorphic.*

Note that the premise $|FA| \leq |A|$ of Lemma 8.4 becomes redundant with Theorem 8.10. However, the relevant direction of the proof of Theorem 8.10 relies on the fact that the function minimize establishes connectedness and collapsedness without increasing the number of states.

We remark that the classifier obtained from a minimal DFA corresponds to the coarsest Myhill-Nerode relation [20,24] (cf. Remark 5.10).

Fact 8.12 *If $A = (Q, s, F, \delta)$ is a minimal DFA for L, then $\hat{\delta}\,s\,x = \hat{\delta}\,s\,y$ iff $\mathscr{R}_L(x) \equiv \mathscr{R}_L(y)$.*

9 Two-Way Finite Automata

A two-way finite automaton (2FA) is essentially a read-only Turing machine, i.e., a machine with a finite state control and a read head that may move back and forth on the input word. One of the fundamental results about 2FAs is that the ability to move back and forth does not increase expressiveness [31]. That is, two-way automata are yet another representation of the class of regular languages. As for one-way automata, we consider both the deterministic and the nondeterministic variant.

In the literature, two-way automata appear in a number of variations. Modern accounts of two-way automata [29] usually consider automata with *end-markers*. That is, on input x the automaton is run on the string $\triangleright x \triangleleft$, where $\triangleright$ and $\triangleleft$ are marker symbols that do not occur in Σ and allow the automaton to detect the word boundaries. These marker symbols are not present in early work on two-way automata [31,34,39]. The ability to detect word boundaries allows for the construction of more compact automata for some languages. In fact, the emptiness problem for nondeterministic two-way automata with only one endmarker over a singleton alphabet is polynomial while the corresponding problem for two-way automata with two endmarkers is NP-complete [40]. Remarkably, the original proofs in [34,39] carry over to the setting with end-markers without conceptual changes.

Definition 9.1 A *nondeterministic two-way automaton (2NFA)* is a structure $M = (Q, s, F, \delta, \delta_\triangleright, \delta_\triangleleft)$ where
- Q is a finite type of *states*
- $s : Q$ is the *starting state*
- $F : 2^Q$ is the set of *final states*
- $\delta : Q \to \Sigma \to 2^{Q \times \{\mathsf{L},\mathsf{R}\}}$ is the *transition function* for symbols
- $\delta_\triangleright : Q \to 2^Q$ is the *transition function* for the left marker
- $\delta_\triangleleft : Q \to 2^Q$ is the *transition function* for the right marker

Let $M = (Q, s, F, \delta, \delta_\triangleright, \delta_\triangleleft)$ be a 2NFA. If $(q, \mathsf{R}) \in \delta\, p\, a$, this means that when reading the symbol a while in state p, the automaton may enter state q and move its read head one position to the right. For end-markers, we only allow the head to move back onto the word (or to the opposing marker if the input is ε), so no direction needs to be given.

On an input word $x : \Sigma^*$ the *configurations of M on x*, written C_x, are pairs $(p, i) \in Q \times \{0, \ldots, |x| + 1\}$ where i is the position of the read head. We take $i = 0$ to mean that the head is on the left marker and $i = |x| + 1$ to mean that the head is on the right marker. Otherwise, the head is on the i-th symbol of x (counting from 1). In particular, we do not allow the head to move beyond the end-markers. In following, we write $x[i]$ for the i-th symbol of x. The *step relation* $\to_x : C_x \to C_x \to \mathbb{B}$ updates state and head position according to the transition function for the current head position:

$$\dot{\delta}\, p\, i := \begin{cases} (\delta_\triangleright\, p) \times \{R\} & i = 0 \\ \delta\, p\, (x[i]) & 0 < i \leq |x| \\ (\delta_\triangleleft\, p) \times \{L\} & i = |x| + 1 \end{cases}$$

$$(p, i) \underset{x}{\to} (q, j) := (q, \mathsf{L}) \in \dot{\delta}\, p\, i \wedge i = j + 1 \;\vee\; (q, \mathsf{R}) \in \dot{\delta}\, p\, i \wedge i + 1 = j$$

We write $\to_x^*$ for the reflexive transitive closure of $\to_x$. The *language* of M is then defined as follows:

$$\mathscr{L}(M) := \{\, x \mid \exists q \in F.\, (s, 1) \underset{x}{\to}^* (q, |x| + 1) \,\}$$

That is, M accepts the word x if it can reach the right end-marker while being in a final state.

In Coq, we represent C_x as the finite type $Q \times \mathsf{ord}(|x|+2)$, where $\mathsf{ord}\, n := \{\, m : \mathbb{N} \mid m < n \,\}$. This allows us to represent $\to_x$ as well as $\to_x^*$ as decidable relations on C_x.[5] Hence, $\mathscr{L}(M)$ is a decidable language. In the mathematical presentation, we treat $\mathsf{ord}\, n$ like $\mathbb{N}$ and handle the bound implicitly. In Coq, we use a conversion function $\mathsf{inord} : \forall n.\, \mathbb{N} \to \mathsf{ord}(n+1)$ which behaves like the 'identity' on numbers in the correct range and otherwise returns 0. This allows us to sidestep most of the issues arising from the dependency of the type of configurations on the input word. In particular, we recover the separation between stating facts about $\to_x$ and proving that all mentioned indices stay within bounds.

Definition 9.2 A *deterministic two-way automaton (2DFA)* is a 2NFA $(Q, s, F, \delta, \delta_\triangleright, \delta_\triangleleft)$ where $|\delta_\triangleleft\, q| \leq 1$, $|\delta_\triangleright\, q| \leq 1$, and $|\delta\, q\, a| \leq 1$ for all $q : Q$ and $a : \Sigma$.

Fact 9.3 *For every n-state DFA one can construct an n-state 2DFA that accepts the same language and only moves its head to the right.*

Remark 9.4 While Fact 9.3 is obvious from the mathematical point of view, the formal proof is somewhat cumbersome due to the mismatch between the acceptance condition for DFAs, which is defined by recursion on the input word, and the acceptance condition for 2FAs, where the word remains constant throughout the computation.

The next two sections are devoted to the translation of two-way automata to one-way automata. There are several such translations in the literature. Vardi [39] gives a simple construction that takes as input some 2NFA M and yields an NFA accepting $\overline{\mathscr{L}(M)}$. This establishes that deterministic and nondeterministic two-way automata accept exactly the regular languages. The size of the constructed NFA is exponential in the size of M. Consequently, if one wants to obtain an automaton for the input language, rather than its complement, the construction incurs a doubly exponential blowup in the number of states. Shepherdson [34] gives a translation from 2DFAs to DFAs that incurs only an exponential blowup. Building on ideas from [39], we adapt this construction to 2NFAs.

We first present the translation to NFAs since it is much simpler. We then give a direct translation from 2NFAs to DFAs. We also show that when applied to 2DFAs, the latter construction yields the bounds on the size of the constructed DFA established in [34].

10 Vardi Construction

Let $M = (Q, s, F, \delta, \delta_\triangleright, \delta_\triangleleft)$ be a 2NFA. We construct an NFA accepting $\overline{\mathscr{L}(M)}$. Vardi [39] formulates the proof for 2NFAs without markers. We adapt the proof to 2NFAs with markers. The main idea is to define certificates for the non-acceptance of a string x by M. The proof then consists of two parts:

1. proving that these negative certificates are sound and complete
2. constructing an NFA whose accepting runs correspond to negative certificates

Definition 10.1 A *negative certificate* for a word x is a set $\mathscr{C} \subseteq C_x$ satisfying:
N1. $(s, 1) \in \mathscr{C}$
N2. If $(p, i) \in \mathscr{C}$ and $(p, i) \to_x (q, j)$, then $(q, j) \in \mathscr{C}$.
N3. If $q \in F$ then $(q, |x|+1) \notin \mathscr{C}$.

[5] That the transitive closure of a decidable relation is decidable is established in the Ssreflect libraries using depth-first search.

The first two conditions ensure that the negative certificates for x overapproximate the configurations M can reach on input x. The third condition ensures that no accepting configuration is reachable.

Lemma 10.2 *Let $x : \Sigma^*$. There exists a negative certificate for x iff $x \notin \mathscr{L}(M)$.*

Proof Let $R := \{\, (q,j) \mid (s,1) \to_x^* (q,j) \,\}$. If there exists a negative certificate $\mathscr{C}$ for x, then $R \subseteq \mathscr{C}$ and, therefore, $x \notin \mathscr{L}(M)$. Conversely, if $x \notin \mathscr{L}(M)$, then R is a negative certificate for x. □

Let x be a word and let $\mathscr{C}$ be a negative certificate for x. The certificate $\mathscr{C}$ can be viewed as $(|x|+2)$-tuple over 2^Q where the i-th component, written $\mathscr{C}_i$, is the set $\{\, q \mid (q,i) \in \mathscr{C} \,\}$.

We define an NFA whose accepting runs correspond to this tuple view of negative certificates. For this, condition (N2) needs to be rephrased into a collection of local conditions, i.e., conditions that no longer mention the head position.

Definition 10.3 Let $U,V,W : 2^Q$ and $a : \Sigma$. We say that
- (U,V) is $\triangleright$-*closed* if $q \in V$ whenever $p \in U$ and $q \in \delta_\triangleright\, p$.
- (U,V) is $\triangleleft$-*closed* if $q \in U$ whenever $p \in V$ and $q \in \delta_\triangleleft\, p$.
- (U,V,W) is *a-closed* if for all $p \in V$ we have
  1. $q \in U$ whenever $(q,\mathsf{L}) \in \delta\, p\, a$
  2. $q \in W$ whenever $(q,\mathsf{R}) \in \delta\, p\, a$

We define an NFA $N = (Q', S', F', \to)$ that incrementally checks the closure conditions defined above:

$$\begin{aligned} Q' &:= 2^Q \times 2^Q \\ S' &:= \{(U,V) \mid s \in V \text{ and } (U,V) \text{ is } \triangleright\text{-closed}\} \\ F' &:= \{(U,V) \mid F \cap V = \emptyset \text{ and } (U,V) \text{ is } \triangleleft\text{-closed}\} \\ (U,V) \xrightarrow{a} (V',W) &:= (V = V' \wedge (U,V,W) \text{ is } a\text{-closed}) \end{aligned}$$

For $q : Q'$, we write $q.1$ for the first component of q and $q.2$ for the second component. Note that transition relation requires the two states to overlap. Hence, the runs of N on some word x, which consist of $|x|$ transitions, define $(|x|+2)$-tuples. We will show that the accepting runs of N correspond exactly to negative certificates.

Lemma 10.4 *$x \in \mathscr{L}(N)$ iff there exists a negative certificate for x.*

Proof By Lemma 6.7, it suffices to show that there exists an accepting run iff there exists a negative certificate.

"$\Rightarrow$" Let $(r_i)_{i \le |x|}$ be an accepting run of N on x. We define a negative certificate $\mathscr{C}$ for x where $\mathscr{C}_0 := (r_0).1$ and $\mathscr{C}_{i+1} := (r_i).2$.

"$\Leftarrow$" If $\mathscr{C}$ is a negative certificate for x we can define a run $(r_i)_{i \le |x|}$ for x on M where $r_0 := (\mathscr{C}_0, \mathscr{C}_1)$ and $r_{i+1} := (\mathscr{C}_i, \mathscr{C}_{i+1})$. □

Remark 10.5 The formalization of the lemma above is straightforward but tedious due to the need to unfold the comparatively intricate definitions of N, $\mathscr{C}$, and $(r_i)_{i \le |x|}$. With about 60 lines, the proof of Lemma 10.4 is considerably longer than the proofs of most other lemmas.

Lemma 10.6 $\mathscr{L}(N) = \overline{\mathscr{L}(M)}$.

Proof Follows immediately with Lemmas 10.2 and 10.4. □

Theorem 10.7 *For every n-state 2NFA M one can construct an NFA accepting $\overline{\mathscr{L}(M)}$ and having at most 2^{2n} states.*

If one wants to obtain a DFA for $\mathscr{L}(M)$ using this construction, one needs to determinize N before complementing it. Since N is already exponentially larger than M, the resulting DFA then has a size that is doubly exponential in $|Q|$.

We remark that, perhaps surprisingly, the translation from 2NFAs to NFAs for the complement language becomes simpler and more 'symmetric' in the presence of end-markers. The original construction [39] uses $2^Q + 2^Q \times 2^Q$ as the type of states while the construction above gets along with the type $2^Q \times 2^Q$. States from the type 2^Q are required to check beginning and end of a negative certificate in the absence of end-markers.

11 Shepherdson Construction

We now give a second proof that the language accepted by a 2NFA is regular. The proof follows the original proof of Shepherdson [34]. In [34], the proof is given for 2DFAs without end-markers. Building on ideas form Vardi [39], we adapt the proof to 2NFAs with end-markers.

We fix some 2NFA $M = (Q, s, F, \delta, \delta_{\rhd}, \delta_{\lhd})$ for the rest of this section. In order to construct a DFA for $\mathscr{L}(M)$, it suffices to construct a classifier for $\mathscr{L}(M)$ (Lemma 5.8). For this, we need to come up with a finite type X and a function $T : \Sigma^* \to X$ which is right-congruent and refines $\mathscr{L}(M)$.

The construction exploits that the input is read-only. Therefore, M can only save a finite amount of information in its finite state control. Consider the situation where M is running on a composite word xz. In order to accept xz, M must move its head all the way to the right. In particular, it must move the read-head beyond the end of x and there is a finite set $S \subseteq Q$ of states that M can possibly be in when this happens for the first time. Once the read head is to the right of x, M may move its head back onto x. However, the only additional information that can be gathered about x is the set of states M may be in when returning to z. Since the possible states upon return may depend on the state M is in when entering x form the right, this defines a relation $R \subseteq Q \times Q$. The set S and the relation R provide all the information required about x to determine whether $xz \in \mathscr{L}(M)$. For every word x, S and R (as defined above) define a finite table. We will define a function

$$T : \Sigma^* \to 2^Q \times 2^{Q \times Q}$$

returning the table for a given word. Note that $2^Q \times 2^{Q \times Q}$ is a finite type, i.e., there are only finitely many possible tables. We will show that $\mathscr{L}(M)$ is regular, by showing that T is a classifier for $\mathscr{L}(M)$, i.e., that T is right-congruent and refines $\mathscr{L}(M)$.

To formally define T, we need to be able to stop M once its head reaches a specified position. We define the *k-stop relation on x*:

$$(p,i) \xrightarrow[x]{k} (q,j) := (p,i) \underset{x}{\to} (q,j) \wedge i \neq k$$

Note that for $k \geq |x| + 2$ the stop relation coincides with the step relation. The function T is then defined as follows:

$$\begin{aligned} T\,x := (\{ \quad q \quad \mid (s,1) \xrightarrow[x]{|x|+1}{}^{*} (q, |x|+1) \}, \\ \{ (p,q) \mid (p,|x|) \xrightarrow[x]{|x|+1}{}^{*} (q, |x|+1) \}) \end{aligned}$$

Note that T returns a pair of a set and a relation. We write $(T\,x).1$ for the first component of $T\,x$ and $(T\,x).2$ for the second component.

Before we can show that T is a classifier for $\mathscr{L}(M)$, we need a number of properties of the stop relation. The first lemma captures the intuition, that for composite words xz, all the information M can gather about x is given by $T\,x$.

Lemma 11.1 *Let $p,q : Q$ and let $x,z : \Sigma^*$. Then*

1. $q \in (T\,x).1$ *iff* $(s,1) \xrightarrow[xz]{|x|+1}{}^* (q,|x|+1)$
2. $(p,q) \in (T\,x).2$ *iff* $(p,|x|) \xrightarrow[xz]{|x|+1}{}^* (q,|x|+1)$

Since for composite words xz everything that can be gathered about x is provided by $T\,x$, M behaves the same on xz and yz whenever $T\,x = T\,y$. To show this, we need to exploit that M moves its head only one step at a time. This is captured by the lemma below.

Lemma 11.2 *Let $i \leq k \leq j$ and let l be a $\xrightarrow[x]{k'}$-path from (p,i) to (q,j). Then there exists some state p' such that l can be split into a $\xrightarrow[x]{k}$-path from (p,i) to (p',k) and a $\xrightarrow[x]{k'}$-path from (p',k) to (q,j).*

Proof By induction on the length of the $\xrightarrow[x]{k'}$-path from (p,i) to (q,j).

Lemma 11.2 can be turned into an equivalence if $k' \geq k$. We state this equivalence in terms of transitive closure since for most parts of the development the concrete path is irrelevant.

Lemma 11.3 *Let $i \leq k \leq j$ and let $k' \geq k$. Then $(p,i) \xrightarrow[x]{k'}{}^* (q,j)$ iff there exists some p' such that $(p,i) \xrightarrow[x]{k}{}^* (p',k) \xrightarrow[x]{k'}{}^* (q,j)$.*

We now show that for runs of M that start and end on the right part of a composite word xz, x can be replaced with y whenever $T\,x = T\,y$.

Lemma 11.4 *Let $p,q : Q$ and let $x,y,z : \Sigma^*$ such that $T\,x = T\,y$. Then for all $k > 1$, $i \leq |z|+1$, and $1 \leq j \leq |z|+1$, we have*

$$(p,|x|+i) \xrightarrow[xz]{|x|+k}{}^* (q,|x|+j) \quad \textit{iff} \quad (p,|y|+i) \xrightarrow[yz]{|y|+k}{}^* (q,|y|+j)$$

Proof By symmetry, it suffices to show the direction from left to right. We proceed by induction on the length of the path from $(p,|x|+i)$ to $(q,|x|+j)$. There are two cases to consider:

$i = 0$. According to Lemma 11.2 the path can be split such that:

$$(p,|x|) \xrightarrow[xz]{|x|+1}{}^* (p',|x|+1) \xrightarrow[xz]{|x|+k}{}^* (q,|x|+j)$$

Thus, $(p,p') \in (T\,x).2$ by Lemma 11.1. Applying Lemma 11.1 again, we obtain

$$(p,|y|) \xrightarrow[yz]{|y|+1}{}^* (p',|y|+1)$$

The claim then follows by induction hypothesis since the path from $(p,|x|)$ to $(p',|x|+1)$ must make at least one step.

$i > 0$. The path from $(p, |x| + i)$ to $(q, |x| + j)$ is either trivial and the claim follows immediately or or there exist p' and i' such that $(p, |x| + i) \xrightarrow[xz]{|x|+k} (p', |x| + i')$. But then $(p, |y| + i) \xrightarrow[yz]{|y|+k} (p', |y| + i')$ and the claim follows by induction hypothesis. □

Now we have everything we need to show that T is a classifier for $\mathscr{L}(M)$.

Lemma 11.5 *T refines $\mathscr{L}(M)$.*

Proof Fix $x, y : \Sigma^*$ and assume $T\,x = T\,y$. By symmetry, it suffices to show $y \in \mathscr{L}(M)$ whenever $x \in \mathscr{L}(M)$. If $x \in \mathscr{L}(M)$, then $(s, 1) \xrightarrow[x]{|x|+2}{}^* (p, |x| + 1)$ for some $p \in F$. We show $y \in \mathscr{L}(M)$ by showing $(s, 1) \xrightarrow[y]{|y|+2}{}^* (p, |y| + 1)$. By Lemma 11.3, there exists a state q such that:

$$(s, 1) \xrightarrow[x]{|x|+1}{}^* (q, |x| + 1) \xrightarrow[x]{|x|+2}{}^* (p, |x| + 1)$$

We can simulate the first part on y using Lemma 11.1 and the second part using Lemma 11.4. □

Lemma 11.6 *T is right-congruent.*

Proof Fix words $x, y : \Sigma^*$ and some symbol $a : \Sigma$ and assume $T\,x = T\,y$. We need to show $T\,xa = T\,ya$. We first show $(T\,xa).2 = (T\,ya).2$. Let $(p, q) \in Q \times Q$. We have to show that

$$(p, |xa|) \xrightarrow[xa]{|xa|+1}{}^* (q, |xa| + 1) \quad \text{implies} \quad (p, |ya|) \xrightarrow[ya]{|ya|+1}{}^* (q, |ya| + 1)$$

Since $|xa| + 1 = |x| + 2$ this follows immediately with Lemma 11.4. It remains to show $(T\,xa).1 = (T\,ya).1$. By symmetry, it suffices to show that

$$(s, 1) \xrightarrow[xa]{|xa|+1}{}^* (q, |xa| + 1) \quad \text{implies} \quad (s, 1) \xrightarrow[ya]{|ya|+1}{}^* (q, |ya| + 1)$$

By Lemma 11.3, there exists a state p such that:

$$(s, 1) \xrightarrow[xa]{|x|+1}{}^* (p, |x| + 1) \xrightarrow[xa]{|xa|+1}{}^* (q, |xa| + 1)$$

Thus, we have $p \in (T\,x).1$ (and therefore also $p \in (T\,y).1$) and $(p, q) \in (T\,xa).2$. Since we have shown above that $(T\,xa).2 = (T\,ya).2$, the claim follows with Lemma 11.1. □

Note that Lemma 11.4 is used very differently in the proofs of Lemma 11.5 and Lemma 11.6. In the first case we are interested in acceptance and set k to $|x| + 2$ so we never actually stop. In the second case we set k to $|xa| + 1$ to stop on the right marker.

Using Lemma 5.8 and the two lemmas above, we obtain:

Theorem 11.7 *Let M be a 2NFA with n states. Then one can construct a DFA accepting $\mathscr{L}(M)$ having at most 2^{n^2+n} states.*

We now show that for deterministic two-way automata, the bound on the size of the constructed DFA can be improved from 2^{n^2+n} to $(n+1)^{(n+1)}$.

Fact 11.8 *Let $M = (Q, s, F, \delta, \delta_\triangleright, \delta_\triangleleft)$ be a 2DFA. Then $\xrightarrow[x]{k}$ is functional for all k and x.*

Corollary 11.9 *Let M be a 2DFA with n states. Then one can construct a DFA accepting $\mathscr{L}(M)$ having at most $(n+1)^{(n+1)}$ states.*

Proof Let $M = (Q, s, F, \delta, \delta_{\triangleright}, \delta_{\triangleleft})$ be deterministic and let $T : \Sigma^* \to 2^Q \times 2^{Q \times Q}$ be defined as above. Since T is right-congruent (Lemma 11.6) we can construct the type $T(\Sigma^*)$ (Corollary 5.5). By Lemma 5.8, it suffices to show that $T(\Sigma^*)$ has at most $(|Q|+1)^{(|Q|+1)}$ elements.

Let $(S,R) : T(\Sigma^*)$. Then $T\,x = (S,R)$ for some $x : \Sigma^*$. We show that S has at most one element. Assume $p, q \in S$. By the definition of T, we have

$$(s,1) \xrightarrow[x]{|x|+1}{}^{*} (p, |x|+1) \qquad \text{and} \qquad (s,1) \xrightarrow[x]{|x|+1}{}^{*} (q, |x|+1)$$

Since $\xrightarrow[x]{|x|+1}$ is functional and both $(p, |x|+1)$ and $(p, |x|+1)$ are terminal, we have $p = q$. A similar argument yields that R is a functional relation. Consequently, we can construct an injection

$$i : T(\Sigma^*) \to Q_\perp \times (Q_\perp)^Q$$

Given some $(S,R) : T(\Sigma^*)$, $(i(S,R)).1$ is defined to be the unique element of S or $\perp$ if $S = \emptyset$. The definition of $(i(S,R)).2$ is analogous. The claim then follows since $Q_\perp \times (Q_\perp)^Q$ has exactly $(|Q|+1)^{(|Q|+1)}$ elements. □

12 Weak Monadic Second-Order Logic

We now consider weak monadic second order logic (WMSO), or more precisely the WMSO theory of one successor relation (WS1S). Our main results about WS1S are the decidability of satisfiability and model checking and a proof of the characterization theorem of Büchi [6], Elgot [13], and Trakhtenbrot [37] ([17, Theorem 12.26]), i.e., the fact that formulas can be assigned languages in such a way that the class of languages described by formulas is exactly the class of regular languages.

The aforementioned results are all closely related. The decidability results are obtained by translating formulas to NFAs. Together with the closure or regular languages under preimages of homomorphisms, this translation also yields an automaton for the language associated to a given formula. To show that formulas can express all regular languages, we give a formula describing the runs of a given automaton. Here, we rely on the fact that the the satisfaction relation is decidable as this ensures that the logic behaves classically even though we are working in a constructive setting.

We consider *MSO formulas* according to the following grammar:

$$\phi, \psi := X \subseteq Y \mid X < Y \mid \perp \mid \phi \to \psi \mid \exists X.\phi \qquad (X, Y : \mathbb{N})$$

We take variables to be natural numbers rather than objects of some abstract type of variables. Informally, a valuation is a function assigning finite sets of natural numbers to every variable. A valuation $\mathscr{V}$ satisfies $X \subseteq Y$ if $\mathscr{V}X \subseteq \mathscr{V}Y$, and $X < Y$ if every element of $\mathscr{V}X$ is smaller than every element of $\mathscr{V}Y$. We write $\mathscr{V} \models \phi$ if $\mathscr{V}$ satisfies ϕ.

We will show that $\mathscr{V} \models \phi$ is decidable. This establishes that the logic behaves classically (i.e., $\mathscr{V} \models \neg\neg\phi \to \phi$ with $\neg\psi := \psi \to \perp$ as usual). In particular, it allows us to use the classical definition of universal quantification (i.e., $\forall X.\phi := \neg\exists X.\neg\phi$) and similarly for the remaining logical operations.

The use of $<$ for second-order variables allows us to employ a syntax without first-order variables. First-order variables can be emulated using second-order variables that are required to be singletons. That is, we define predicates

$$\begin{aligned}\mathsf{empty}\,X &:= \forall Y.X \subseteq Y\\ \mathsf{singleton}\,X &:= \neg\mathsf{empty}\,X \wedge \forall Y.\neg\mathsf{empty}\,Y \to Y \subseteq X \to X \subseteq Y\end{aligned}$$

and emulate the first-order quantifier $\exists n.\phi$ using $\exists n.\mathsf{singleton}\,n \wedge \phi$. If n and m are singletons, $n < m$ corresponds to the usual less-than relation on the respective unique members. We remark that our MSO formulas are technically first-order formulas.

We begin by giving a formal definition of the satisfaction relation. Recall that variables are natural numbers. The formalization employs formulas in De Bruijn representation [1]. That is, the named binder $\exists X.\phi$ is replaced by the nameless binder $\exists\phi$ that always binds the variable 0 and lowers all free variables by one. The named formula $\exists Y.X \subseteq Y$ is then represented as $\exists(X+1 \subseteq 0)$ as usual.

The *satisfaction relation* between *valuations* $\mathscr{V} : \mathbb{N} \to \mathbb{N}^*$ (finite sets being represented as lists) and formulas ϕ, written $\mathscr{V} \models \phi$, is defined by recursion on formulas:

$$\begin{aligned}\mathscr{V} \models X \subseteq Y &:= \forall n \in \mathscr{V}X.n \in \mathscr{V}Y\\ \mathscr{V} \models X < Y &:= \forall n \in \mathscr{V}X\ \forall m \in \mathscr{V}Y.n < m\\ \mathscr{V} \models \bot &:= \bot\\ \mathscr{V} \models \phi \to \psi &:= \mathscr{V} \models \phi \to \mathscr{V} \models \psi\\ \mathscr{V} \models \exists\phi &:= \exists N : \mathbb{N}^*.N :: \mathscr{V} \models \phi\end{aligned}$$

Here, $N :: \mathscr{V} := \lambda n.\mathsf{if}\ n = 0\ \mathsf{then}\ N\ \mathsf{else}\ \mathscr{V}(n-1)$ is the 'stream cons' operation on valuations and handles the shifting of free variables underneath of quantifiers. While valuations assign lists of natural numbers to the free variables, we will treat these these lists like finite sets and use standard set notations.

The first step in deciding $\mathscr{V} \models \phi$ is to capture the fact that only the values that $\mathscr{V}$ assigns to the free variables of ϕ are relevant. We write $\mathsf{bound}\,\phi$ for the least number n such that all free variables of ϕ are smaller than n.

Lemma 12.1 *If $\mathscr{V}$ and $\mathscr{V}'$ agree on all variables up to* $\mathsf{bound}\,\phi$*, then $\mathscr{V} \models \phi$ iff $\mathscr{V}' \models \phi$.*

For every $n : \mathbb{N}$, the class of valuations that assign the empty lists to all variables $X \geq n$ can be represented using words over bit-vectors of length n.

Definition 12.2 We write $\mathbb{B}_n$ for the alphabet of bit-vectors of length n (i.e., n-tuples over $\mathbb{B}$) and $\mathbb{B}_n^*$ for the type of words over $\mathbb{B}_n$. When talking about bit-vectors, we write 1 for true and 0 for false. We write 0^n for the bit-vector with n zeroes and $1v$ for extending the bit-vector v with a 1 on the left. If $v : \mathbb{B}_n^*$, we write $v[k]$ for the k-th symbol of v (0^n if $|v| \leq k$). For $u : \mathbb{B}_n$, we write $u[k]$ for the k-th element of u (or 0 if $n \leq k$).

Definition 12.3 Let $v : \mathbb{B}_n^*$. The valuation for v, written $\mathscr{V}_v$, is defined as

$$\mathscr{V}_v X := \{\, n \mid n < |v| \ \wedge\ v[n][X] = 1 \,\}$$

Example 12.4 Words over $\mathbb{B}_n$ can be thought of as matrices where the characters are columns. Then the X-th row of the matrix for a word v encodes $\mathscr{V}_v(X)$. Consider the 5-letter word v over $\mathbb{B}_3$ depicted by the following matrix.

	0	1	2	3	4
0	1	0	1	0	0
1	0	1	0	0	0
2	0	1	0	1	0

Then $\mathscr{V}_v(0) = \{0,2\}$, $\mathscr{V}_v(1) = \{1\}$, $\mathscr{V}_v(2) = \{1,3\}$, and $\mathscr{V}_v(X) = \emptyset$ for $X \geq 3$. □

Lemma 12.5 *For each $\mathscr{V}$ and n, there exists some $v : \mathbb{B}_n^*$ such that $\mathscr{V}$ agrees with $\mathscr{V}_v$ on all variables smaller than n.*

By assigning valuations to words over $\mathbb{B}_n$, we can assign to every formula the language of those words whose valuations satisfy the formula.

Definition 12.6 (Vector Language) Let ϕ be a formula and let $n : \mathbb{N}$. The language of n-vectors of ϕ is defined as

$$\mathscr{L}_n(\phi) := \{\, v \in \mathbb{B}_n^* \mid \mathscr{V}_v \models \phi \,\}$$

It order to decide the satisfaction relation and satisfiability of formulas, it suffices to decide language membership and language emptiness for vector languages. For this purpose, we construct an NFA $A_{n,\phi}$ accepting $\mathscr{L}_n(\phi)$ for any given ϕ and n. The construction proceeds by recursion on the formula ϕ. The main reason for including the index n is the case for formulas of the form $\exists\phi$. Removing the existential quantifier yields a formula with an additional free variable. Hence, the problem of constructing an NFA accepting $\mathscr{L}_n(\exists\phi)$ is reduced to the problem of constructing an NFA accepting $\mathscr{L}_{n+1}(\phi)$.

It is straightforward, albeit tedious, to construct automata for $\subseteq$ and $<$.

Lemma 12.7 *For each n, X, Y, one can construct automata $A_{n,X\subseteq Y}$ and $A_{n,X<Y}$ such that $\mathscr{L}(A_{n,X\subseteq Y}) \equiv \mathscr{L}_n(X \subseteq Y)$ and $\mathscr{L}(A_{n,X<Y}) \equiv \mathscr{L}_n(X < Y)$.*

For additional detail on these constructions we refer to the Coq development. In the following, we describe the construction for $A_{n,\exists\phi}$, which is the most technical of the required constructions.

Intuitively, $A_{n,\exists\phi}$ is obtained from $A_{n+1,\phi}$ by guessing the content of the first "row" of the input word (cf. Example 12.4). The construction is complicated by the fact that one has to allow for this guessed row to be longer than the input word. This is necessary since the numbers occurring in the witness of the the quantified variable may need to be larger than the length of the input word.

Definition 12.8 Let $A_{n+1,\phi} = (Q,S,F,\rightarrow)$ be an NFA with alphabet $\mathbb{B}_{n+1}$. We define an NFA $A_{n,\exists\phi} := (Q,S,F',\rightsquigarrow)$ with alphabet $\mathbb{B}_n$ where

$$p \overset{v}{\rightsquigarrow} q := p\,(\overset{1v}{\rightarrow} \cup \overset{0v}{\rightarrow})\,q$$
$$F' := \left\{ p \;\middle|\; \exists q \in F.\, p \left(\overset{10^n}{\rightarrow} \cup \overset{00^n}{\rightarrow}\right)^* q \right\}$$

In order to express the correctness properties of the construction above, we define an *extension operation* and a *projection operation* on words.

Definition 12.9 For $b : \mathbb{B}^*$ and $v : \mathbb{B}_n^*$, we write $\binom{b}{v}$ for the word over $\mathbb{B}_{n+1}$ where the first components are given by b and the remaining components are given by v (filling with zeros or 0^n in case $|b| \neq |v|$). For $v : \mathbb{B}_{n+1}^*$, we write $\mathsf{pr}\, v : \mathbb{B}_n^*$ for the result of removing the first component of every tuple.

Lemma 12.10 *1.* $|\binom{b}{v}| = \mathsf{max}(|b|,|v|)$
2. $\mathsf{pr}\binom{b}{v} = v \cdot (0^n)^k$ *for some* $k : \mathbb{N}$.
3. If $v \in \mathscr{L}_{n+1}(\phi)$*, then* $\mathsf{pr}\, v \in \mathscr{L}_n(\exists\phi)$.
4. If $v \in \mathscr{L}_n(\exists\phi)$*, then* $\binom{b}{v} \in \mathscr{L}_{n+1}(\phi)$ *for some* $b : \mathbb{B}^*$.

Proof Claims (1) and (2) are easy to prove. Claims (3) and (4) follow with Lemma 12.1. □

Lemma 12.11 *Let* $\binom{b}{v} \in \mathscr{L}(A_{n+1,\phi})$*. Then* $v \in \mathscr{L}(A_{n,\exists\phi})$.

Proof It suffices to show that every state q of $A_{n+1,\phi}$ that accepts $\binom{b}{v}$ accepts v when seen as a state of $A_{n,\exists\phi}$. We proceed by induction on b. The base case follows by induction on v. For the step case either v is nonempty and we can justify a transition in $A_{n,\exists\phi}$ or v is empty and $A_{n+1,\phi}$ makes a transition from one final state of $A_{n,\exists\phi}$ to another. □

Lemma 12.12 *Let* $v \in \mathscr{L}(A_{n,\exists\phi})$*. Then* $\binom{b}{v} \in \mathscr{L}(A_{n+1,\phi})$ *for some* $b : \mathbb{B}^*$

Proof Similar to the proof of the previous lemma. □

Lemma 12.13 *Let* $v : \mathbb{B}_n^*$*. Then* $v \cdot (0^n)^k \in \mathscr{L}_n(\phi)$ *iff* $v \in \mathscr{L}_n(\phi)$.

Lemma 12.14 *Let* $A_{n+1,\phi}$ *be an NFA accepting* $\mathscr{L}_{n+1}(\phi)$*. Then* $A_{n,\exists\phi}$ *accepts* $\mathscr{L}_n(\exists\phi)$.

Proof $\mathscr{L}(A_{n,\exists\phi}) \subseteq \mathscr{L}_n(\exists\phi)$: Let $v \in \mathscr{L}(A_{n,\exists\phi})$. By Lemma 12.12, there exists some $b : \mathbb{B}^*$ such that $\binom{b}{v} \in \mathscr{L}(A_{n+1,\phi})$ and therefore $\binom{b}{v} \in \mathscr{L}_{n+1}(\phi)$ by assumption. Hence, $\mathsf{pr}\binom{b}{v} \in \mathscr{L}_n(\exists\phi)$ by Lemma 12.10(3). By Lemma 12.10(2) $\mathsf{pr}\binom{b}{v} = v \cdot (0^n)^k$ for some k. Thus, $v \in \mathscr{L}_n(\exists\phi)$ by Lemma 12.13.
$\mathscr{L}_n(\exists\phi) \subseteq \mathscr{L}(A_{n,\exists\phi})$: Let $v \in \mathscr{L}_n(\exists\phi)$. Then there exists some $b : \mathbb{B}^*$ such that $\binom{b}{v} \in \mathscr{L}_{n+1}(\phi)$ (Lemma 12.10(4)) and therefore $\binom{b}{v} \in \mathscr{L}(A_{n+1,\phi})$. Hence $v \in \mathscr{L}(A_{n,\exists\phi})$ by Lemma 12.11. □

Theorem 12.15 *Given* ϕ *and n, one can construct an NFA* $A_{n,\phi}$ *accepting* $\mathscr{L}_n(\phi)$.

Proof By induction on ϕ. The base cases follow with Lemma 12.7 The case where $\phi = \exists\psi$ follows with Lemma 12.14. The remaining cases are straightforward.

Corollary 12.16 $\mathscr{V} \models \phi$ *is decidable.*

Proof By Lemma 12.5, we obtain some $v : (\mathbb{B}_{\mathsf{bound}\,\phi})^*$, such that $\mathscr{V}_v$ agrees with $\mathscr{V}$ up to $\mathsf{bound}\,\phi$. By Lemma 12.1, it suffices to decide $\mathscr{V}_v \models \phi$ which amounts to checking whether $v \in \mathscr{L}(A_{\mathsf{bound}\,\phi,\phi})$. □

Corollary 12.17 $\mathscr{V} \models \neg\neg\phi \to \phi$.

Proof Immediate with Corollary 12.16.

Corollary 12.18 *Satisfiablity of MSO formulas is decidable.*

Proof Similar to the proof of Corollary 12.16, the satisfiability problem reduces to the emptiness problem for $\mathscr{L}(A_{\mathsf{bound}\,\phi,\phi})$. □

Remark 12.19 The construction outlined above yields a decision procedure of nonelementary complexity (Negation requires complementation, and thus determinization, while existential quantification introduces nondeterminism). This is essentially optimal [32].

The formalization of the results described in this section employs dependent types in a particularly pleasing manner. The type $\mathbb{B}_n$ is represented using n-tuples (i.e., length-indexed lists) as provided by Ssreflect. The only type-changing operations employed in the proof are extension and projection (cf. Definition 12.9). The types

$$\begin{aligned} \binom{\cdot}{_} &: \forall n.\mathbb{B}^* \to \mathbb{B}_n^* \to \mathbb{B}_{n+1}^* \\ \mathsf{pr} &: \forall n.\mathbb{B}_{n+1}^* \to \mathbb{B}_n^* \end{aligned}$$

can be inferred automatically from the underlying list operations. This simplicity is, at least in part, due to the De Bruijn representation. Since quantifiers in De Bruijn representation always quantify over the variable 0, we never have to insert rows in the middle or remove any row but the first.

13 WS1S as a Representation for Regular Languages

We now prove the characterization theorem of Büchi [6], Elgot [13], and Trakhtenbrot [37] which establishes WS1S formulas as representations of regular languages.

By assigning valuations to words over Σ, one can associate to the formula ϕ the language $\mathscr{L}_\Sigma(\phi)$ of those words whose valuations satisfy the formula. Let n be the size of the alphabet Σ and assume $\mathsf{bound}\,\phi \leq n$. The valuation corresponding to a word $x : \Sigma^*$ maps the variable X to the list of occurrences of the X-th symbol of Σ in the word x (or to the empty list if $X \geq n$). Note that the valuation for x interprets the free variables of ϕ as a partition since there is exactly one symbol at every position. We formalize $\mathscr{L}_\Sigma(\phi)$ by bijectively mapping the symbols of Σ to the unit vectors of length n, i.e., the elements of $\mathbb{B}_n$ having exactly one occurrence of 1.

Definition 13.1 **(Σ-language)** Let $n := |\Sigma|$ and for $a : \Sigma$ let $u_a : \mathbb{B}_n$ be the unit vector with a single 1 at position $\mathsf{rank}\,a$ and 0 at all other positions. We write $h_\Sigma : \Sigma^* \to \mathbb{B}_n^*$ for the homomorphism mapping each symbol $a : \Sigma$ to u_a. The Σ-language of a formula ϕ is then defined as

$$\mathscr{L}_\Sigma(\phi) := \{\, x \in \Sigma^* \mid \mathscr{V}_{h_\Sigma(x)} \models \phi \,\}$$

Lemma 13.2 *$\mathscr{L}_\Sigma(\phi)$ is regular.*

Proof Since $\mathscr{L}_\Sigma(\phi) \equiv h^{-1}(\mathscr{L}_n(\phi))$, regularity follows with Theorems 7.14 and 12.15.

We now show that formulas can describe all regular languages by giving a language preserving translation from NFAs to formulas. To simplify the presentation, we describe the construction using named formulas. If $a : \Sigma$, we write P_a for the free variable containing the positions of a in the input word. Further, we use lower-case letters for first-order variables represented using $\mathsf{singleton}$, e.g., $\exists x.\, x \in Y$ is just notation for $\exists X.\, \mathsf{singleton}(X) \wedge X \subseteq Y$. The formalization in Coq is carried out using the De Bruijn representation used in the previous sections.

We fix some NFA $A = (Q, S, F, \to)$ (with alphabet Σ) for the rest of this section. Further we set $n := |Q|$ and refer to the individual states of A as $q_0, \ldots, q_{n-1}$. We now construct a formula ϕ_A such that $\mathscr{L}_\Sigma(\phi_A) \equiv \mathscr{L}(A)$.

The idea is to construct a formula of the form

$$\phi_A := \exists m.(\phi_{\mathsf{max}} \wedge \exists X_{n-1} \ldots X_0.\, \phi_{\mathsf{run}})$$

$$\phi_{\mathsf{part}} := \forall k.k \leq m. \bigvee_{q_i \in Q} \left(k \in X_i \wedge \bigwedge_{q_j \in Q \wedge q_i \neq q_j} k \notin X_j \right)$$

$$\phi_{\mathsf{init}} := \forall x.\mathsf{zero}(x) \rightarrow \bigvee_{q_i \in S} x \in X_i$$

$$\phi_{\mathsf{fin}} := \bigvee_{q_j \in F} m \in X_j$$

$$\phi_{\mathsf{trans}} := \forall k k'.\mathsf{succ}(k,k') \rightarrow k < m \rightarrow \bigvee_{q_i \xrightarrow{a} q_j} k \in X_i \wedge k \in P_a \wedge k' \in X_j$$

$$\phi_{\mathsf{run}} := \phi_{\mathsf{part}} \wedge \phi_{\mathsf{init}} \wedge \phi_{\mathsf{fin}} \wedge \phi_{\mathsf{trans}}$$

Fig. 1 Encoding of the runs of A

where ϕ_{max} ensures that the witness for m is the length of the input word and ϕ_{run} ensures that the witnesses for $X_{n-1} \ldots X_0$ encode an accepting run of A. More precisely, $k \in X_i$ is to mean that after reading k symbols of the input word, the automaton is in state q_i.

We start by defining

$$\phi_{\mathsf{max}} := \forall k. k < m \leftrightarrow \bigvee_{a \in \Sigma} k \in P_a$$

For ϕ_{run}, we need to ensure that the X_i form a partition of the set $\{0, \ldots, m\}$ in such way that $0 \in X_i$ for some stating state q_i, $m \in X_j$ for some final state q_j, and if $k \in X_i$ and $k+1 \in X_j$ then $q_i \xrightarrow{w[k]} q_j$. This leads to the definition of ϕ_{run} given in Figure 1, where $\mathsf{zero}(x)$ ensures that x is (the singleton containing) 0 and $\mathsf{succ}(x,y)$ ensures that (the unique element of) y is the successor of (the unique element of) x. The formula employed here is a simplified version of the formula employed in [19] that more closely matches the notion of run defined in Section 6.

Lemma 13.3 *Let A be an NFA over Σ. Then $\mathscr{L}_\Sigma(\phi_A) \equiv \mathscr{L}(A)$.*

Proof Let $x \in \Sigma^*$. We show $\mathscr{V}_{h_\Sigma(x)} \models \phi_A$ iff $x \in \mathscr{L}(A)$. If $x \in \mathscr{L}(A)$ then there exists an accepting run $(r_i)_{i \leq |x|}$ of A on x. We set the existential witness for m to $|x|$ and for each X_i to $\{ k \mid \exists j. r_j = q_i \}$. It is straightforward to verify that $\mathscr{V}_{h_\Sigma(x)} \models \phi_A$ with these instantiations.

Conversely, if $\mathscr{V}_{h_\Sigma(x)} \models \phi_A$, we need to construct a run from the witnesses used for m and the X_i. Since ϕ_{part} is satisfied, the X_j form a partition of $\{0, \ldots, |x|\}$. We set r_i to be the unique state q_j such that $i \in X_j$. Again, it is straightforward to verify that this constitutes an accepting run of A on x. □

Theorem 13.4 *Let $L \subseteq \Sigma^*$. Then L is regular iff $L \equiv \mathscr{L}_\Sigma(\phi)$ for some ϕ.*

Proof Immediate with Lemma 13.2 and Lemma 13.3. □

Remark 13.5 The proof of Lemma 13.3 relies on the fact that $h(\mathscr{L}_\Sigma(\phi))$ is comprised of unit vectors only. In particular, there is no translation t from NFAs with alphabet $\mathbb{B}_n$ to formulas satisfying $\mathscr{L}_n(tA) \equiv \mathscr{L}(A)$ for all A since $\mathscr{L}_n(\phi)$ is always closed under adding or removing trailing 0^n vectors (Lemma 12.13).

While the simple structure of formulas in De Bruijn representation is nice when analyzing formulas recursively (as in the translation from formulas to NFAs), one has to be careful

when manually writing down larger formulas. We decompose the formula describing NFA runs into parts of manageable size as shown in Figure 1 that each come with their own correctness lemmas that fully characterize the formula. The most complex monolithic formula is ϕ_{trans} whose formalization spans 6 lines. The correctness proofs for various parts employ contexts of the form $X_0 :: \ldots :: X_{n-1} :: m :: I$. That is, for the state q_j the term $\mathsf{rank}\, q_j$ (shifted to respect local quantifiers) refers to X_j. Consequently, the X_j can be accessed using readable names even in De Bruijn representation. For context lookups, the shifts required for local quantifiers can usually be eliminated though simplification (i.e., Coq's internal notion of computation). Consequently, wrong indices usually did not simplify as expected and where therefore easy to spot and correct.

14 Related Work

Given the theoretical and practical importance of regular language representations and their associated algorithms, there are a number of developments formalizing various aspects of the theory. This includes verified practical decision procedures as well as more mathematically minded developments like the one described in this paper.

Constable et al. [8] constructively formalize automata theory, including the Myhill-Nerode theorem, in Nuprl. They use a notion of finite type to represent state sets. However, Nuprl allows the construction of (finite) quotient types with respect to undecidable relations. This allows proving the Myhill-Nerode theorem and the existence of a minimization function for DFAs assuming only decidability of the language. However, the minimization function obtained this way is computationally trivial, i.e., does not actually perform minimization [8, Sect. 7.2].

On the Coq side, one of the earliest works is a verified translation from regular expressions to finite automata by Filliâtre [14]. Here, states are represented using finite sets of numbers leading to a representation that is considerably more difficult to work with than finite types. Further, there are a number of verified and executable decision procedures. Coquand and Siles [9] build a reflective decision procedure for regular expression equivalence based on Brzozowski's algebraic method. The main technical difficulty in their work is in formalizing the termination argument, i.e., the fact that up to a certain decidable equivalence, regular expressions have only finitely many derivatives. In a substantial development, Braibant and Pous [4] verify a more efficient decision procedure for regular expression equivalence based on finite automata and prove it sound and complete for all Kleene algebras (KA). For efficiency reasons, they represent state sets as bounded natural numbers. In [30], Pous describes a complete redesign of [4] based on partial derivatives. As corollary of the completeness proof for an axiomatization of KA, he obtains a proof of Kleene's theorem. The completeness proof uses generalized automata represented as matrices over regular expressions. Moreira et.al. [26] develop a similar decision procedure for KA also based on partial derivatives.

On the HOL-side (i.e., Isabelle/HOL) the situation is more complex. Since quantification over types is not available in these systems, using types to represent state spaces is not feasible. One alternative representation is to use numbered states or bit lists [27]. However, this requires the explicit renaming of states when combining automata and weakens the type checking for automata. Due to this difficulty, much of the work on regular languages in Isabelle/HOL focuses on regular expressions rather than finite automata. This includes verified (partial correctness) decision procedures for regular expression and relation algebra equivalences [25], a decision procedure for WS1S and M2L(Str) based on regular expres-

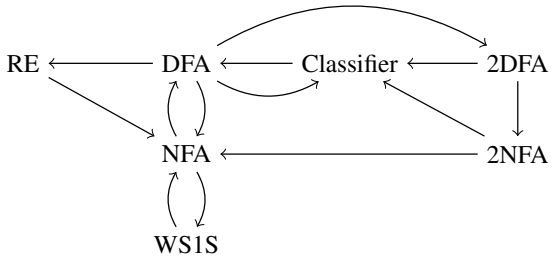

Fig. 2 Summary of translations between representations. The translation from 2NFAs to NFAs yields an automaton for the complement language.

sions with projection [38], and a proof of the Myhill-Nerode theorem only using regular expressions [42]. The last paper also includes a detailed discussion of the problem of defining a type of finite automata in Isabelle/HOL. In a substantial development, Berghofer [2] develops a verified decision procedure for Presburger arithmetic based on finite automata over bit-strings.

More recently, and partly in response to [41] and [11], Paulson [28] formalized automata theory, including the Myhill-Nerode theorem and Brzozowski's minimization algorithm, in Isabelle/HOL based on hereditarily finite sets. Like finite types, HF sets have all the closure properties required for the usual constructions on finite automata. However, there are a number of differences. Due to the absence of dependent types, the definition of DFAs in [28] is split into a type that overapproximates DFAs and a predicate that checks well-formedness conditions (e.g., that the starting state is a state of the automaton). Finite types, being types, benefit from type checking and thus avoid the need for explicit well-formedness conditions. Moreover, many finite types are inductively defined data types (e.g., product and sum types) and thus allow pattern matching leading to more natural definitions.

15 Conclusion

We have shown that many results about regular languages can be obtained constructively and formally with reasonable effort. We have shown the equivalence of seven representations of regular languages (DFAs, NFAs, regular expressions, classifiers, 2DFAs, 2NFAs, and MSO formulas) and a number of closure properties of these representations. Equivalence of the various representations is obtained trough language-preserving translations as summarized in Figure 2. As part of the equivalence proof between formulas and NFAs, we establish decidability of satisfaction and satisfiability for WS1S. The part of this work where constructive logic forced us to deviate most from the presentation in the literature is the Myhill Nerode theorem. This led to the new notion of a classifier and a constructive reformulation of the theorem providing for the construction of DFAs from classifiers (cf. Remark 5.10). This construction is used to obtain redundant translations from 2FAs to DFAs providing better size bounds.

Obtaining an elegant formalization in Coq first requires rethinking the original proofs, which are mostly phrased in terms of classical set theory, in terms of the constructive type theory underlying Coq. Finite automata are a typical example of a finite dependently-typed mathematical structure. Consequently, finite types and dependent types are used pervasively throughout the whole development. Our representation of finite automata relies on dependent record types and finite types being first-class objects. Dependent types are also used to

represent the possible configurations of a two-way automaton as a word-indexed collection of finite types and to obtain the vector languages for formulas.

In addition to the mathematical redesign, we implement all concepts in a manner that is both faithful to the mathematical development and easy to work with. The Coq development accompanying this paper [10] matches the paper proofs closely and provides the details elided in the paper. Altogether the Coq development consists of about 3000 lines of code (1300 lines of specification / 1700 lines of proof). The two biggest parts of the development are the results about WS1S (870 lines) and the results about two-way automata (550 lines). The conciseness of the Coq development is, at least in part, due to our use of the Mathematical Components Libraries (Math-Comp) [36], which provide the necessary infrastructure for our development. With the exception of modular arithmetic and binomial coefficients, our Coq development uses just about every concept provided by the Ssreflect component of Math-Comp (boolean reflection, finite and countable types, finite sets, graph reachability, length-indexed lists, etc.).

The present work shows that current interactive theorem proving tools (Coq and Ssreflect in our case) allow for a mathematically rewarding and reusable formalization of non-trivial results about regular languages with reasonable effort.

As mentioned above, the two most technical parts of our development are the translations from two-way automata to one-way automata and the the translations between NFAs and MSO formulas. We comment on each of these in turn.

The original translations from two-way automata to one-way automata [34, 39] are described at a fairly informal level. When spelling out the details, the constructions and proofs become delicate and call for formalization, in particular if one does not have the robust intuitions of experts in automata theory. We adapt the proof of Shepherdson [34] from deterministic two-way automata without end-markers to nondeterministic two-way automata with end-markers. This does not require conceptual changes. We had formalized the construction for 2DFAs (with end-markers) before realizing that the construction applies, with minimal changes, also to 2NFAs. That this is the case appears to be known to the experts [29]. However, to the best of our knowledge, the construction was never published. We not only adapt Shepherdson's construction to 2NFAs, we also show that when applied to 2DFAs, the construction yields the bounds established in [34]. One of the advantages of having formal proofs is that one can modify definitions and lemma statements and have the theorem prover point out the places in the proof that needed to be adapted. The process of adapting the construction from 2DFAs to 2NFAs profited greatly from this.

Similar to the translations from two-way automata to one-way automata, the translations between MSO formulas and NFAs are fairly technical. For the direction from formulas to NFAs, the main tool for factoring the proof into small lemmas is the introduction of extension and projection operations on words over bit-vectors. Here, we profit from the De Bruijn representation for formulas, which ensures that we never have to insert a row into the middle or remove a row other than the first. The translation from formulas to automata also establishes decidability of the satisfaction relation and satisfiability. The former establishes that the logic behaves classically, thus allowing us to show that the logical operations defined in terms of our minimal syntax behave correctly. These operations are then used in the translation from NFAs to formulas. As a consequence, and unlike in a classical setting, the correctness proofs for the two translations are not entirely independent.

Acknowledgements We thank Jan-Oliver Kaiser, who was involved in our previous work on one-way automata and also in some of the early experiments with two-way automata. We also thank Damien Pous and the anonymous reviewers for helpful comments.

This is a post-peer-review, pre-copyedit version of an article published in the Journal of Automated Reasoning. The final authenticated version is available online at: https://doi.org/10.1007/s10817-018-9460-x

References

1. Abadi, M., Cardelli, L., Curien, P., Lévy, J.: Explicit substitutions. J. Funct. Program. 1(4), 375–416 (1991)
2. Berghofer, S., Reiter, M.: Formalizing the logic-automaton connection. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Theorem Proving in Higher Order Logics (TPHOLs 2009). LNCS, vol. 5674, pp. 147–163. Springer (2009)
3. Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.): Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings, LNCS, vol. 7998. Springer (2013)
4. Braibant, T., Pous, D.: Deciding Kleene algebras in Coq. Log. Meth. Comp. Sci. 8(1:16), 1–42 (2012)
5. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM 11(4), 481–494 (1964)
6. Büchi, J.R.: Weak second-order arithmetic and finite automata. Zeitschr. f. math. Logic und Grundladen d. Math. 6, 66–92 (1960)
7. Cohen, C.: Pragmatic quotient types in Coq. In: Blazy et al. [3], pp. 213–228
8. Constable, R.L., Jackson, P.B., Naumov, P., Uribe, J.C.: Constructively formalizing automata theory. In: Plotkin, G.D., Stirling, C., Tofte, M. (eds.) Proof, Language, and Interaction. pp. 213–238. The MIT Press (2000)
9. Coquand, T., Siles, V.: A decision procedure for regular expression equivalence in type theory. In: Jouannaud, J.P., Shao, Z. (eds.) Certified Programs and Proofs (CPP 2011). LNCS, vol. 7086, pp. 119–134. Springer (2011)
10. Doczkal, C., Kaiser, J.O., Smolka, G.: Coq development accompanying this paper, https://github.com/chdoc/coq-reglang
11. Doczkal, C., Kaiser, J., Smolka, G.: A constructive theory of regular languages in Coq. In: Gonthier, G., Norrish, M. (eds.) Certified Programs and Proofs (CPP 2013). LNCS, vol. 8307, pp. 82–97. Springer (2013)
12. Doczkal, C., Smolka, G.: Two-way automata in Coq. In: Blanchette, J.C., Merz, S. (eds.) Interactive Theorem Proving (ITP 2016). LNCS, vol. 9807, pp. 151–166. Springer (2016)
13. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Transactions of the American Mathematical Society 98 (1961)
14. Filliâtre, J.C.: Finite automata theory in Coq: A constructive proof of kleene's theorem. Tech. Rep. 97-04, LIP - ENS Lyon (1997)
15. Gelade, W., Neven, F.: Succinctness of the complement and intersection of regular expressions. ACM Trans. Comput. Logic 13(1), 4:1–4:19 (2012)
16. Gonthier, G., Mahboubi, A., Rideau, L., Tassi, E., Théry, L.: A modular formalisation of finite group theory. In: Schneider, K., Brandt, J. (eds.) Theorem Proving in Higher Order Logics (TPHOLs 2007). LNCS, vol. 4732, pp. 86–101. Springer (2007)
17. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001], LNCS, vol. 2500. Springer (2002)
18. Hedberg, M.: A coherence theorem for Martin-Löf's type theory. J. Funct. Program. 8(4), 413–436 (1998)
19. Hofmann, M., Lange, M.: Automatentheorie und Logik. eXamen.press, Springer (2011)
20. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation - international edition (2. ed). Addison-Wesley (2001)
21. Johnsonbaugh, R., Miller, D.P.: Converses of pumping lemmas. In: Austing, R.H., Cassel, L.N., Miller, J.E., Joyce, D.T. (eds.) Proceedings of the 21st SIGCSE Technical Symposium on Computer Science Education, 1990, Washington, DC, USA, 1990. pp. 27–30. ACM (1990)
22. Khoussainov, B., Nerode, A.: Automata Theory and its Applications. Springer (2012)
23. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Shannon, C.E., McCarthy, J. (eds.) Automata Studies, pp. 3–42. Princeton University Press (1956)
24. Kozen, D.: Automata and computability. Undergraduate texts in computer science, Springer (1997)
25. Krauss, A., Nipkow, T.: Proof pearl: Regular expression equivalence and relation algebra. J. Autom. Reasoning 49(1), 95–106 (2012)
26. Moreira, N., Pereira, D., de Sousa, S.M.: Deciding kleene algebra terms equivalence in Coq. J. Log. Algebr. Meth. Program. 84(3), 377–401 (2015)

27. Nipkow, T.: Verified lexical analysis. In: Grundy, J., Newey, M.C. (eds.) Theorem Proving in Higher Order Logics (TPHOLs '98). LNCS, vol. 1479, pp. 1–15. Springer (1998)
28. Paulson, L.C.: A formalisation of finite automata using hereditarily finite sets. In: Felty, A.P., Middeldorp, A. (eds.) Automated Deduction (CADE-25). LNCS, vol. 9195, pp. 231–245. Springer (2015)
29. Pighizzini, G.: Two-way finite automata: Old and recent results. Fundam. Inform. 126(2-3), 225–246 (2013)
30. Pous, D.: Kleene algebra with tests and coq tools for while programs. In: Blazy et al. [3], pp. 180–196
31. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. 3(2), 114–125 (1959)
32. Reinhardt, K.: The complexity of translating logic to finite automata. In: Grädel et al. [17], pp. 231–238
33. Rosenberg, A.L.: State. In: Goldreich, O., Rosenberg, A.L., Selman, A.L. (eds.) Theoretical Computer Science, Essays in Memory of Shimon Even. LNCS, vol. 3895, pp. 375–398. Springer (2006)
34. Shepherdson, J.: The reduction of two-way automata to one-way automata. IBM J. Res. Develp. 3 (1959)
35. The Coq Proof Assistant: `http://coq.inria.fr`
36. The Mathematical Components Project: `http://math-comp.github.io/math-comp/`
37. Trakhtenbrot, B.A.: Finite automata and the logic of monadic predicates. Dokl. Akad. Nauk SSSR 140, 326–329 (1961)
38. Traytel, D., Nipkow, T.: Verified decision procedures for MSO on words based on derivatives of regular expressions. J. Funct. Program. 25, 1–30 (2015)
39. Vardi, M.Y.: A note on the reduction of two-way automata to one-way automata. Inf. Process. Lett. 30(5), 261–264 (1989)
40. Vardi, M.Y.: Endmarkers can make a difference. Inf. Process. Lett. 35(3), 145–148 (1990)
41. Wu, C., Zhang, X., Urban, C.: A formalisation of the Myhill-Nerode theorem based on regular expressions (proof pearl). In: van Eekelen, M.C.J.D., Geuvers, H., Schmaltz, J., Wiedijk, F. (eds.) Interactive Theorem Proving (ITP 2011). LNCS, vol. 6898, pp. 341–356. Springer (2011)
42. Wu, C., Zhang, X., Urban, C.: A formalisation of the Myhill-Nerode theorem based on regular expressions. J. Autom. Reasoning 52(4), 451–480 (2014)