Two-Way Automata in Coq

Christian Doczkal Gert Smolka

Published in Proc. of Interactive Theorem Proving, LNCS vol. 9807, Springer, 2016
DOI: http://dx.doi.org/10.1007/978-3-319-43144-4_10

We formally verify translations from two-way automata to one-way automata based on results from the literature. Following Vardi, we obtain a simple reduction from nondeterministic two-way automata to one-way automata that leads to a doubly-exponential increase in the number of states. By adapting the work of Shepherdson and Vardi, we obtain a singly-exponential translation from nondeterministic two-way automata to DFAs. The translation employs a constructive variant of the Myhill-Nerode theorem. Shepherdson's original bound for the translation from deterministic two-way automata to DFAs is obtained as a corollary. The development is formalized in Coq/Ssreflect without axioms and makes extensive use of countable and finite types.

1 Introduction

Two-way finite automata are a representation for regular languages introduced by Rabin and Scott [15]. Unlike one-way automata, two-way automata may move back and forth on the input word and may be seen as read-only Turing machines without memory.

Both deterministic two-way automata (2DFAs) and nondeterministic two-way automata (2NFAs) exactly accept regular languages [17, 15, 20]. However, some languages have 2DFAs that are exponentially smaller than the minimal DFA; for instance the languages $I_n := (a+b)^* a (a+b)^n$ from [14]. It is known that the cost (in terms of the number of states) of simulating both 2DFAs and 2NFAs with DFAs is exponential [17, 20]. Whether the cost of simulating NFAs and 2NFAs using 2DFAs is also exponential is still an open problem [16, 14].

As is frequently the case with language-theoretic results, the proofs in the literature are described in a fairly informal manner. When carried out in detail, the constructions are delicate and call for formalization. We are the first to provide constructive and machine-checked proofs of the following results:

1. For every n-state 2NFA M there exists an NFA with at most 2^{2n} states accepting the complement of the language of M.
2. For every n-state 2DFA there is an equivalent DFA with at most $(n+1)^{(n+1)}$ states.
3. For every n-state 2NFA there is an equivalent DFA with at most 2^{n^2+n} states.

Our proofs mostly refine the proofs given by Shepherdson [17] and Vardi [20]. Result (1) is easiest to show. It establishes that the languages accepted by 2NFAs (and therefore also 2DFAs) are regular. Our proof is based on a construction in [20]. If one wants to obtain an automaton for the original language, using (1) leads to a doubly exponential increase in the number of states. A singly-exponential bound can be obtained using a construction from Shepherdson [17] originally used to establish (2). Building on ideas from [20], we adapt Shepherdson's construction to 2NFAs. That this is possible appears to be known [14], but to the best of our knowledge the construction for 2NFAs has never been published. Once we have established (3), we obtain (2) by showing that if the input automaton is deterministic, the constructed DFA has at most $(n+1)^{(n+1)}$ states. This allows us to get both results with a single construction.

The reduction to DFAs makes use of the Myhill-Nerode theorem. We employ a constructive variant where Myhill-Nerode relations are represented as functions we call classifiers that are are supplemented with decidability assumptions to provide for a constructive proof. When constructing DFAs from 2NFAs, the decidability requirements are easily satisfied. The application of the constructive Myhill-Nerode theorem to the reduction from 2NFAs to DFAs demonstrates that the construction is useful.

We formalize our results in Coq [18] using the Ssreflect [9] extension. The formalization accompanying this paper[1] extends and revises previous work [6] and contains a number of additional results. The development makes extensive use of finite and countable types [8, 7] as provided by Ssreflect. In particular, we use finite types to represent states for finite automata.

Various aspects of the theory of regular languages have been formalized in different proof assistants. In addition to executable certified decision methods [2, 3, 5, 12, 19] based on automata or regular expressions, there are a number of purely mathematical developments. Constable et al. [4] formalize automata theory in Nuprl, including the Myhill-Nerode theorem. Wu et al. [22] give a proof of the Myhill-Nerode theorem based on regular expressions. Recently, Paulson [13] has formalized the Myhill-Nerode theorem and Brzozowski's minimisation algorithm in Isabelle.

The paper is organized as follows. Sections 2 and 3 recall some type theoretic constructions underlying our proofs and describe how the usual language

[1] www.ps.uni-saarland.de/extras/itp16-2FA

theoretic notions are represented in type theory. In Section 4 we define one-way automata. In Section 5 we prove the constructive variant of the Myhill-Nerode theorem. Section 6 defines two-way automata. Section 7 presents the reduction from 2NFAs to NFAs (for the complement) and Section 8 the reductions from 2NFAs and 2DFAs to DFAs.

2 Type Theory Preliminaries

We formalize our results in the constructive type theory of the proof assistant Coq [18]. In this setting, decidability properties are of great importance. We call a proposition **decidable**, if it is equivalent to a boolean expression. Similarly, we call a predicate decidable, if it is equivalent to a boolean predicate. In the mathematical presentation, we will not distinguish between decidable propositions and the associated boolean expressions.

In type theory, 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 [8].

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 $g : \mathbb{N} \to X_\bot$ such that $g(f\,x) = \mathsf{Some}\,x$ for all $x : X$, where $X_\bot$ is the option type over X. All countable types are also discrete. We will make use of the fact that surjective functions from countable types to discrete types have right inverses.

Lemma 2.1 Let X be countable, Y be discrete, and $f : X \to Y$ be surjective. Then there exists a function $f^{-1} : Y \to X$ such that $f(f^{-1}y) = y$ for all y.

Proof The countable type X is equipped with a choice operator

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

satisfying $p(\mathsf{xchoose}_X\,p\,E)$ for all $E : (\exists x : X.\,p\,x)$. Given some $y : Y$, we construct $f^{-1}y$ using the choice operator with $p := \lambda x : X.\,f\,x = y$. ∎

A **finite type** is a type X together with a list enumerating all elements of X. If X is finite, we write $|X|$ for the number of elements of X. For our purposes, the most important property of finite types is that quantification over finite types preserves decidability.

Discrete, countable, and finite types are closed under forming product types $X \times Y$, sum types $X + Y$, and option types $X_\bot$. 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 $\{ x : X \mid p\, x \}$, whose elements are dependent pairs of elements $x : X$ and proofs of $p\, x = \mathsf{true}$, can be treated as a subtype of X. In particular, the first projection yields an injection from $\{ x : X \mid p\, x \}$ to X since $p\, x = \mathsf{true}$ is proof irrelevant [10].

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 Type Theory

For us, an **alphabet** is a finite type. For simplicity, we fix some alphabet Σ throughout the paper and refer to its elements as **symbols**. The type of lists over Σ, written Σ^*, is a countable type. We refer to terms of this type as **words**.

The letters a, b always denote symbols. The letters x, y, and z always denote words and ε denotes the empty word. We write $|x|$ to denote the **length** of the word x and xy or $x \cdot y$ (if this increases readability) for the concatenation of x and y. We also write $x[n, m]$ for the subword from position n (inclusive) to m (exclusive), e.g., $x = x[0, j] \cdot x[j, |x|]$.

A **language** is a predicate on words, i.e., a function of type $\Sigma^* \to \mathsf{Prop}$ (or $\Sigma^* \to \mathbb{B}$ for decidable languages). This yields an intensional representation. We write $L_1 \equiv L_2$ to denote that L_1 and L_2 are equivalent (i.e, extensionally equal). The absence of extensionality causes no difficulties since all our constructions respect language equivalence. To increase readability, we employ the usual set-theoretic notations for languages. In particular, we write $\overline{L}$ for the complement of the language L.

4 One-Way Automata

Deterministic one-way automata (DFAs) can be seen as the most basic operational characterization of regular languages. In addition to DFAs, we also define nondeterministic finite automata (NFAs) since both will serve as targets for our translations from two-way automata to one-way automata.

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**.

- $F : Q \to \mathbb{B}$ determines the **final states**.
- $\delta : Q \to \Sigma \to Q$ is the **transition function**.

In Coq, we represent DFAs using dependent records:

$$\begin{array}{lll} \mathsf{dfa} := \{ & \mathsf{state} & : \mathsf{finType} \\ & \mathsf{start} & : \mathsf{state} \\ & \mathsf{final} & : \mathsf{state} \to \mathbb{B} \\ & \mathsf{trans} & : \mathsf{state} \to \Sigma \to \mathsf{state}\} \end{array}$$

Here, $\mathsf{state} : \mathsf{finType}$ restricts the type states to be a finite type. Finite types provide for a formalization of finite automata that is very convenient to work with. In particular, finite types have all the closure properties required for the usual constructions on finite automata [6].

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:

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

We say that a state q of A **accepts** a word x if $\hat{\delta}\, q\, x \in F$. The **language of** A, written $\mathcal{L}(A)$, is then defined as the collection of words accepted by the starting state:

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

Note that is a decidable language.

Definition 4.2 We say that a DFA A **accepts** the language L if $L \equiv \mathcal{L}(A)$. We call L **regular** if it is accepted by some DFA.

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

Definition 4.3 A **nondeterministic finite automation (NFA)** is a structure (Q, S, F, δ) 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.
- $\delta : Q \to \Sigma \to Q \to \mathbb{B}$ is the **transition relation**.

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

$$\begin{aligned} \mathsf{accept}\, p\, \varepsilon &:= p \in F \\ \mathsf{accept}\, p\, (a :: x) &:= \exists q \in Q.\, \delta\, p\, a\, q \wedge \mathsf{accept}\, q\, x \end{aligned}$$

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

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

Note that since S is finite, this is also a decidable language. As with DFAs, acceptance of languages is defined up to language equivalence.

NFAs can be converted to DFAs using the well-known powerset construction.

Fact 4.4 For every n-state NFA A, there exists a DFA with at most 2^n states accepting $\mathcal{L}(A)$.

5 Classifiers and Myhill-Nerode

We now introduce classifiers as an abstract characterization of DFAs. For us, classifiers play the role of Myhill-Nerode relations (cf. [11]). Classifiers differ from Myhill-Nerode relations mainly in that they include decidability assumptions required for constructive proofs. Classifiers have a cut-off property which yields a number of useful decidability properties. Further, classifiers provide a sufficient criterion for the existence of DFAs that is useful for the translation from two-way automata to one-way automata.

Definition 5.1 Let Q be a type and let $f : \Sigma^* \to Q$. Then f is called **right congruent** if $fx = fy$ implies $f(xa) = f(ya)$ for all $x, y : \Sigma^*$ and all $a : \Sigma$.

Definition 5.2 A function $f : \Sigma^* \to Q$ is called a **classifier** if it is right congruent and Q is a finite type. If L is a decidable language, a **classifier for L** is a classifier that **refines** L, i.e., that satisfies $\forall x\, y.\, f\, x = f\, y \to (x \in L \leftrightarrow y \in L)$.

Fact 5.3 If $A = (Q, s, F, \delta)$ is a DFA, then $\hat{\delta} s$ is a classifier for $\mathcal{L}(A)$.

If $f : \Sigma^* \to Q$ is a classifier, the congruence property of f allows us to decide whether a certain element of Q is in the image of f.

Theorem 5.4 (Cut-Off) Let $f : \Sigma^* \to Q$ be a classifier and let $P : Q \to \mathsf{Prop}$. Then

$$\exists x.\, P(f\, x) \iff \exists x.\, |x| \le |Q| \wedge P(f\, x)$$

Proof The direction from right to left is trivial. For the other direction let x such that $P(f\, x)$. We proceed by induction on $|x|$. If $|x| \le |Q|$ the claim is trivial. Otherwise, there exist $i < j < |x|$ such that $f(x[0, i]) = f(x[0, j])$. Since f is right congruent, we have $f\, x = f\, (x[0, i] \cdot x[j, |x|])$ and the claim follows by induction hypothesis. ∎

Corollary 5.5 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 5.4, 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 5.6 Language emptiness for DFAs is decidable.

Proof Let $A = (Q, s, F, \delta)$ be a DFA. Then $\mathcal{L}(A)$ is empty iff $\hat{\delta}\,s\,x \notin F$ for all $x : \Sigma^*$. Since $\hat{\delta}s$ is a classifier, this is a decidable property (Corollary 5.5). ∎

Remark 5.7 The proof of Corollary 5.6 is essentially the proof of decidability of emptiness given by Rabin and Scott [15].

As mentioned above, every DFA yields a classifier for its language. We now show that a classifier for a given decidable language L contains all the information required to construct a DFA accepting L.

Lemma 5.8 Let $f : \Sigma^* \to Q$ be a classifier. Then the image of f can be constructed as a subtype of Q.

Proof By Corollary 5.5, we have that $\exists x \in \Sigma^*.\, f\,x = q$ 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.

Theorem 5.9 (Myhill-Nerode) 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 (Lemma 5.8). Since g is surjective, it has a right inverse g^{-1} (Lemma 2.1). It is straightforward to verify that the DFA $(f(\Sigma^*), s, F, \delta)$ where

$$\begin{aligned} s &:= g\,\varepsilon \\ F &:= \{\, q \mid g^{-1}q \in L \,\} \\ \delta\,q\,a &:= g((g^{-1}q) \cdot a) \end{aligned}$$

accepts the language L. ∎

We remark that in order to use Theorem 5.9 for showing that a language is regular, one first has to show that the language is decidable. It turns out that this restriction is unavoidable in a constructive setting. Let P be some independent proposition. Then $P \vee \neg P$ is not provable. Now consider the language $L := \{ w \in \Sigma^* \mid P \}$. Save for the decidability requirement on L, the constant function from Σ^* into the unit type is a regular classifier for L. If Theorem 5.9 were to apply, the resulting DFA would allow us to decide $\varepsilon \in L$ and consequently obtain a proof of $P \vee \neg P$.

For the translation from two-way automata to one-way automata, the restriction to decidable languages poses no problem since the language of a two-way automaton is easily shown to be decidable.

6 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 [15]. 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 [14] 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 [15, 17, 20]. Marker symbols allow the detection of the word boundaries and allow 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 [21].

Definition 6.1 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
- $\delta_\triangleright : Q \to 2^{Q \times \{\mathsf{L},\mathsf{R}\}}$ is the **transition function** for the left marker

- $\delta_\triangleleft : Q \to 2^{Q \times \{\mathsf{L},\mathsf{R}\}}$ is the **transition function** for the right marker

Let $M = (Q, s, F, \delta, \delta_\triangleright, \delta_\triangleleft)$ be a 2NFA. 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:

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

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

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

$$\mathcal{L}(M) := \{\, x \mid \exists q \in F.\, (s, 1) \xrightarrow[x]{}{}^* (q, |x| + 1) \,\}$$

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.[2] Hence, $\mathcal{L}(M)$ 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.

Definition 6.2 A **deterministic two-way automaton (2DFA)** is a 2NFA $(Q, s, F, \delta, \delta_\triangleright, \delta_\triangleleft)$ where $|\delta_\triangleleft\, q| \le 1$, $|\delta_\triangleright\, q| \le 1$, and $|\delta\, q\, a| \le 1$ for all $q : Q$ and $a : \Sigma$.

Fact 6.3 For every n-state DFA there is an n-state 2DFA that accepts the same language and only moves its head to the right.

Remark 6.4 While Fact 6.3 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

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

the acceptance condition for 2FAs, where the word remains constant throughout the computation.

The rest of the paper is devoted to the translation of two-way automata to one-way automata. There are several such translations in the literature. Vardi [20] gives a simple construction that takes as input some 2NFA M and yields an NFA accepting $\overline{\mathcal{L}(M)}$. 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 [17] gives a translation from 2DFAs to DFAs that incurs only an exponential blowup. Building on ideas from [20], we adapt the construction to 2NFAs.

We first present the translation to NFAs since it is conceptually 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 [17].

7 Vardi Construction

Let $M = (Q, s, F, \delta, \delta_{\triangleright}, \delta_{\triangleleft})$ be a 2NFA. We construct an NFA accepting $\overline{\mathcal{L}(M)}$. Vardi [20] formulates the proof for 2NFAs without markers. We adapt the proof to 2NFAs with markers. The main idea is to define certificates for the nonacceptance 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 7.1 A **negative certificate** for a word x is a set $C \subseteq C_x$ satisfying:

N1. $(s, 1) \in C$

N2. If $(p, i) \in C$ and $(p, i) \to_x (q, j)$, then $(q, j) \in C$.

N3. If $q \in F$ then $(q, |x| + 1) \notin C$.

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 7.2 Let $x \in \Sigma^*$. There exists a negative certificate for x iff $x \notin \mathcal{L}(M)$.

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

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

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 7.3 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 $(q, \mathsf{R}) \in \delta_\triangleright\, p$.
- (U,V) is **$\triangleleft$-closed** if $q \in U$ whenever $p \in V$ and $(q, \mathsf{L}) \in \delta_\triangleleft\, p$.
- (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 $N = (Q', S', F', \delta')$ that incrementally checks the closure conditions defined above:

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

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.

For this we need to make the notion of accepting runs (of N) explicit. For many results on NFAs this is not necessary since the recursive definition of acceptance allows for proofs by induction on the input word. However, for two-way automata, the word remains static throughout the computation. Having a matching declarative acceptance criterion for NFAs makes it easier to relate the two automata models.

We define an inductive relation $\mathsf{run} : \Sigma^* \to Q \to Q^* \to \mathsf{Prop}$ relating words and non-empty sequences of states:

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

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 $|x|+1$. In the following 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 7.4 $x \in \mathcal{L}(N)$ iff there exists an accepting run for x.

Lemma 7.5 $x \in \mathcal{L}(N)$ iff there exists a negative certificate for x.

Proof By Lemma 7.4, it suffices to show that there exists an accepting run iff there exists a negative certificate.

"⇒" Let $(r_i)_{i \leq |x|}$ be an accepting run of N on x. We define a negative certificate C for x where $C_0 := (r_0).1$ and $C_{i+1} := (r_i).2$.

"⇐" If C is a negative certificate for x we can define a run $(r_i)_{i \leq |x|}$ for x on M where $r_0 := (C_0, C_1)$ and $r_{i+1} := (C_i, C_{i+1})$. ∎

Remark 7.6 The formalization of the lemma above is a straightforward but tedious proof of about 60 lines.

Lemma 7.7 $\mathcal{L}(N) = \overline{\mathcal{L}(M)}$.

Proof Follows immediately with Lemma 7.2 and Lemma 7.5. ∎

Theorem 7.8 For every n-state 2NFA M there exists an NFA accepting $\overline{\mathcal{L}(M)}$ and having at most 2^{2n} states.

If one wants to obtain a DFA for $\mathcal{L}(M)$ 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 $|Q|$.

8 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 [17]. In [17], the proof is given for 2DFAs without end-markers. Building on ideas form Vardi [20], we adapt the proof to 2NFAs with end-markers.

We fix some 2NFA $M = (Q, s, F, \delta, \delta_{\triangleright}, \delta_{\triangleleft})$ for the rest of this section. In order to construct a DFA for $\mathcal{L}(M)$, it suffices to construct a classifier for $\mathcal{L}(M)$ (Theorem 5.9). 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 $\mathcal{L}(M)$.

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 of states 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 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 on $Q \times Q$. This is all the information required about x to determine whether $xz \in \mathcal{L}(M)$. This information can be recorded in 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. To show that $\mathcal{L}(M)$ is regular, it suffices to show that T is right-congruent and refines $\mathcal{L}(M)$.

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**:

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

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

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

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 $\mathcal{L}(M)$, 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 8.1 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 8.2 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 $\xrightarrow{k'}_x$-path from (p,i) to (q,j). ∎

Lemma 8.2 can be turned into an equivalence if $k' \geq k$. We state this equivalence in terms in terms of transitive closure since for most parts of the development the concrete path is irrelevant.

Lemma 8.3 Let $i \leq k \leq j$ and let $k' \geq k$. Then $(p,i) \xrightarrow{k'}_x{}^* (q,j)$ iff there exists some p' such that $(p,i) \xrightarrow{k}_x{}^* (p',k) \xrightarrow{k'}_x{}^* (q,j)$.

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 $Tx = Ty$.

Lemma 8.4 Let $p,q : Q$ and let $x,y,z : \Sigma^*$ such that $Tx = Ty$. Then for all $k \geq 1$, $i \leq |z|+1$, and $1 \leq j \leq |z|+1$, we have

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

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 8.2 the path can be split such that:

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

Thus, $(p,p') \in (Tx).2$ by Lemma 8.1. Applying Lemma 8.1 again, we obtain $(p,|y|) \xrightarrow{|y|+1}_{yz}{}^* (p',|y|+1)$. 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 there exist p' and i' such that $(p,|x|+i) \xrightarrow{|x|+k}_{xz} (p',|x|+i')$. But then $(p,|y|+i) \xrightarrow{|y|+k}_{yz} (p',|y|+i')$ and the claim follows by induction hypothesis. ∎

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

Lemma 8.5 T refines $\mathcal{L}(M)$.

Proof Fix $x,y : \Sigma^*$ and assume $Tx = Ty$. By symmetry, it suffices to show $y \in \mathcal{L}(M)$ whenever $x \in \mathcal{L}(M)$. If $x \in \mathcal{L}(M)$, then $(s,1) \xrightarrow{|x|+2}_x{}^* (p,|x|+1)$ for some $p \in F$. We show $y \in \mathcal{L}(M)$ by showing $(s,1) \xrightarrow{|y|+2}_y{}^* (p,|y|+1)$. By Lemma 8.3, there exists a state q such that:

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

We can simulate the first part on y using Lemma 8.1 and the second part using Lemma 8.4. ∎

Lemma 8.6 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

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

Since $|xa| + 1 = |x| + 2$ this follows immediately with Lemma 8.4. It remains to show $(T\,xa).1 = (T\,ya).1$. By symmetry, it suffices to show:

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

By Lemma 8.3, 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 8.1. ∎

Note that Lemma 8.4 is used very differently in the proofs of Lemma 8.5 and Lemma 8.6. 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 Theorem 5.9 and the two lemmas above, we obtain:

Theorem 8.7 Let M be a 2NFA with n states. Then there exists a DFA with at most 2^{n^2+n} states accepting $\mathcal{L}(M)$.

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

Corollary 8.9 Let M be a 2DFA with n states. Then there exists a DFA with at most $(n + 1)^{(n+1)}$ states accepting $\mathcal{L}(M)$.

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 8.6) we can construct the type $T(\Sigma^*)$ (Lemma 5.8). By Theorem 5.9, it suffices to show that $T(\Sigma^*)$ has at most $(|Q| + 1)^{(|Q|+1)}$ elements.

Let $(A, R) : T(\Sigma^*)$. Then $T\,x = (A, R)$ for some $x : \Sigma^*$. We show that A has at most one element. Assume $p, q \in A$. 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 $(A, R) : T(\Sigma^*)$, $(i(A, R)).1$ is defined to be the unique element of A or $\perp$ if $A = \emptyset$. The definition of $(i(A, R)).2$ is analogous. The claim then follows since $Q_\perp \times (Q_\perp)^Q$ has exactly $(|Q| + 1)^{(|Q|+1)}$ elements. ∎

9 Conclusion

We have shown how results about two-way automata can be formalized in Coq with reasonable effort. The translation from 2NFAs to DFAs makes use of a constructive variant of the Myhill-Nerode theorem that is interesting in its own right. When spelled out in detail, the constructions involved become fairly delicate. The formalization accompanying this paper matches the paper proofs fairly closely and provides additional detail.

Even though both Shepherdson [17] and Vardi [20] consider two-way automata without end-markers, the changes required to handle two-way automata with end-markers are minimal. Perhaps surprisingly, the translation to NFAs for the complement becomes simpler and more 'symmetric' when end-markers are added. The original construction [20] uses $2^Q + 2^Q \times 2^Q$ as the type of states while the construction in Section 7 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.

Automata are a typical example of a dependently typed mathematical structure. Our representation of finite automata relies on dependent record types and on finite types being first-class objects. Paulson [13] formalizes finite automata in Isabelle/HOL using heriditarily finite (HF) sets to represent states. Like finite types, HF sets have all the closure properties required for the usual constructions on finite automata. Due to the absence of dependent types, the definition

of DFAs in [13] 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). Thus, the natural typing of DFAs is lost.

We also use dependent types in the representation of two-way automata. The possible configurations of a two-way automaton are represented as a word-indexed collection of finite types. The truncation of natural numbers to bounded natural numbers mentioned in Section 6 allows us to recover the separation between stating lemmas (e.g. Lemma 8.4) and establishing that all indices stay within the correct bounds, thus avoiding many of the problems commonly associated with using dependent types.

Acknowledgments

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 one of the anonymous referees for his helpful comments.

References

[1] Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.): Theorem Proving in Higher Order Logics (TPHOLs 2009), LNCS, vol. 5674. Springer (2009)

[2] Berghofer, S., Reiter, M.: Formalizing the logic-automaton connection. In: Berghofer et al. [1], pp. 147–163

[3] Braibant, T., Pous, D.: Deciding Kleene algebras in Coq. Log. Meth. Comp. Sci. 8(1:16), 1–42 (2012)

[4] 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)

[5] Coquand, T., Siles, V.: A decision procedure for regular expression equivalence in type theory. In: Jouannaud, J.P., Shao, Z. (eds.) CPP. LNCS, vol. 7086, pp. 119–134. Springer (2011)

[6] 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)

[7] Garillot, F., Gonthier, G., Mahboubi, A., Rideau, L.: Packaging mathematical structures. In: Berghofer et al. [1], pp. 327–342

[8] 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). Lecture Notes in Computer Science, vol. 4732, pp. 86–101. Springer (2007)

[9] Gonthier, G., Mahboubi, A., Tassi, E.: A Small Scale Reflection Extension for the Coq system. Rapport de recherche RR-6455, INRIA (2008)

[10] Hedberg, M.: A coherence theorem for Martin-Löf's type theory. J. Funct. Program. 8(4), 413–436 (1998)

[11] Kozen, D.: Automata and computability. Undergraduate texts in computer science, Springer (1997)

[12] Nipkow, T., Traytel, D.: Unified decision procedures for regular expression equivalence. In: Klein, G., Gamboa, R. (eds.) Interactive Theorem Proving (ITP 2014). LNCS, vol. 8558, pp. 450–466. Springer (2014)

[13] 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)

[14] Pighizzini, G.: Two-way finite automata: Old and recent results. Fundam. Inform. 126(2-3), 225–246 (2013)

[15] Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. 3(2), 114–125 (1959)

[16] Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two way finite automata. In: Lipton, R.J., Burkhard, W.A., Savitch, W.J., Friedman, E.P., Aho, A.V. (eds.) Proc. 10th Annual ACM Symp. on Theory of Computing. pp. 275–286. ACM (1978)

[17] Shepherdson, J.: The reduction of two-way automata to one-way automata. IBM J. Res. Develp. 3 (1959)

[18] The Coq Development Team: `http://coq.inria.fr`

[19] Traytel, D., Nipkow, T.: Verified decision procedures for MSO on words based on derivatives of regular expressions. J. Funct. Program. 25, 1–30 (2015)

[20] Vardi, M.Y.: A note on the reduction of two-way automata to one-way automata. Inf. Process. Lett. 30(5), 261–264 (1989)

[21] Vardi, M.Y.: Endmarkers can make a difference. Inf. Process. Lett. 35(3), 145–148 (1990)

[22] 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)