Büchi Automata in Coq

Moritz Lichter

October 25, 2016

Abstract

Büchi automata are a well-known automata model for infinitely long words. We give a formalization of Büchi automata in the constructive type theory in Coq. While the word problem becomes undecidable for Büchi automata, we can decide language emptiness. Büchi automata are known to be closed under boolean operations in classical mathematics. In contrast to NFAs on finite words, Büchi automata cannot be made deterministic and thus the complement construction gets more difficult. In order to prove the closure properties we need assumptions for combinatorial reasoning on infinite words. These assumptions are weaker then excluded middle. We follow the complementation proof idea originally given by Büchi and analyze which assumptions are sufficient to prove its correctness.

1 Introduction

Büchi automata are an automata model for infinitely long words introduced by Büchi. These words are sometimes called infinite words, ω-words or sequences. We will refer to them as sequences. Büchi automata were motivated by their usage in the decidability proof of Monadic Second Order Logic on the successor relation (known as S1S) and have e.g. applications in model checking.

An NFA accepts a string if there is a run on the NFA ending in a final state. Essentially, Büchi automata are NFAs with a different acceptance criterion. Because the run of an NFA on a sequence has no end, we have no last state of the run. The Büchi acceptance criterion requires a final state to be visited infinitely often by the NFA. To distinguish the language of strings recognized by an NFA from the language of sequences, we will speak of the string language and the Büchi language of an NFA.

Many proofs given in the literature are on an rather informal level. We will formalize NFAs with Büchi acceptance in the constructive type theory of Coq. We refine existing proofs:

1. It is decidable whether the Büchi language of an NFA is empty. In the case it is not empty, we can give a sequence belonging to the Büchi language. This can be proven completely constructively.

2. Büchi automata are closed under union and intersection. The union construction is very similar to the sum construction known for string languages while the intersection construction is different (and not constructively).

3. Büchi automata are closed under complement. This proof is not constructively, too.

We were not able to carry out the proofs without assumptions. We need assumptions to reason about the infiniteness of sequences: while in classical logic the existence of one final state occurring infinitely often in a sequence is equivalent to infinitely many final states in the sequence (as long as there are only finitely many states), in intuitionist logic the second one is weaker because there is no possibility to compute the required state. But e.g. the closure under intersection relies on the possibility to go from the second to the first property. Therefore we need an assumption (called (A1)).

NFAs on strings can be made deterministic using the power set construction. For the Büchi acceptance this is not possible. The Büchi acceptance on DFAs is strictly weaker than on NFAs. This makes the complementation of Büchi languages more difficult. There are many different complement constructions for Büchi languages motivated by the number of required states. Because we do not care about the size of the complement automaton, we formalized the sometimes called Ramsey-based approach. The proof breaks in two parts. In the first one we show that some finitely many Büchi languages are compatible with the language, so they are either part of it or of the complement. In the second part we show totality of these languages, so they cover all sequences. While for the first part we need no further assumption then (A1), we were not able to reduce the second part to it. We present two variants of the totality proof using different assumptions:

1. Using Ramsey's Theorem (original proof of Büchi): This is a very short and elegant proof and requires Ramsey's Theorem as assumption.

2. Without Ramsey's Theorem: [1] gives a totality proof without using Ramsey's Theorem. Ramsey's Theorem encapsulates infinite combinatorics from the totality proof nicely. In this proof more infinite combinatorics is involved in the proof. We need a different assumption (A2) similar to (A1), which essentially does not require decidability but only propositional decidability.

Both assumptions, Ramsey's Theorem or (A2) are strong enough to derive (A1). But Ramsey's Theorem and (A2) themselves do not imply each other: Ramsey's Theorem makes a stronger combinatorial statement while it requires the input to be decidable, (A2) is combinatorial weaker and does not require decidability.

As consequences of the complementation theorem we can prove decidability of language universality, inclusion and equality for Büchi languages.

The memo is organized as follows: We first develop some basic techniques for sequences and sequence languages in section 2. Then we introduce, relate and discuss the assumptions (A1), (A2) and Ramsey's Theorem in more detail in section 3. Next we introduce Büchi acceptance of NFAs in section 4. We begin with the proofs of closure under union and intersection in section 5 and of decidability of language emptiness in section 6. Before we can prove the complementation theorem in section 8 we need some constructions on NFAs in section 7.

2 Sequences and Strings

Sequences over a type X are functions: $\texttt{Seq}\ X := \mathbb{N} \to X$. **Strings** over X are nonempty prefixes of sequences over X: $\texttt{String}\ X := \texttt{Seq}\ X \times \mathbb{N}$. The first component gives the sequence and the second one the last index which belongs to the string. Conceptually, once can still think of strings as nonempty lists over X.

Sequence g over $\mathbb{N}$ is called **strictly monotone**, if $\forall n, g(n) < g(n+1)$. We obtain the **subsequence of w induced by g** as $w \circ g$. Similar, we can build substrings of a sequence. A string f over $\mathbb{N}$ is called **strictly monotone** if $\forall n < |f|, f[n] < f[n+1]$. Then the substring of w induced by f is the string of length $|f|$ whose n-th character is $w(f[n])$. For simplicity, we write $w \circ f$, too.

We defined strings as nonempty prefixes of sequences. While this seams unnecessarily complicated, this definition makes it easy to treat sequences and strings uniformly. Because strings are defined as nonempty, we cannot represent the empty string. This removes a lot of trouble in some constructions. Since we are interested in strings only as parts of sequences, this is no restriction. If we speak of strings in the following, we *only allow nonempty strings.*

2.1 Basic Operations on Sequences

$w(i,j)$ is the string contained in the sequence w starting at index i (inclusive) to index j (exclusive). If $i \geq j$ then $w(i,j) = [i]$. $v +\!\!+\ w$ is the concatenation of two strings, $v +\!\!+\!\!+\ w$ prepends the string v to the sequence w. An element x **occurs infinitely often** in a sequence w if $\forall n, \exists m, m \geq n \wedge w(n) = x$.

2.2 String and Sequence Languages

A **string language** over a type X is a predicate on strings over X. A **sequence language** over a type X is a predicate on sequences over X. We define the **universal and empty sequence language** on X:

$$X^\omega := \lambda\_.\top \text{ and } \emptyset := \lambda\_.\bot$$

Union, intersection and complement are defined using conjunction, disjunction and negation, e.g. the complement is $L^C = \lambda w.\neg L(w)$. Analogues subset and equality on languages is defined using implication and equivalence.

A sequence language L is called **extensional** if given a sequence $w \in L$ it contains all pointwise equal sequences:

$$\text{extensional } L := \forall w, w \in L \to \forall w', (\forall n, w(n) = w'(n)) \to w' \in L.$$

Because strings are defined as nonempty prefixes of sequences, we need a notion of extensionality for string languages too. A string language L is extensional if $\forall v, v \in L \to \forall v', |v| = |v'| \to (\forall n \leq |v|, v[n] = v'[n]) \to v' \in L$.

Extensionality of languages will be useful later. Fortunately all languages occurring later have this property.

2.3 Infinite Filter

Given a sequence w over a type X, a decidable predicate P on X and a proof that there are infinitely many positions in w at which P holds ($\forall n, \exists m \geq n, P(w(m))$), we want a function $\mathsf{filter}_\infty(w, P) : \mathsf{Seq}\ \mathbb{N}$, such that

- $\mathsf{filter}_\infty(w, P)$ is strictly monotone,
- P holds for all positions in the induced subsequence of w

$$\forall n, P((w \circ \mathsf{filter}_\infty(w, P))(n))$$

- and $\mathsf{filter}_\infty(w, P)$ contains all positions in w at which P holds:

$$\forall m, \forall k, \mathsf{filter}_\infty(w, P)(m) < k < \mathsf{filter}_\infty(w, P)(m+1) \to \neg P(w(k))$$

Because P is decidable, filter_∞ is unique (up to extensionality).

We establish filter_∞ recursively using constructive choice for $\mathbb{N}$ and recursion trees. Assume we know $\mathsf{filter}_\infty(w, P)(n)$, we construct an $m > \mathsf{filter}_\infty(w, P)(n)$ using constructive choice such that $P(w(m))$. With recursion trees we can go backward towards $\mathsf{filter}_\infty(w, P)(n)$ to find the smallest $m' > \mathsf{filter}_\infty(w, P)(n)$, such that $P(w(m'))$. So we obtain $\mathsf{filter}_\infty(w, P)(n + 1) = m'$. The same technique applies to the base case $\mathsf{filter}_\infty(w, P)(0)$ with $m \geq 0$.

2.4 History Filter

This is a slightly different version of the infinite filter. Given a sequence w over X and a decidable and extensional string language (a predicate) $P : \texttt{String}\ X \to \mathbb{P}$, we want function $\mathsf{filter}_{\mathsf{history}}(w, P) : \texttt{Seq}\ \mathbb{N}$ such that

- $\mathsf{filter}_{\mathsf{history}}(w, P)$ is strictly monotone,
- P holds for all prefixes of the induced subsequence of w

$$\forall n, P((w \circ \mathsf{filter}_{\mathsf{history}}(w, P))(0, n)).$$

- and $\mathsf{filter}_{\mathsf{history}}(w, P)(0) = 0$ (which is rather special but required in the later application).

This function is not unique, because we have no completeness requirement. It is not clear, what such a requirement should be, because not including a position of w may allow including another one. Additionally, it is not obvious what property w needs to fulfill, such that we can find the required function.

Assume w fulfills the following two properties:

1. $P(w(0, 1))$ and
2. given any strictly monotone string g such that $P(w \circ g)$, then there is a bigger m, such that P holds for the substring induced by appending m to g:

$$\forall (g : \texttt{String}\ \mathbb{N}), \text{strictly monotone } g \to P(w \circ g) \to \\ \exists m, m > g[|g| - 1] \wedge P(w \circ (g +\!\!+ [m])$$

Now we can construct $\mathsf{filter}_{\mathsf{history}}$ recursively using constructive choice for $\mathbb{N}$. The base case is given by the first property: $\mathsf{filter}_{\mathsf{history}}(w, P)(0) = 0$. If we know $(\mathsf{filter}_{\mathsf{history}}(w, P))(0, n)$, we can find the $(n + 1)$-th position using the second property.

2.5 Infinite Concatenation

At some point we need to concat a sequence of strings W by concatenating all string in W to a flat sequence (the infinite version of flattening a list of lists). Here it is important that our strings are nonempty, otherwise we could not be sure that the result is a sequence (if only finitely many strings were nonempty).

Given $W : \texttt{Seq}\ (\texttt{String}\ X)$ we want a function $\mathsf{concat}_\infty(W) : \texttt{Seq}\ X$ such that

- $\mathsf{concat}_\infty(W)(0) = (W(0))[0]$ and

- if $\mathsf{concat}_\infty(W)(n)$ is the last character of $W(m)$, then $\mathsf{concat}_\infty(W)(n+1) = (W(m+1))[0]$. Otherwise if it is the k-th character of $W(m)$, $\mathsf{concat}_\infty(W)(n+1) = (W(m))[k+1]$

To make the second requirement completely formal, we want to build another function $\mathsf{concatI}_\infty(W) : \texttt{Seq}\ (\mathbb{N} \times \mathbb{N})$. The intuition is, that $\mathsf{concatI}_\infty(W)(n)$ locates the position of the string in W and the position of the character in this string at the n-th position of $\mathsf{concat}_\infty(W)$:

$$\mathsf{concat}_\infty(W) := \lambda n.\text{let } (i,k) := \mathsf{concatI}_\infty(W)(n) \text{ in } (W(i))[k]$$

$\mathsf{concatI}_\infty(W)$ can be specified completely formally and uniquely (up to extensionality):

- $\mathsf{concatI}_\infty(W)(0) = (0,0)$ and
- Let $\mathsf{concatI}_\infty(W)(n) = (i,k)$. Then

  $$\mathsf{concatI}_\infty(W)(n+1) = \begin{cases} (i+1,0) & \text{if } k = |W(i)| - 1 \\ (i,k+1) & \text{otherwise} \end{cases}$$

  In the first case the i-th strings ends and we need to switch to the next string (which we can do safely, because it is nonempty) while in the second case there is another character in the i-th string.

The implementation is straightforward. Now we can define the **ω-iteration of a string** formally

$$v^\omega := \mathsf{concat}_\infty(\lambda\_.v).$$

2.6 ω-Iteration of a String Language

Given a string language L over X we want to the **ω-iteration** of L. This is a sequence language L^ω which repeats strings from L infinitely often. We have two definitions. Given a sequence w, it belongs to the ω-iteration of L if

1. there is a strictly monotone sequence f which partitions w into infinitely many strings belonging to L:

   $$L^\omega := \lambda w.\exists \text{ strictly monotone } f, f(0) = 0 \wedge \forall n, L\,(w(f(n), f(n+1)))$$

2. there is a sequence W of strings belonging to L such that the infinite concatenation of W is pointwise equal to w:

   $$\begin{aligned} L^{\omega 2} :=& \lambda w, \exists (W : \texttt{Seq}\ (\texttt{String}\ X)), \\ & (\forall n, L(W(n))) \wedge (\forall n, w(n) = \mathsf{concat}_\infty(W)(n)) \end{aligned}$$

In general, the second definition is weaker than the first one: $L^{\omega 2} \subseteq L^{\omega}$. If L is extensional, we can prove both definition to be equivalent:

$$\text{extensional } L \to L^{\omega 2} = L^{\omega}.$$

It will be useful to switch to the second definition later.

3 Necessary Assumptions

We need the following assumption: given a finite coloring of $\mathbb{N}$ then there is a monochromatic strictly monotone sequence over $\mathbb{N}$. Technically, this assumption is needed in two variations:

Sequences over Finite Types (A1) Given a sequence w over a finite type X, there is an x such that x occurs infinitely often in w.

$$\forall X : \text{finite type}, \forall f : \texttt{Seq } X, \exists x, x \text{ infinitely often in } f.$$

Equivalence Relations on $\mathbb{N}$ with Finite Index (A2) Let $\sim$ be an equivalence relation over a type X such that

1. $\sim$ is propositionally decidable: $\forall xy, x \sim y \lor x \not\sim y$
2. $\sim$ is of finite index. That is, there is a number n, such that all strings over X longer than n contain two equivalent elements:

$$\exists n, \forall l : \texttt{String } X, |l| > n \to \exists i < j < |l|, l[i] \sim l[j]$$

Given a sequence w over $\mathbb{N}$, there is a strictly monotone sequence f such that the induced subsequence of w contains only members of one equivalence class:

$$\exists f, f \text{ strictly monotone } \land \forall n, (w \circ f)(0) \sim (w \circ f)(n)$$

3.1 (A2) implies (A1)

From (A2) one can derive (A1). Let X be a finite type and w a sequence over X. Take equality as equivalence relation. Equality on finite types is decidable and so propositionally decidable. It is of finite index, because every string over X longer than the cardinality of X has at least two positions with the same element of X. So we can apply (A2) and obtain the strictly monotone sequence f. Then $(w \circ f)(0)$ occurs infinitely often in w.

Nevertheless it is useful to have (A1) as an assumption on its own, since many results only depend on (A1) and not on (A2).

3.2 For some $\sim$, (A1) implies (A2)

Let $\sim$ be an equivalence relation which can be written as $R : X \to F$ for a finite type F. For $\sim := \lambda xy, R(x) = R(y)$ one can derive (A2) from (A1).

Let w be a sequence over X. From (A1) one gets an element a of type F which occurs infinitely often in $R \circ w$. Then we get the strictly monotone sequence required for (A2) as $\mathsf{filter}_\infty(w, \lambda x.R(x) = a)$. We can apply filter_∞, because equality on F is decidable and we have a proof (from (A1)) that there are infinitely many positions at which the filter predicate holds.

3.3 (A1) for Predicates (A1P)

Let X be a finite type, P a decidable predicate on X and w a sequence over X. If there are infinitely many positions of w at which P holds, then there is an element x occurring infinitely often in w for which P holds:

$$(\forall n, \exists m \geq n, P(w(m))) \to \exists x, x \text{ infinitely often in } w \wedge P(x).$$

This formulation is equivalent to (A1): $(A1) \leftrightarrow (A1P)$. $\to$ can be obtained by applying (A1) on $\mathsf{filter}_\infty(f, P)$. $\leftarrow$ is simple using the predicate $True$.

3.4 Ramsey's Theorem

We will see that Ramsey's Theorem is used in classical proofs later while it is more effort to prove the same thing using (A2).

Ramsey's Theorem: Let U be an infinite set, C a finite set of colors and q a coloring (using colors of C) of all subsets of U containing exactly two elements. Then there is an infinite subset $M \subseteq U$ colored in one color: there is a $c \in C$ such that $q(\{x, y\}) = c$ for all $x, y \in M$ such that $x \neq y$.

Constructive Formulation of Ramsey's Theorem for $\mathbb{N}$: Because we do not have sets in the type theory of Coq, we use the following formulation of Ramsey's Theorem for $U = \mathbb{N}$. Given a finite type C and a coloring $q : \mathbb{N} \to \mathbb{N} \to C$ such that

$$\forall nm, q(n, m) = q(m, n)$$

then

$$\exists (c : C), \exists (M : \mathtt{Seq}\ \mathbb{N}), M \text{ strictly monotone} \wedge \forall nm, n \neq m \to q(M(n), M(m)) = c.$$

The function q can be transformed to a coloring of sets with two elements, because $q(n, m) = q(m, n)$. The infinite subset of $\mathbb{N}$ is represented by a strictly monotone sequence M and the property that the color of sets with two (and not the only one) element is equal is asserted by the restriction $n \neq m$.

3.4.1 Relationship of Ramsey's Theorem to (A2)

We claim none of Ramsey's Theorem and (A2) implies the other one constructively.

- (A2) $\nrightarrow$ Ramsey's Theorem: The important point in Ramsey's Theorem is that it gives the closure property that all pairs in the selected subset are colored equally while the coloring is arbitrary. If one would like to derive Ramsey's Theorem from (A2), one needs to transform the coloring into an equivalence relation. But because the coloring is arbitrarily, we have no chance to build a transitive relation.
- Ramsey's Theorem $\nrightarrow$ (A2): It would be nice if (A2) was weaker than Ramsey. But to apply Ramsey in the setting of (A2) we need to construct a coloring for the equivalence relation (e.g. by constructing quotients). But since the equivalence relation is only propositionally decidable, we cannot do this.

3.4.2 Ramsey's Theorem implies (A1)

Using Ramsey's Theorem we can prove (A1). Given a finite type X and a sequence w over X. We apply Ramsey's Theorem with colors **2** and the coloring

$$q := \lambda n\ m.\ \text{if } (w(n) = w(m)) \text{ then } 1 \text{ else } 2$$

We can decide the if condition, because X is finite. It is easy to see that $\forall nm, q(n, m) = q(m, n)$. From Ramsey we get that a strictly monotone sequence U over $\mathbb{N}$ and a color x which colors all two distinct positions in M equally.

- If $x = 2$, there are infinitely may different elements in f which contradicts finiteness of X.
- If $x = 1$, this implies that all positions in M are equal, so the element $(w \circ M)(0)$ occurs infinitely often in w.

3.4.3 For some $\sim$, Ramsey's Theorem implies (A2)

For equivalence relations as in 3.2 Ramsey's Theorem implies (A2). Because Ramsey implies (A1) and for this special class of equivalence relations (A1) implies (A2), Ramsey's Theorem implies (A2) for such relations.

3.5 Summary

The relationship between the assumptions can be visualized as follows. Each $\leftarrow$ is an implication, where the opposite direction does not hold:

$$\begin{tikzcd}
 & \text{XM} \arrow[dl] \arrow[dr] & & \\
\text{Ramsey} \arrow[dr] & & \text{(A2)} \arrow[dl] & \\
 & \text{(A1)} \arrow[r, leftrightarrow] & \text{(A1P)} &
\end{tikzcd}$$

4 NFAs

A **nondeterministic finite automaton (NFA)** A over a finite type X is a structure consisting of

- a finite type $\mathsf{state}(A)$
- a decidable transition relation $T : \mathsf{state}(A) \to X \to \mathsf{state}(A) \to \mathbb{P}$
- a list of final states
- a list of initial states

A **run** of A is a sequence over $\mathsf{state}(A)$.

4.1 Büchi Acceptance

We call a run r

- **valid** on a sequence w over X if it agrees with the transition relation: $\forall n, T(r(n))(w(n))(r(n+1))$,
- **initial** if $r(0)$ is an initial state, and
- **final** if there is a final state s which occurs infinitely often in r.

The NFA A **accepts** w if there is a run r such that r is valid on w, initial and final. The **Büchi language** recognized by A is the sequence language

$$L_B(A) := \lambda(w : \mathtt{Seq}\ X).A \text{ accepts } w.$$

Classically it makes no difference whether a run r is defined to be final

1. if there is a final state occurring infinitely often in r or
2. if there are infinitely many final states in r.

Intuitionistically it makes a difference because we need (A1P) to switch from (2) to (1). The definition we used is the one used commonly(e.g. [6], [1] or [3]). If we defined it using (2), we would be able to show some facts constructively which we cannot using (1) (e.g. the intersection construction). But unfortunately, there are facts which cannot be proven constructively using (2) which we can prove now using (1) (e.g. decidability of language emptiness). So we follow the common definition.

4.2 String Acceptance

We call a run r

- **string-valid** on a string v over X if it agrees with the transition relation up to the length of the input string $\forall n < |v|, T(r(n))(v(n))(r(n+1))$ and
- **string-final** if $r(|v|)$ is a final state.

The NFA A **string-accepts** v if there is a run r such that r is string-valid on v, initial and string-final. The string language recognized by A is

$$L_S(A) := \lambda(v : \texttt{String}\ X).A \text{ string-accepts } v.$$

It is technically easier to use sequences as runs on strings. Because strings are prefixes of sequences, the sequence containing the run is there anyway and the length of the run string is determined by the length of v (to $|v| + 1$). So the length of the run string is redundant and we can just work with sequences (filled with garbage from position $|v| + 2$).

5 Union and Intersection of Büchi Languages

Let A_1 and A_2 be NFAs.

5.1 Closure under Union

There is a function unite_B : NFA $\to$ NFA $\to$ NFA such that

$$L_B(\mathsf{unite}_B(A_1, A_2)) = L_B(A_1) \cup L_B(A_2).$$

The union construction is straightforward using the sum construction and completely constructive.

5.2 Closure under Intersection

There is another function $\mathsf{intersect}_B$: NFA $\to$ NFA $\to$ NFA such that

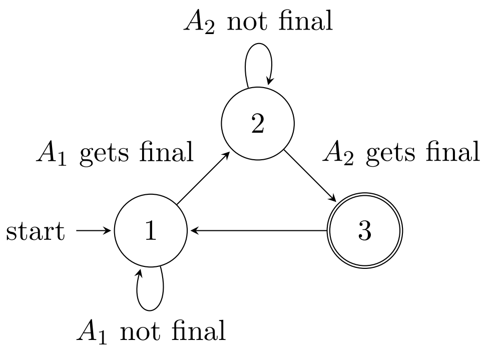
$$L_B(\mathsf{intersect}_B(A_1, A_2)) = L_B(A_1) \cap L_B(A_2).$$

This construction is more difficult and different from the construction for string languages. The NFA $\mathsf{intersect}_B(A_1, A_2)$ needs to run both automata in parallel. So all states of $\mathsf{intersect}_B(A_1, A_2)$ need to contain a state of A_1 and A_2. But when is a state final? Because A_1 and A_2 not necessarily ever reach a final state at the same position even if both automata accept the input sequence, one cannot define the states of $\mathsf{intersect}_B(A_1, A_2)$ to be final if the contained states of A_1 and A_2 are final.

The idea is that $\mathsf{intersect}_B(A_1, A_2)$ waits for a final state of A_1 and A_2 alternatingly. Because $\mathsf{intersect}_B(A_1, A_2)$ needs to remember on which automaton to wait, this piece of information needs to be stored in the states of, too:

$$\mathsf{state}(\mathsf{intersect}_B(A_1, A_2)) := \mathsf{state}(A_1) \times \mathsf{state}(A_2) \times \mathbf{3}$$

If the last component of the current state of $\mathsf{intersect}_B(A_1, A_2)$ is 1, it waits for A_1 to get final and the same for A_2 if the last component is 2. The state 3 is directly passed.



If $\mathsf{intersect}_B(A_1, A_2)$ sees infinitely many final states of A_1 and A_2 it visits infinitely often a state with 3 as last component. So all states with a 3 as last component are final states of $\mathsf{intersect}_B(A_1, A_2)$.

If we want to prove that $L_B(\mathsf{intersect}_B(A_1, A_2)) = L_B(A_1) \cap L_B(A_2)$ there is a certain difficulty. In both directions we can not find the infinitely occurring final state constructively:

Given accepting runs of A_1 and A_2 on a sequence w, we can prove that $\mathsf{intersect}_B(A_1, A_2)$ loops infinitely often through the last component of its states which implies that there are infinitely many final states. Given an accepting run of $\mathsf{intersect}_B(A_1, A_2)$ on w we only know that that both A_1 and A_2 visit final states infinitely often, because $\mathsf{intersect}_B(A_1, A_2)$ loops infinitely often through the last component of its states.

In both cases, concluding that A_1 and A_2 resp. $\mathsf{intersect}_B(A_1, A_2)$ accept w, depends on (A1P).

6 Decidability of Büchi Language Emptiness

In this section we show that it is decidable whether the Büchi language of an NFA is empty or not. We even show a stronger informative decision which gives in the case that the language is nonempty a concrete sequence of the language. Conceptually this section follows [3].

6.1 Trimming of Büchi Automata

A state s of an NFA A is called **accessible**, if s is reachable from an initial state[1]. The state s is called **final coaccessible** if there is a final and valid run of A for some sequence w starting at s.

The state s is called **trim** if it is accessible and final coaccessible. The trim automaton A_T of A is the NFA derived from A when only keeping all trim states of A. It is easy to see, that $L_B(A) = L_B(A_T)$ and that $L_B(A_T) = \emptyset \leftrightarrow \mathsf{state}(A_T) = \emptyset$. So if we want to decide $L_B(A) = \emptyset$, we can decide $\mathsf{state}(A_T) = \emptyset$.

6.2 Decision of a State being Trim

Because the definition of final coaccessible quantifies over a sequence, it is not obvious how to decide whether a state is final coaccessible or not. Here we use some ideas given in [1]. We call a state s of A **final cyclic** if there is a nonempty loop from s to s visiting a final state. Then we want to show the following lemma:

$$s \text{ is final coaccessible } \leftrightarrow \exists s', s' \text{ is reachable from } s \land s' \text{ is final cyclic}$$

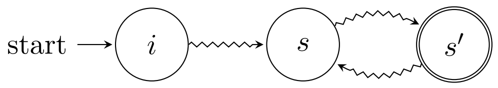
Showing $\leftarrow$ is clear. To show $\rightarrow$ one can see that if there is a valid and final run r and the infinitely often occurring state is s', that s' is reachable from s and that r contains a loop from s' to s'.

Now we reduce the existence of a loop visiting a final state to reachability:

$$s \text{ is final cyclic} \leftrightarrow \exists s', s' \text{ is final} \land s' \text{ is reachable from } s \land s \text{ is reachable from } s'$$

(Note that the loop is not restricted to visiting a state only once.) Then

$$\begin{aligned} s \text{ is trim } \leftrightarrow & \exists i\ s', i \text{ is initial } \land s \text{ reachable from } i \land s' \text{ is final} \land \\ & s' \text{ is reachable from } s \land s \text{ is reachable from } s' \end{aligned}$$



Now It is easy to see that the rewritten condition is decidable, because existential quantifiers over finite types (the states of A) and reachability in NFAs are decidable. So we can decide whether s is trim or not, so we can decide whether there is a trim state and finally whether $L_B(A) = \emptyset$.

[1]Strictly speaking with non empty strings, reachable implies a nonempty path, then s is accessible if it is reachable from an initial state or is initial itself.

6.3 Informative Decision of Language Emptiness

We show

$$\{\exists w, w \in L_B(A)\} + \{L_B(A) = \emptyset\}$$

If $L_B(A) \neq \emptyset$ there is a trim state s. By the equivalences already shown, s is reachable from an initial state on a string v and has a nonempty loop from s to s visiting a final state on a string w. Then $v +\!\!+\, w^\omega \in L_B(A)$.

7 Constructions on NFAs

The following constructions are for example described in [1] and raise no conceptional difficulties in constructive logic. While [1] does the last two constructions in one step, we split them for simplicity.

Let A, A_1 and A_2 be NFAs.

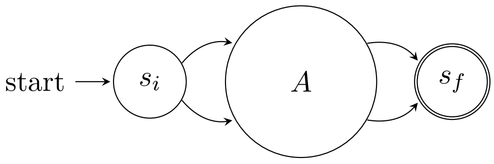
7.1 Normalizing an NFA

An NFA A is **normalized** if A has only one initial state s_i without any entering transition and A has only one final state s_f without any leaving transition.

There is a function $\mathsf{normalize} : \text{NFA} \to \text{NFA}$ such that

$$L_S(\mathsf{normalize}(A)) = L_S(A) \wedge \text{ normalized } \mathsf{normalize}(A).$$

The function $\mathsf{normalize}$ introduces two now states s_i and s_f. Whenever there is a transition $s \xrightarrow{a} s'$ for an initial state s, we allow the transition $s_i \xrightarrow{a} s'$. Vice versa, if there is a transition $s \xrightarrow{a} s'$ for a final s', we allow the transition $s \xrightarrow{a} s_f$. The only new initial state is s_i, the only final s_f.



By construction there is no transition to s_i and from s_f. It is easy to see that this transformation does not change the string language $L_S(\mathsf{normalize}(A)) = L_S(A)$, because only the first and the last transition needs to be adjusted.

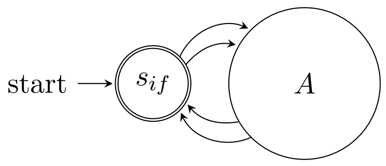
7.2 NFA recognizing $L_S(A)^\omega$

There is a function $\mathsf{nfa}\text{-}\omega\text{-}\mathsf{iter} : \text{NFA} \to \text{NFA}$ such that

$$L_B(\mathsf{nfa}\text{-}\omega\text{-}\mathsf{iter}(A)) = L_S(A)^\omega$$

and $\mathsf{nfa}\text{-}\omega\text{-}\mathsf{iter}(A)$ has a state s_{if} which is the only final and initial one.

The NFA normalize(A) has two states s_i and s_f. The function nfa-ω-iter merges the states s_i and s_f to a new state s_{if}. s_{if} is the only final and initial state of A_B.



$L_B(\text{nfa-}\omega\text{-iter}(A)) \subseteq L_S(A)^\omega$: An accepting run on nfa-ω-iter(A) for a sequence w visits s_{if} infinitely often, because s_{if} is the only final state. Then the run needs to pass through A infinitely often. Each pass from s_{if} through A to s_{if} corresponds exactly to one string of $L_S(A)$. In the formal proof the partition w into the infinitely many strings of $L_S(A)$ can be obtained using filter$_\infty$ very elegantly. The accepting run is filtered for the positions at which it reaches s_{if}.

$L_S(A)^\omega \subseteq L_B(\text{nfa-}\omega\text{-iter}(A))$: Assume we have infinitely many string of $L_S(A)$ together with their runs on normalize(A), then the accepting run for nfa-ω-iter(A) is the infinite concatenation of these runs without the last state. The last state of any of these runs is s_f, which is merged into s_i in nfa-ω-iter(A) and s_i is the first state of the run on the next string.

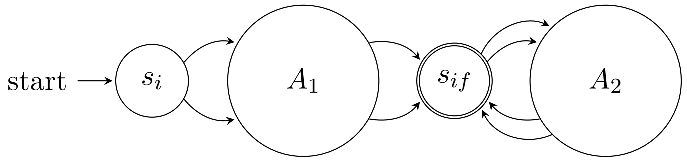
7.3 NFA recognizing $L_S(A_1) \cdot L_B(A_2)$

There is a function nfa-prepend : NFA $\to$ NFA $\to$ NFA such that

$$L_B(\text{nfa-prepend}(A_1, A_2)) = L_S(A_1) \cdot L_B(A_2)$$

if A_2 has a state s_{if}, which is the only initial and final state in A_2.

The NFA normalize(A_1) has the two states s_i and s_f. The function nfa-prepend merges s_f into the state s_{if} of A_2. s_i is the only initial and s_{if} the only final state of nfa-prepend(A_1, A_2).



It is easy to see that we need to prepend for each accepted sequence of nfa-prepend(A_1, A_2) an accepted string from A_1 to come from s_i to s_{if}. Vice versa, if we have an accepting run of nfa-prepend(A_1, A_2), we can split it at the first occurrence of s_{if} in an accepting run of A_1 and of A_2.

As a corollary we obtain that there is a function f : NFA $\to$ NFA $\to$ NFA such that

$$L_B(f(A_1, A_2)) = L_S(A_1) \cdot L_S(A_2)^\omega.$$

We define $f(A_1, A_2) :=$ nfa-prepend$(A_1, \text{nfa-}\omega\text{-iter}(A_2))$.

8 Complementation Theorem

Fix an NFA A. We want to show that there is a function $\mathsf{complement}_B$: NFA $\to$ NFA such that

$$L_B(\mathsf{complement}_B(A)) = L_B(A)^C$$

, so we can construct the NFA recognizing the complement Büchi language of A.

There are different classical ways to construct $\mathsf{complement}_B(A)$ in literature. An overview is given e.g. [10] or [11]. We follow the sometimes called Ramsey-based complementation as e.g. found in [6]. Our translation to constructive type theory follows the same layout. At some points we will need to carry out more details or to work hard on something that is classically easy.

We will define an equivalence relation $\sim$ of finite index on strings, such that each equivalent class can be recognized by an NFA. Moreover, the Büchi languages VW^ω, where V and W are equivalence classes of $\sim$, are total: together they cover all sequences. We will see that the VW^ω are compatible with $L_B(A)$: a VW^ω is either included in $L_B(A)$ or in $L_B(A)^C$ and we will be able to decide this. Using these VW^ω languages, we will construct the complement NFA.

This proof is similar to the original one given by Büchi [4] (but in an logical notation) and gives some insights into the structure of Büchi languages.

8.1 $\sim$ Equivalence Relation on Strings

We say that a string v **transforms** state s_1 to state s_2 if A can run from s_1 to s_2 on v. We write $s_1 \Longrightarrow_v s_2$. If there is a run on which A visits a final state, we say that v **final transforms** s_1 to s_2 and denote this by $s_1 \Longrightarrow_w^F s_2$. Because $\mathsf{state}(A)$ is finite, we can construct the path through A on v given $s_1 \Longrightarrow_v s_2$ or $s_1 \Longrightarrow_v^F s_2$.

We define two strings v and w to be $\sim$ **equivalent**:

$$v \sim w := \forall s s', (s \Longrightarrow_v s' \leftrightarrow s \Longrightarrow_w s') \land \left(s \Longrightarrow_v^F s' \leftrightarrow s \Longrightarrow_w^F s'\right)$$

Intuitively this means that A cannot distinguish between v and w if $v \sim w$. (Note that $\sim$ is a stronger version than $\sim$ in the Myhill-Nerode Theorem). It is easy to see that $\sim$ is a congruence relation: $v \sim w \to \forall u, v +\!\!+\, u \sim w +\!\!+\, u$. Because $\sim$ talks about strings, it is decidable.

The equivalence classes of $\sim$ can be indexed by the finite type

$$E_I := \mathbf{3}^{state(A) \times state(A)}$$

as follows. Let i be of this type. Then the equivalence class denoted by $[\![i]\!]$ is:

$$[\![i]\!] := \lambda w.\forall ss', \begin{cases} s \not\Longrightarrow_w s' & i(s,s') = 1 \\ s \Longrightarrow_w s' \wedge s \not\Longrightarrow^F_w s' & i(s,s') = 2 \\ s \Longrightarrow^F_w s' & i(s,s') = 3 \end{cases}$$

There is a function $\mathsf{class}_\sim$: $\texttt{String}\ X \to E_I$ computing the $\sim$-equivalence class for all strings:

$$\forall (v : \texttt{String}\ X), [\![\mathsf{class}_\sim v]\!]$$

We denote all equivalence classes of $\sim$ as

$$E_\sim := \{[\![i]\!] | i : E_I\}.$$

8.1.1 NFAs for Equivalence Classes of $\sim$

We need that $[\![i]\!]$ is NFA recognizable later. There are two possibilities:

- Use A to recognize string languages: $\lambda v.s \Longrightarrow_v s'$ is recognizable when setting s to the only initial and s' to the only final state. $\lambda v.s \Longrightarrow^F_v s'$ can be recognized similarly by carrying a boolean flag in the states of A which remembers whether a final state of A was seen. With the closure properties of string languages on NFAs we get that we can construct an NFA recognizing $[\![i]\!]$.

- Myhill-Nerode using $\sim$ as classifier: In [2] Doczkal and Smolka show that a decidable string language given a classifier (a right congruent function in a finite type which refines the language) for it is NFA recognizable. Certainly, we can decide whether $v \in [\![i]\!]$ or not because v is finite. As a classifier we take the function which assigns to each string its $\sim$ equivalence class.

8.2 Compatibility of VW^ω

Lets fix some $V, W \in E_\sim$. The next goal is to show that VW^ω is **compatible** with $L_B(A)$:

$$(\exists w, w \in (VW^\omega \cap L_B(A))) \to VW^\omega \subseteq L_B(A)$$

We do not require the VW^ω languages to be a partition of X^ω and in fact they are not necessarily. The constructive proof depends on the given witness w, while $(VW^\omega \cap L_B(A)) \neq \emptyset$ suffices in the classical setting.

While compatibility is rather easy to see informally, one needs to work a lot harder to make it completely formal, but the basic idea stays the same: If $w \in VW^\omega$ it is $w = v +\!\!\!+\ (w_1 +\!\!\!+\ w_2 +\!\!\!+\ \dots)$ for some $v \in V$ and $w_i \in W$. We can switch to this representation according to our second definition $W^{\omega 2}$

in section 2.6, because the equivalence classes of $\sim$ are extensional. Because $w \in VW^\omega$, there is a valid, initial and final run r of A on w.

Take any other $w' = v' +\!\!\!+\!\!\!+ (w'_1 +\!\!\!+ w'_2 +\!\!\!+ \dots) \in VW^\omega$. Because $v \sim v'$ and $w_i \sim w'_i$ we can stitch the constructed partial runs of A on v' and w'_i together to get a valid run r' on w'. r' is initial because it starts with the same state because $v \sim v'$. Showing that r' is final is not constructive completely: there is a final state s in r which occurs infinitely often. From the the definition of $\sim$ we know that A visits a final state on w' at some w'_i if A visits s on the corresponding w_i in w. But we do not know which final state A visits on w'_i. So we can show that there are infinitely many final states in r' only and need (A1P) to conclude that $w' \in L_B(A)$.

Together with the informative decision of language emptiness and NFA recognizability of V and W, we could obtain the corollary

$$\{VW^\omega \subseteq L_B(A)\} + \{VW^\omega \subseteq L_B(A)^C\}$$

but we do not need it for the remaining proof.

8.3 Totality of VW^ω

We want to prove that every sequence is contained in one VW^ω:

$$\forall (w : \mathsf{Seq}\ X), \exists V, W \in E_\sim, w \in VW^\omega$$

The classical proof uses Ramsey's Theorem. Unfortunately, Ramsey's Theorem does not hold in constructive logic. In order to prove totality, we can either assume Ramsey's Theorem and follow the proof first given by Büchi [4] (or e.g. in [6]) or do more infinite combinatorics directly. In this case we follow the proof given by Khoussainov and Nerode in [1]. This argumentation can be formalized using our assumption (A2).

In Section 3 we argued that (A2) and Ramsey's Theorem do not imply each other. So it is interesting to see that we can use both to prove totality. The proof using Ramsey's Theorem is elegant and compact, the proof using (A2) is more complicated. For now fix an arbitrary sequence w over X.

8.3.1 Proof using Ramsey's Theorem

We sketch the classical proof of totality using Ramsey's Theorem[6]:

We color the indices of w with the $\sim$ equivalence classes: Let $i < j$. Then $q(\{i, j\}) := \mathsf{class}_\sim(w(i, j))$ is the equivalence class of $w(i, j)$. Ramsey's Theorem gives an equivalence class W and an infinite $M \subseteq \mathbb{N}$ such that $w(i, j) \in W$ for $i < j \in M$. W.l.o.g. $0 \notin M$. Let $i_0 = \min M$. Then V is the equivalence class of $w(0, i_0)$. Then (up to extensionality) $w = w(0, i_o) +\!\!\!+\!\!\!+ (w(i_0, i_1) +\!\!\!+ w(i_1, i_2) +\!\!\!+ \dots) \in VW^\omega$.

Using the formulation of Ramsey's Theorem for $\mathbb{N}$ in 3.4 the proof is valid constructively (assuming Ramsey for $\mathbb{N}$).

8.3.2 Equivalence Relation $\sim_w$ on Sequence Indices

Let $k_1, k_2, m \in \mathbb{N}$, so they are indicies in w. We define

$$k_1 \sim_w k_2 \text{ (at } m) := m > k_1 \wedge m > k_2 \wedge w(k_1, m) \sim w(k_2, m)$$

$$k_1 \sim_w k_2 := \exists m, k_1 \sim_w k_2 \text{ (at } m)$$

From $\sim$ being a congruence relation we can derive

$$k_1 \sim_w k_2 \text{ (at } m) \to \forall n \geq m, k_1 \sim_w k_2 \text{ (at } n).$$

8.3.3 Excluded Middle for $\sim_w$

From decidability of $\sim$ decidability of $k_1 \sim_w k_2$ (at m) follows. Because of the unbounded existential quantifier $k_1 \sim_w k_2$ is not decidable. But we can show propositional decidability of $k_1 \sim_w k_2$ using (A1).

Let

$$b := \lambda n.\text{if } \mathit{decision}(k_1 \sim_w k_2 \text{ (at } m)) \text{ then } \mathsf{true} \text{ else } \mathsf{false}$$

be the sequence on $\mathbb{B}$, which is true at index n if $k_1 \sim_w k_2$ (at n). Now we make the following obnaservations:

- If $b(n) = \mathsf{true}$ then $\forall n \geq m, b(n) = \mathsf{true}$ which follows because $k_1 \sim_w k_2$ (at m) $\to \forall n \geq m, k_1 \sim_w k_2$ (at n).
- If false occurs infinitely often in b, then b is constant false. If there was any true in b, b would be constant true from this position by the first observation. But then false would not occur infinitely often.

Using (A1) we get that true or false occurs infinitely often in b. If false occurs infinitely often, by the second observation $k_1 \not\sim_w k_2$. If true occurs infinitely often, there is one index m such that $b(m) = \mathsf{true}$ and by definition of b $k_1 \sim_w k_2$. So

$$\forall k_1 k_2, k_1 \sim_w k_2 \vee k_1 \not\sim_w k_2$$

8.3.4 Finite Index of $\sim_w$

Because $\sim$ is of finite index, $\sim_w$ needs to be of finite index, too. Unfortunately, we were not able to give a type indexing the equivalence classes as we did for $\sim$. We are not able to give the equivalence class for a position in w, because w is arbitrary. But using the already proven excluded middle for $\sim_w$ we can prove

$$\forall k : \texttt{String}\ \mathbb{N}, |k| > |\mathbf{3}^{state(A) \times state(A)}| \to \exists i < j < |k|, k[i] \sim_w k[j]$$

which is exactly the designed requirement to apply (A2) later.

8.3.5 Existence of V and W

Once we have these more technical details on $\sim_w$, we now want to show that there are desired $V, W \in E_\sim$. The basic proof idea follows closely the argumentation in [1].

Using (A2) we can prove the existence of a strictly monotone sequence g such that

$$\begin{aligned} &g(0) > 0 \wedge \\ &\forall n, g(0) \sim_w g(n) \end{aligned}$$

, because we already proved that there are finitely many equivalence classes of $\sim_w$. So all indices in g are $\sim_w$ equivalent.

Now we will repeat finding subsequences of g which have more properties. Because we do this carefully, they inherit all properties.

Because $\sim$ is of finite index, we can apply (A1) on the sequence

$$f : (\mathbb{N} \to E_I) := \lambda n.\mathsf{class}_\sim(w(g(0), g(n)))$$

mapping n to the $\sim$ equivalence class of $w(g(0), g(n))$. (A1) gives one equivalence class, which occurs infinitely often in f. Using filter_∞ we filter g for these infinitely many positions and obtain the next subsequence g' such that

$$\begin{aligned} &g'(0) > 0 \wedge \\ &\forall n, g'(0) \sim_w g'(n) \wedge \\ &\forall m > 0, n > 0, w(g'(0), g'(n)) \sim w(g'(0), g'(m)) \end{aligned}$$

So all indices in g' are $\sim_w$ equivalent and all strings $w(g'(0), g'(m))$ are $\sim$ equivalent.

In the last step, we need to show the existence of a subsequence g'' of g' such that

$$\begin{aligned} &g''(0) > 0 \wedge \\ &\forall n, g''(0) \sim_w g''(n) \wedge \\ &\forall m > 0, n > 0, w(g''(0), g''(n)) \sim w(g''(0), g''(m)) \wedge \\ &\forall j \leq i, g''(j) \sim_w g''(i) \text{ (at } g''(i+1)) \end{aligned}$$

We construct g'' using $\mathsf{filter}_{\mathsf{history}}$ on g' using the predicate

$$\lambda g'' : \texttt{String}\ \mathbb{N}, g''[0] = g'(0) \wedge \forall j \leq i < |g''|, g''(j) \sim_w g''(i) \text{ (at } g''(i+1))$$

We need to assert that $g''[0] = g'(0)$ because otherwise we would lose the last property of g' (here it is crucial that $\mathsf{filter}_{\mathsf{history}}(w, P)(0) = 0$) . We need to prove:

1. $g'(0) \sim_w g'(0)$ (at $g'(1)$) and $g'(0) = g'(0)$ which is trivial.
2. If we have a strictly monotone string g'' such that $g''[0] = g'(0) \wedge \forall j \leq i < |g''|, g''(j) \sim_w g''(i)$ (at $g''(i+1)$) we can find a new index of $g' > g''[|g''| - 1]$ which preserves the property when appended to g''. We do this proof by induction on $|g''|$.
   - If $|g''| = 1$ (remember, that our strings are nonempty), taking $g''[0] + 1$ sufficies.
   - Assume that there is such an index m_k for the first k elements of g''. From the first property of g' we know that $g'(0) \sim_w g''[k+1]$, so there is an m such that $w(g'(0), g'(m_k)) \sim w(g''[k+1], g'(m))$. Because g' is strictly monotone, there is an smallest index m_{k+1} such that $g'(m_{k+1}) \geq \max(m_k, m)$.

So all indices in g are $\sim_w$ equivalent, all strings $w(g'(0), g'(m))$ are $\sim$ equivalent and two indices $i \leq j$ in g'' have the property $i \sim_w j$ (at $j+1$).

Finally we have to show that V is the equivalence class of $w(0, g''(0))$ and W the one of $w(g''(0), g''(1))$. Basically we can use g'' as the partition of w into strings. Then we only need to show that $w(g''(0), g''(1)) \sim w(g''(i), g''(i+1))$ for all i: By the last property of g'' we get

$$w(g''(0), g(i+1)) \sim w(g''(i), g''(i+1)).$$

By the third property we get

$$w(g''(0), g''(1)) \sim w(g''(0), g''(i+1)).$$

Then we get by transitivity that

$$w(g''(0), g''(1)) \sim w(g''(i), g''(i+1)).$$

So $w \in VW^\omega$.

8.4 Complement Construction

We can enumerate the finitely many equivalence classes of $\sim$ and thus we can enumerate all pairs of them. For each $V, W \in E_\sim$ we can decide whether $VW^\omega \cap L_B(A) = \emptyset$: Because we can construct NFAs recognizing V and W, we can construct an NFA recognizing the Büchi languages VW^ω and $VW^\omega \cap L_B(A)$. For the last one we can decide whether its language is empty or not. Then the complement of $L_B(A)$ is given as

$$L_B(A)^C = \bigcup_{V,W \in E_\sim \text{ s.t. } VW^\omega \cap L_B(A) = \emptyset} VW^\omega =: \mathsf{complement}_B(A).$$

We can construct the NFA $\mathsf{complement}_B(A)$ because we can construct the Büchi NFA for a union of finitely many Büchi NFAs. For the correctness, we establish the following four lemmata:

- $L_B(A) \cap L_B(\mathsf{complement}_B(A)) = \emptyset$
- $L_B(A) \cup L_B(\mathsf{complement}_B(A)) = X^\omega$
- $L_B(A)^C = L_B(\mathsf{complement}_B(A))$
- $L_B(A) = L_B(\mathsf{complement}_B(A))^C$

For the proofs the informative decision of language emptiness is important, because we need a sequence w to apply the compatibility theorem. As corollaries we obtain for arbitrary NFAs A, A_1 and A_2 the following decidability statements for Büchi languages:

- Language universality: $dec(L_B(A) = X^\omega)$
- Language inclusion: $dec(L_B(A_1) \subseteq L_B(A_2))$
- Language equality: $dec(L_B(A_1) = L_B(A_2))$

9 Formalization

Our formalization depends on the implementation of finite types given in [5]. This implementation contains [7], from which we used basic lemmata for decidability and lists directly. Additionally [5] contains the notion of pure predicates originally from [9]. Constructive choice for natural numbers is taken from [8].

Our Coq formalization consists about 5000 lines of code with roughly one fourth specifications and three fourths proofs. About 3000 lines deal with Büchi languages directly. Because we use functions $\mathbb{N} \to X$ to implement sequences over X, there are a lot of calculations on indices and they appear rather everywhere. These indices make many proofs more complicated than we wish. This is one reason why our formalization got so big. We would like to find a more elegant way for dealing with indices or to use a different representation of sequences without indices.

Defining strings as non empty prefixes of sequences turned out be really useful in general. But this comes at the price of loosing a useful equality on strings (which one would e.g. get when using lists). At the end this did not harm because we could prove everything using pointwise equality only.

The formalization is divided into the following important parts:

- *Seqs.v*: This module contains the definition of sequences and strings as basic operations on them (first part of section 2).
- *SeqOperations.v*: Contains the implementation of filter_∞, $\mathsf{filter}_{\mathsf{history}}$, and concat_∞ (last part of section 2).
- *StrictlyMonotoneSeqs.v*: Facts about strictly monotone sequences.

- *NFAs.v*: Definition of NFAs, string acceptance, facts about strings and sequences on NFAs, decidability of reachability in NFAs (section 4).
- *Buechi.v*: Necessary assumptions (section 3), Büchi Acceptance (section 4.1), closure constructions of union and intersection (section 5).
- *DecLanguageEmpty.v*: Decision of Büchi language emptiness as in section 6.
- *NFAConstructions.v*: NFA constructions given in section 7.
- *Complement.v*: Complementation Theorem (section 8).

Moreover, we have some utility code. This code shows some not so interesting facts (e.g. basic properties of the finite type $\{n \leq k | n \in \mathbb{N}\}$) or addresses technical difficulties (e.g. conversion between the sequence based string definition and lists to apply external theorems).

References

[1] Bakhadyr Khoussainov and Anil Nerode. *Automata Theory and its Applications*. Birkhäuser, Boston, 2011.

[2] Christian Doczkal and Gert Smolka. "Two-Way Automata in Coq". In: *Interactive theorem proving (itp 2016)*. Ed. by Jasmin Blanchette and Stephan Merz. Vol. 9807. LNCS. 2016, pp. 151–166.

[3] Dominique Perrin and Jean-Éric Pin. *Infinite Words - Automata, Semigroups, Logic and Games*. Elsevier, 2004.

[4] J. R. Büchi. "On a Decision Method in Restricted Second-Order Arithmetic". In: *International Congress on Logic, Methodology, and Philosophy of Science*. Stanford University Press, 1962, pp. 1–11.

[5] Jan Christian Menz. *A Coq Library for Finite Types*. Bachelor Thesis, Saarland University. 2016.

[6] Martin Hofmann and Martin Lange. *Automatentheorie und Logik*. Springer-Verlag Berlin Heidelberg, 2011.

[7] Gert Smolka. *Base Library for ICL*. Saarland University. 2016.

[8] Gert Smolka and Chad E. Brown. "Introduction to Computational Logic. Lecture Notes SS 2014". Saarland University. 2014.

[9] Gert Smolka and Kathrin Stark. "Hereditarily Finite Sets in Constructive Type Theory". In: *Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-27, 2016*. Ed. by Jasmin Christian Blanchette and Stephan Merz. Vol. 9807. LNCS. Springer, 2016, pp. 374–390.

[10] Ming-Hsien Tsai et al. “State of Büchi Complementation”. In: *Logical Methods in Computer Science* 10.4 (2014). DOI: 10.2168/LMCS-10(4:13)2014. URL: http://dx.doi.org/10.2168/LMCS-10(4:13)2014.

[11] Moshe Y. Vardi. “The Büchi Complementation Saga”. In: *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*. 2007, pp. 12–22. DOI: 10.1007/978-3-540-70918-3_2. URL: http://dx.doi.org/10.1007/978-3-540-70918-3_2.