# 6 Conclusion

We give a short overview of the theorems presented in this thesis and their corresponding statements in the Coq development. We then evaluate our choice of the SSReflect plugin. Finally, we discuss opportunities for future work.

## 6.1 Results

Theorem 4.2.1 and Theorem 4.2.5 show that deterministic and non-deterministic finite automata are equally expressive. For this, we construct two functions dfa_to_nfa and nfa_to_dfa to convert between the two characterizations.

**Lemma** dfa_to_nfa_correct (A: dfa): dfa_lang A =i nfa_lang (dfa_to_nfa A).

**Lemma** nfa_to_dfa_correct (A: nfa): nfa_lang A =i dfa_lang (nfa_to_dfa A).

We show in Theorem 4.6.1 that there is an equivalent DFA for every extended regular expression. For this, we construct a function re_to_dfa to compute an equivalent DFA from an extended regular expressions.

**Lemma** re_to_dfa_correct (r: regular_expression char) : dfa_lang (re_to_dfa r) =i r.

Building on that, we prove the decidability of equivalence of regular expressions in Theorem 4.7.1 with the help of a decision procedure for equivalence of finite automata. We give a function re_equiv to decide the equivalence of regular expressions.

**Lemma** re_equiv_correct (r s: regular_expression char): re_equiv r s <-> r =i s.

Theorem 4.8.2 shows that we can give an equivalent regular expression for every automaton. We construct a function dfa_to_re to compute the regular expression.

**Lemma** dfa_to_re_correct (A: dfa): dfa_lang A =i (dfa_to_re A).

Based on this and the results from previous chapters, we also show that extended and standard regular expressions are equally expressive and, thus, that extended regular expressions and regular languages are equally expressive. We give a function ext_re_to_std_re which constructs an equivalent standard regular expression for every extended regular expression.

**Lemma** ext_re_to_std_re_standard (r: regular_expression char): standard char (ext_re_to_std_re r).

**Lemma** ext_re_to_std_re_correct (r: regular_expression char): (ext_re_to_std_re r) =i r.

With Theorem 5.2.4, we prove that we can construct a Myhill relation for a language from a DFA for that language.

**Lemma** dfa_to_myhill (A: dfa): Myhill_Rel (dfa_lang A).

We prove in Theorem 5.3.1 that every Myhill relations is also a weak Nerode relation.

```
Lemma myhill_to_weak_nerode (L: language char): Myhill_Rel L -> Weak_Nerode_Rel L.
```

Theorem 5.5.12 shows that, if there is a weak Nerode relation, the Nerode relation is of finite index.

```
Lemma weak_nerode_to_nerode (L: language char): Weak_Nerode_Rel L -> Nerode_Rel L.
```

Finally, we prove in Theorem 5.4.1 that, if we are given a Nerode relation of finite index for a language, we can construct a DFA that accepts this language.

```
Lemma nerode_to_dfa_correct (L: language char) (f_N: Nerode_Rel L):
  L =i dfa_lang (nerode_to_dfa f_N).
```

## 6.2 SSReflect

We make extensive use of SSReflect's features in our development. The formalization of finite automata depends crucially on finite types (and, to a lesser extent, finite sets).

We also employ the `reflect` paradigm whenever possible. It offers a very convenient way of working with propositional and boolean predicates at the same time. The built-in support for changing from propositional to boolean statements lets us choose the most appropriate representation for the task at hand.

Furthermore, the very extensive library of general purpose lemmas in SSReflect enables us to focus on high-level proofs. The sole exception to this is the `allbutlast` predicate we need for Theorem 4.8.2. However, even in this case, we can mostly rely on the lemmas for `all` provided by SSReflect. All we need to do is provide a thin layer between the two predicates.

Additionally, the scripting language offered by SSReflect leads to very concise proof scripts. It succeeds in removing some of the bookkeeping overhead.

There are several disadvantages to SSReflect. One is that it not as widely used as Coq itself. This means that the group of people who can understand the proof scripts is small. However, in some cases, it might be sufficient to explain a small subset of SSReflect in order to give an understandable presentation of the formalized statements.

We also lose practical executability. Specifically, the implementation underlying finite types does not lend itself to computation. Since practical executability is not always a requirement, this restriction may not be relevant to some projects, as is the case with our development.

Based on these considerations, we believe that the use of SSReflect is very beneficial to formalizations that do not require executability, especially if there is algorithmic content.

## 6.3 Future Work

There a several possible extensions to our development. Additionally, there are some topics that are quite extensions but rather candidates for future formalizations.

### 6.3.1 $\varepsilon$-Transitions

We have avoided $\varepsilon$-transitions in our formalization. Non-deterministic finite automata with $\varepsilon$-transitions and regular expressions are equally expressive. They are, as we have shown, unnecessary in order to derive the results proven in this thesis. Nonetheless, it would be very interesting to add them to the list of formalized characterizations of regular languages.

### 6.3.2 Pumping Lemma

The Pumping Lemma [8] gives a sufficient condition for the non-regularity of a language. It is a well-known part of the theory of regular languages and, thus, a good candidate for an extension to our development.

### 6.3.3 Regular Grammars

Another characterization of regular languages is given by regular grammars. Regular grammars seem to enjoy less popularity than other characterizations. A formalization of formal grammars in general would also be a good starting point to formalize other parts of the Chomsky Hierarchy [15]. The context-free languages could be a good candidate for a formalization. We speculate that pushdown automata could be formalized similarly to how we formalized finite automata.

### 6.3.4 $\omega$-regular languages

A possible next step after the formalization of regular languages is a formalization of $\omega$-regular languages. There does not seem much literature on formalizing this topic. Such a development could include all commonly used acceptance criteria on $\omega$-automata. This would also make for a good opportunity to study Coq's co-inductive definitions.