

## 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 the equivalence of deterministic and non-deterministic finite automata.

**Lemma** `dfa_to_nfa_correct` : `dfa_lang A =i nfa_lang dfa_to_nfa` .

**Lemma** `nfa_to_dfa_correct` : `nfa_lang A =i dfa_lang nfa_to_dfa` .

We show in Theorem 4.6.1 that there is an equivalent DFA for every extended regular expression.

**Lemma** `re_to_dfa_correct` `r` : `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.

**Lemma** `re_equiv_correct` `r s` : `re_equiv r s <-> r =i s` .

Theorem 4.8.2 shows that we can give an equivalent regular expression for every automaton. Based on this and the results from previous chapters, we also show that extended and standard regular expressions are equivalent and, thus, that extended regular expressions are equivalent to regular languages.

**Lemma** `dfa_to_re_correct` : `dfa_lang A =i dfa_to_re` .

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

**Lemma** `ext_re_to_std_re_correct` `r` : `(ext_re_to_std_re r) =i r` .

With Theorem 5.2.4, we prove that we can construct a Myhill relation on a language that is accepted by a DFA.

**Lemma** `dfa_to_myhill` : `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` : `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` : `Nerode_Rel L` .

## 6 Conclusion

Finally, we prove in Theorem 5.4.1 that, if the Nerode relation on a language is of finite index, we can construct a DFA that accepts this language.

**Lemma** `nerode_to_dfa_correct`: `L =i dfa.lang nerode_to_dfa`.

### 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 extend, 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 executability is not always a requirement, this restriction may not be relevant to some projects, as is the case with our development.

Based on this 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 are several possible extensions to our development. Additionally, there are some topics that are not so much 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 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] describe 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 [14]. 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.