Constructive Formalization of Regular Languages

Jan-Oliver Kaiser
Advisors: Christian Doczkal, Gert Smolka
Supervisor: Gert Smolka

UdS

November 7, 2012
Table of Contents

1 Motivation

2 Recap
   Regular Expressions
   Finite Automata
   Finite Automata
   Regular Expression to Finite Automaton

3 Previous Talk
   Finite Automaton to Regular Expression

4 Myhill-Nerode Theorem
   Formalization of Equivalence Classes
   Myhill Relations and Nerode Relations
   Minimizing Equivalence Classes

5 Summary
Motivation

• **Goal:** Build an extensive, elegant, constructive formalization of regular languages that includes
  1. regular expressions,
  2. the decidability of equivalence of regular expressions,
  3. finite automata,
  4. and the Myhill-Nerode theorem.

• **How:** Coq with SSReflect. No sacrifices for performance.
Regular Expressions

Definition

- We use extended Regular Expressions (RE) over an alphabet Σ:

\[ r; s ::= \emptyset | \varepsilon | a | rs | r + s | r \& s | r^* | \neg r \]

\[
\begin{align*}
\mathcal{L}(\emptyset) &= \emptyset & \mathcal{L}(r^*) &= \mathcal{L}(r)^* \\
\mathcal{L}(\varepsilon) &= \{\varepsilon\} & \mathcal{L}(r + s) &= \mathcal{L}(r) \cup \mathcal{L}(s) \\
\mathcal{L}(.) &= \Sigma & \mathcal{L}(r \& s) &= \mathcal{L}(r) \cap \mathcal{L}(s) \\
\mathcal{L}(a) &= \{a\} & \mathcal{L}(rs) &= \mathcal{L}(r) \cdot \mathcal{L}(s) \\
\mathcal{L}(\neg r) &= \Sigma^* \setminus \mathcal{L}(r) 
\end{align*}
\]

- Implementation taken from Coquand and Siles\(^1\). This saved us quite a bit of work.

- \approx 200 lines of code including an implementation of regular languages and lots of useful lemmas.

---

\(^1\)Thierry Coquand and Vincent Siles. “A Decision Procedure for Regular Expression Equivalence in Type Theory”. In: \textit{CPP}. 2011, pp. 119–134.
Finite Automata

Definition

• Our Finite Automata (FA) over an alphabet $\Sigma$ consist of
  1. a set of states $Q$,
  2. a starting state $s \in Q$,
  3. a set of final states $F \subseteq Q$,
  4. a transition relation $\delta \in Q \times \Sigma \times Q$.

• Two types: one for non-deterministic FA ($\delta$ may be non-functional), one for deterministic FA ($\delta$ is functional).

• For our deterministic FA, $\delta$ is also total and, thus, a function. This helped, but maybe not a lot.
Finite Automata

Formalization

- Our definition is very close to the textbook definition.
- \( \approx 120 \) lines of code including some general lemmas for later proofs.

```ocaml
Record nfa : Type :=
{ nfa_state :> finType;
  nfa_s : nfa_state ;
  nfa_fin : pred nfa_state ;
  nfa_step : nfa_state -> char
            -> pred nfa_state }.

Record dfa : Type :=
{ dfa_state :> finType;
  dfa_s : dfa_state ;
  dfa_fin : pred dfa_state ;
  dfa_step : dfa_state -> char
            -> dfa_state }.
```
RE $\iff$ FA

- Structure of proof given by inductive definition of RE.
- **Plan:** Construct FA for every RE constructor.
- Sounds simple enough..
- .. $\approx 800$ lines of code.
- $\approx 100$ lines of that needed for equivalence of DFA and NFA.
- Another $\approx 100$ lines of code for **extended** regular expressions.
- This is a candidate for improvement.
• We use the “Transitive Closure method”, Kleene’s original proof\(^2\).

• This method recursively builds a regular expression \( R^X_{x,y} \) that recognizes words whose runs starting in \( x \) only pass through states in \( X \) and end in \( y \).

• The previous version constructed \( R^k_{x,y} \) which translates to \( R^\{z | \#(z) < k\}_x,y \) where \( \# \) is an ordering on \( Q \).

• Instead of \texttt{nat}, we now recurse on the size of a \textbf{finite subset} of \( Q \).\(^3\)

• This also avoids cumbersome conversions from \texttt{nat} to \texttt{SSReflect}’s ordinals and, finally, to states.

---


We introduce a decidable language $L^X_{x,y}$ that directly encodes the idea behind $R^X_{x,y}$ as a boolean predicate.

The proof of $FA \implies RE$ consists of three steps:

1. We show that $L^X_{x,y}$ respects the defining recursive equation of $R^X_{x,y}$.
2. We show that for all $w \in \Sigma^*$, $w \in L^X_{x,y} \iff w \in R^X_{x,y}$.
3. We show that $\bigcup_{f \in F} L^Q_{s,f} = \mathcal{L}(A)$ and thus $\sum_{f \in F} R^Q_{s,f} = \mathcal{L}(A)$. 
After some restructuring: \( \approx 550 \) lines of code, \( \approx 150 \) of which are general infrastructure.

Previous version: \( \approx 800 \) lines of code, much harder to read.

**Lemma** \( L\_\text{split} \): \( k' \ i \ j \ a \ w: \)

\[
\begin{align*}
& \text{let } k := k\_\text{ord} \ k' \ \text{in} \\
& (a \::\: w) \in L^{k'+1} i \ j \quad \rightarrow \\
& (a \::\: w) \in L^{k'} i \ j \quad \backslash
\end{align*}
\]

\[
\begin{align*}
& \exists w_1, \exists w_2, \\
& a \::\: w = w_1 ++ w_2 \quad \backslash
\end{align*}
\]

\[
\begin{align*}
& w_1 \neq [::] \quad \backslash
\end{align*}
\]

\[
\begin{align*}
& w_1 \in L^{k'} i (\text{enum\_val} \ k) \quad \backslash
\end{align*}
\]

\[
\begin{align*}
& w_2 \in L^{k'+1} (\text{enum\_val} \ k) \ j .
\end{align*}
\]

**Lemma** \( L\_\text{split} \): \( X \times y \ z \ w: \)

\[
\begin{align*}
& w \in L^{(z \mid: X) \times y} \quad \rightarrow \\
& w \in L^X \times y \quad \backslash
\end{align*}
\]

\[
\begin{align*}
& \exists w_1, \exists w_2, \\
& w = w_1 ++ w_2 \quad \backslash
\end{align*}
\]

\[
\begin{align*}
& \text{size } w_2 < \text{size } w \quad \backslash
\end{align*}
\]

\[
\begin{align*}
& w_1 \in L^X \times z \quad \backslash
\end{align*}
\]

\[
\begin{align*}
& w_2 \in L^{(z \mid: X) \ z \ y}
\end{align*}
\]

**Figure:** Previous and current version of the same lemma
Myhill-Nerode Theorem

• It turns out that there are two different concepts: Myhill relations and the Nerode relation.
• We also consider a related characterization: weak Nerode relations.
• All these are equivalence relations which we require to be of finite index, i.e. to have a finite number of equivalence classes.
• However, CoQ has no notion of quotient types.
Myhill-Nerode Theorem
Equivalence Relations of Finite Index

- We use functions of finite domain to represent equivalence relations of finite index.
- Think of the domain as the set of equivalence classes.
- We also need to have a representative of every equivalence class. Thus, we require surjectivity.

**Record** \text{Fin\_Eq\_Cls} :=
\begin{align*}
\{ & \text{fin\_type} : \text{finType}; \\
& \quad \text{fin\_func} : \rightarrow \text{word} \rightarrow \text{fin\_type}; \\
& \quad \text{fin\_surjective} : \text{surjective} \text{ fin\_func} \}
\end{align*}

- With constructive choice, we can then give a canonical representative of every equivalence class.

**Definition** \text{cr} (f : \text{Fin\_Eq\_Cls}) x := \text{xchoose} (\text{fin\_surjective} f x).
Myhill-Nerode Theorem

Myhill relations, weak Nerode relations, Nerode relation

- An equivalence relation \( \equiv \) of finite index is a **Myhill relation**\(^4\) for \( L \) iff

  \( \equiv \) is **right-congruent**, i.e. \( \forall u, v \in \Sigma^*. \forall a \in \Sigma. u \equiv v \implies ua \equiv va \),

  and \( \equiv \) **refines** \( L \), i.e. \( \forall u, v \in \Sigma^*. u \equiv v \implies (u \in L \iff v \in L) \).

- An equivalence relation \( \equiv \) of finite index is a **weak Nerode relation** for \( L \) iff

  \( \forall u, v \in \Sigma^*. u \equiv v \implies \forall w \in \Sigma^*. (uw \in L \iff vw \in L) \).

- The **Nerode relation**\(^5\) \( \equiv_L \) for \( L \) is defined such that

  \( \forall u, v \in \Sigma^*. u \equiv_L v \iff \forall w \in \Sigma^*. (uw \in L \iff vw \in L) \).

---


Myhill-Nerode Theorem
Formalization of Myhill, weak Nerode and Nerode relation

• For all three relations, we build a record that consists of an
equivalence relation of finite index and proofs of the properties of the
respective relation.

• Example:

Record Myhill_Rel :=
{ myhill_func :> Fin_Eq_Cls;
  myhill_congruent : right_congruent myhill_func;
  myhill_refines : refines myhill_func }. 
Myhill-Nerode Theorem

• Our version of the Myhill-Nerode theorem states that the following are equivalent
  1. language $L$ is accepted by a DFA,
  2. we can construct a Myhill relation for $L$,
  3. we can construct a weak Nerode relation for $L$,
  4. the Nerode relation for $L$ is of finite index.

• $(1) \implies (2)$ is easy. (Map word $w$ to the last state of its run on the automaton.)

• $(2) \implies (3)$ is a trivial inductive proof.

• $(4) \implies (1)$ is also straight-forward. (Use equivalence classes as states.)
Myhill-Nerode Theorem

(3) \implies (4)

- A weak Nerode relation (given as a function \( f \)) has at least as many equivalence classes as the Nerode relation.
- Some of them are redundant w.r.t. to \( L \), i.e. we may have equivalence classes s.t.

\[ \forall uv \in \Sigma^*. fu \neq fu \land \forall w \in \Sigma^*. (uw \in L \iff vw \in L). \]

- Our goal is to merge these equivalence classes.
- In fact, we construct an equivalence relation on these equivalence classes.
- Equivalence classes are contained in that relation iff they are equivalent w.r.t. to \( L \).
Myhill-Nerode Theorem
Finding equivalent equivalence classes

- We use the table-filling algorithm\(^6\) for minimizing DFA.
- Our fixed-point algorithm finds all equivalence classes whose representatives are distinguishable w.r.t. to L.
- The remaining equivalence classes are then equivalent w.r.t. to L.
- Start: \(\{(x, y) \mid cr(x) \notin L \iff cr(y) \in L\}\).
- Step: if the previous result is \(d\), the new result is \(d \cup \{(x, y) \mid \exists a. (f(cr(x)a), f(cr(y)a)) \in d\}\).
- Due to finiteness of the domain and monotonicity of the algorithm, it has a fixed point which we can compute.

Myhill-Nerode Theorem

Proof Outline

• We construct a function \( f_{\text{min}} \) that maps every equivalence class to the set of equivalence classes equivalent to it w.r.t. \( L \).
• We then show that \( f_{\text{min}} \) is surjective and encodes an equivalence relation of finite index.
• Finally, we show that \( f_{\text{min}} \) is equivalent to the Nerode relation.
Myhill-Nerode Theorem

- The lemmas of this chapter are rather abstract, which makes for nice and short statements.
- The proofs also received more refinement than the other chapters. They are very concise.
- The entire chapter consists of $\approx 430$ lines of code.
Summary

• All in all we have \( \approx 2100 \) lines of code.
• Mostly very close to the mathematical definitions.
• Code produced in the beginning of the project might be improved quite a bit.
Thank you for your attention!