Constructive Formalization of Regular Languages

Jan-Oliver Kaiser

Advisors: Christian Doczkal, Gert Smolka

Supervisor: Gert Smolka

Contents

Motivation

We want to develop an elegant formalization of regular languages in Coq based on finite automata.

There are several reasons for choosing this topic and our specific approach:

- Strong interest in formalizations in this area.
- Few formalizations of regular languages in Coq, most of them very long or incomplete.
- Most formalizations avoid finite automata in favor of regular expressions.
- It's fun.

Quick Recap

The regular languages over an alphabet Σ can be defined recursively:

- $\emptyset \in RL_\Sigma$
- $a \in \Sigma \rightarrow \{a\} \in RL_\Sigma$
- $A, B \in RL_\Sigma \rightarrow A \cup B \in RL_\Sigma$
- $A, B \in RL_\Sigma \rightarrow A \bullet B \in RL_\Sigma$
- $A \in RL_\Sigma \rightarrow A^* \in RL_\Sigma$

Additionally, regular languages are also exactly those languages accepted by **finite automata**.

One possible definition of FA over an alphabet Σ is:

- a finite set of states Q
- an initial state $s_0 \in$ Q
- a transition relation $\Delta \in$ (Q, Σ, Q)
- a set of finite states F, F $\sqsubseteq$ Q

Let A be a FA.

$\mathcal{L}(A) := \{ w \mid \exists s_1, \ldots s_{|w|} \in Q \ s.t. \ \forall \ i : 0 < i \leq n \rightarrow (s_{i-1}, w_i, s_i) \in \Delta \}$

They can also be defined using **regular expressions** which are usually converted to FA for matching:

- $\emptyset \in regexp_\Sigma,\ \mathcal{L}(\emptyset) := \{\}$
- $\varepsilon \in regexp_\Sigma,\ \mathcal{L}(\varepsilon) := \{\varepsilon\}$
- $a \in \Sigma \to a \in regexp_\Sigma,\ \mathcal{L}(a) := \{a\}$
- $r, s \in RL_\Sigma \to (r + s) \in regexp_\Sigma,\ \mathcal{L}(r + s) := \mathcal{L}(r) \cup \mathcal{L}(s)$
- $r, s \in RL_\Sigma \to (rs) \in regexp_\Sigma,\ \mathcal{L}(r \bullet s) := \mathcal{L}(r) \bullet \mathcal{L}(s)$
- $r \in RL_\Sigma \to r^* \in regexp_\Sigma,\ \mathcal{L}(r^*) := \mathcal{L}(r)^*$

Brzozowski showed that matching words against regular expressions can be done without converting them to FA. (1964, Derivatives of Regular Expressions)

- der a $\emptyset$= $\emptyset$
- der a ε= $\emptyset$
- der a b = if a = b then εelse $\emptyset$
- der a (r + s) = (der a r) + (der a s)
- der a (r s) = if $\delta(r)$ then (der a s) + ((der a r) s) else (der a r) s
- der a (r*) = (der a r) r*

with $\delta(r) = true \Leftrightarrow \varepsilon \in \mathcal{L}(r)$.

$w \in \mathcal{L}(r)$ if and only if the derivative of r with respect to $w_1 .. w_{|w|}$ accepts ε.

Finally, regular languages are also characterized by the Myhill-Nerode theorem.

- First, we define a binary relation on L:
  $R_L xy := \neg \exists z,\ x \bullet z \in L \oplus y \bullet z \in L$
- L is regular if and only if $R_L$divides L into a finite number of equivalence classes.

Previous work

- Constructively formalizing automata theory (2000)

  Robert L. Constable, Paul B. Jackson, Pavel Naumov, Juan C. Uribe

  PA: Nuprl

  The first constructive formalization of MH.

  Based on **FA**.

- Proof Pearl: Regular Expression Equivalence and Relation Algebra (2011)

  Alexander Krauss, Tobias Nipkow

  PA: Isabelle

  Based on **derivatives of regexps**. No proof of termination.

- Deciding Kleene Algebras in Coq (2011)

  Thomas Braibant, Damien Pous

  PA: Coq

  Based on **FA**, matrices. Focus on performance.

- A Decision Procedure for Regular Expression Equivalence in Type Theory (2011)

  Thierry Coquand, Vincent Siles

  PA: Coq

  Based on **derivatives of regexps**.

- A Formalisation of the Myhill-Nerode Theorem based on Regular Expressions (Proof Pearl) (2011)

  Chunhan Wu, Xingyuan Zhang, Christian Urban

  PA: Isabelle

  The first proof of MH based on **derivatives of regexps**.

- Deciding Regular Expressions (In-)Equivalence in Coq (2011)

  Nelma Moreira, David Pereira, Simão Melo de Sousa

  PA: Coq

  Based on Krauss, Nipkow. Proof of termination.

Our Development

- We want to focus on elegance, not performance.
- Our main goals are MH and the decidability of regexp equivalence.
- We use finite automata.

  They are not at all impractical. (Partly thanks to Ssreflect's finType)

Ssreflect

- Excellent support for all things boolean.
- Finite types with all necessary operations and closure properties.
  (very useful for alphabets, FA states, etc.)
- Lots and lots of useful lemmas and functions.

Finite automata

DFA and NFA without e-transitions.

- DFA to prove closure under $\cup$, $\cap$, and $\neg$.
- NFA to prove closure under $\bullet$ and $*$.

Also proven: NFA $\Leftrightarrow$ DFA.

This gives us: regexp $\Rightarrow$ FA.

Roadmap

1. Emptiness test on FA ($\emptyset(A) := \mathcal{L}(A) = \emptyset$)
2. FA $\Rightarrow$ regexp
3. Dedicedability of regexp equivalence using regexp $\Rightarrow$FA, (2) and (1):

   $\mathcal{L}(r) = \mathcal{L}(s)$

   $\Leftrightarrow$

   $\emptyset(\mathcal{A}(r) \cap \overline{\mathcal{A}(s)}) \wedge \emptyset(\overline{\mathcal{A}(r)} \cap \mathcal{A}(s))$

4. Finally, we want to prove the MH theorem

With this we'll have an extensive formalization of regular languages including regular expressions, FA and MH and all corresponding equivalences.