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. Regular expressions (with Brzozowski derivatives) lead to more complex but also more performant algorithms.

Quick Recap

We use extended **regular expressions** (regexp):

$$r,s \ ::= \ \emptyset \mid \varepsilon \mid a \mid rs \mid r+s \mid r\&s \mid r^* \mid \neg r$$

- $\mathcal{L}(\emptyset) := \{\}$
- $\mathcal{L}(\varepsilon) := \{\varepsilon\}$
- $\mathcal{L}(a) := \{a\}$
- $\mathcal{L}(rs) := \mathcal{L}(r) \cdot \mathcal{L}(s)$
- $\mathcal{L}(r+s) := \mathcal{L}(r) \cup \mathcal{L}(s)$
- $\mathcal{L}(r\&s) := \mathcal{L}(r) \cap \mathcal{L}(s)$
- $\mathcal{L}(r^*) := \mathcal{L}(r)^*$
- $\mathcal{L}(\neg r) := \overline{\mathcal{L}(r)}$

Derivatives of Regular Expressions (1964), *Janusz Brzozowski*:

- der a $\emptyset$= $\emptyset$
- der a ε= $\emptyset$
- der a b = if a = b then ε else $\emptyset$
- der a (r s) = if $\delta(r)$ then (der a s) + ((der a r) s) else (der a r) s
  with $\delta(r) = true \Leftrightarrow \varepsilon \in \mathcal{L}(r)$.
- der a (r + s) = (der a r) + (der a s)
- der a (r & s) = (der a r) & (der a s)
- der a (r*) = (der a r) r*
- der a ($\neg$r) = $\neg$(der a r)

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

Regular languages are also exactly those languages accepted by **finite automata** (FA).

Our definition of FA over an alphabet Σ :

- The finite set of states Q
- The initial state $s_0 \in$ Q
- The (decidable) transition relation $\Delta \in$ (Q, Σ, Q)

  Deterministic FA: Δ is functional and **total**.
- The set of finite states F, F $\sqsubseteq$ Q

Let A be a FA.

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

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

First, we define an equivalence relation on L:

$$x \; R_L \; y \; := \forall z, \; x \cdot z \; \in L \; \Leftrightarrow \; y \cdot z \; \in L$$

Myhill-Nerode theorem: 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)

Quick examples

```
Record dfa : Type :=
  dfaI {
    dfa_state :> finType;
    dfa_s0: dfa_state;
    dfa_fin: pred dfa_state;
    dfa_step: dfa_state -> char -> dfa_state
  }.

Fixpoint dfa_accept A (x: A) w :=
match w with
  | [::] => dfa_fin A x
  | a::w => dfa_accept A (dfa_step A x a) w
end.
```

```
Record nfa : Type :=
  nfaI {
    nfa_state :> finType;
    nfa_s0: nfa_state;
    nfa_fin: pred nfa_state;
    nfa_step: nfa_state -> char -> pred nfa_state
  }.

Fixpoint nfa_accept A (x: A) w :=
match w with
  | [::] => nfa_fin A x
  | a::w => existsb y, (nfa_step A x a y) && nfa_accept A y w
end.
```

50% of **NFA** $\Rightarrow$ **DFA** (powerset construction)

```
Lemma nfa_to_dfa_correct2 (X: nfa_to_dfa) w:
  dfa_accept nfa_to_dfa X w -> existsb x, (x \in X) && nfa_accept A x w.
Proof. elim: w X => [|a w IHw] X.
  by [].
move/IHw => /existsP [] y /andP [].
rewrite /dfa_step /nfa_to_dfa /=. rewrite cover_imset.
move/bigcupP => [] x H0 H1 H2.
apply/existsP. exists x. rewrite H0 andTb.
apply/existsP. exists y. move: H1. rewrite in_set => ->.
exact: H2.
Qed.
```

Roadmap

1. regexp $\Rightarrow$ FA: closure of FA under $\cdot$, $\cup$, $\cap$, $*$, $\neg$. (**Done**)
2. Emptiness test on FA ($\emptyset(A) := \mathcal{L}(A) = \emptyset$).
3. FA $\Rightarrow$ regexp.
4. Dedicedability of regexp equivalence.

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

5. Finally, we want to prove the Myhill-Nerode theorem.

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