Constructive Formalization of Regular Languages

Jan-Oliver Kaiser

September 5, 2012

Contents

Chapter 3

Decidable Languages

3.1 Definition

We closely follow the definitions from [**?**]. An **alphabet** Σ is a finite set of symbols. Throughout this document, we will assume a fixed alphabet. A **word** w is a finite sequence of symbols chosen from some alphabet. We use $|w|$ to denote the **length** of a word w. ε denotes the empty word. Given two words $w_1 = a_1a_2 \cdots a_n$ and $w_2 = b_1b_2 \cdots b_m$, the **concatenation** of w_1 and w_2 is defined as $a_1a_2 \cdots a_nb_1b_2 \cdots b_m$ and denoted $w_1 \cdot w_2$ or just w_1w_2. A **language** is a set of words. The **residual language** of a language L with respect to symbol a is the set of words u such that av is in L. It is denoted $res_a(L)$. We define Σ^k to be the **set of words of length k**. The **set of all words** over an alphabet Σ is denoted Σ^*, i.e.,$\Sigma^* = \bigcup_{k\in\mathbb{N}} \Sigma^k$.

We restrict all further discussion to **decidable languages**:

$$\mathcal{L}_{dec} := \{\, L \subseteq \Sigma^* \,|\, \exists f. \forall x \in \Sigma^*. f(x) = 1 \Leftrightarrow x \in L.\}$$

We employ finite types to formalize alphabets. Words are formalized as sequences over the alphabet. Decidable languages are represented by functions from *word* to *bool*.

```
Variable char: finType.
Definition word := seq char.
Definition language := pred word.
Definition residual x L : language := [preim cons x of L].
```

3.1.1 Operation on languages

The **concatenation** of two languages L_1 and L_2 is denoted $L_1 \cdot L_2$ and is defined as the set of words $w = w_1w_2$ such that w_1 is in L_1 and w_2 is in L_2. The **Kleene Star** (also called Kleene closure) of a language L is denoted L^*

and is defined as the set of words $w = w_1 w_2 \cdots w_k$ such that $w_1, w_2, \cdots w_k$ are in L. ε is contained in L^* ($k = 0$). We define the **complement** of a language L as $L \backslash \Sigma^*$, which we denote $\neg L$. Furthermore, we make use of the standard set operations **union** and **intersection**.

We take Coquand and Siles's [?] implementation of these operators. "plus" and "prod" refer to union and intersection, respectively.

```
Definition conc L1 L2 : language :=
  fun v => existsb i : 'I_(size v).+1, L1 (take i v) && L2 (drop i v).
Definition star L : language :=
  fix star v := if v is x :: v' then conc (residual x L) star v' else true.
Definition compl L : language := predC L.
Definition plus L1 L2 : language := [predU L1 & L2].
Definition prod L1 L2 : language := [predI L1 & L2].
```

Lemma 3.1.1. *Let $L_1, L_2, w = a_1 a_2 \cdots a_k$ be given. We have that*

$w \in L_1{\cdot}L_2$ *if and only if* $\exists n \in \mathbb{N}. 0 < n \leq k \wedge a_1 \cdots a_{n-1} \in L_1 \wedge a_n \cdots a_k \in L_2$.

Proof. "$\Rightarrow$" From $w \in L_1 \cdot L_2$ we have w_1, w_2 such that $w = w_1 w_2 \wedge w_1 \in L_1 \wedge w_2 \in L_2$. We chose $n := |w_1| + 1$. We then have that $a_1 \cdots a_{n-1} = a_1 \cdots a_{|w_1|} = w_1$ and $w_1 \in L_1$ by assumption. Similarly, $a_n \cdots a_k = a_{|w_1|+1} \cdots a_k = w_2$ and $w_2 \in L_2$ by assumption.
"$\Leftarrow$" We chose $w_1 := a_1 \cdots a_{n-1}$ and $w_2 := a_n \cdots a_k$. By assumption we have that $w = w_1 w_2$. We also have that $a_1 \cdots a_{n-1} \in L_1$ and $a_n \cdots a_k \in L_2$. It follows that $w_1 \in L_1$ and $w_2 \in L_2$. □

Listing 3.1: Formalization of lemma 1.1.1

```
Lemma concP : forall {L1 L2 v},
  reflect (exists2 v1, v1 \in L1 & exists2 v2, v2 \in L2 & v = v1 ++ v2)
          (v \in conc L1 L2).
```

Lemma 3.1.2. *Let $L, w = a_1 a_2 \cdots a_k$ be given. We have that*

$$w \in L^* \text{ if and only if } a_2 \cdots a_k \in res_{a_1}(L) \cdot L^* \vee w = \varepsilon.$$

Proof. "$\Rightarrow$" We do a case distinction on $|w| = 0$.

1. $|w| = 0$. It follows that $w = \varepsilon$.

2. $|W| \neq 0$, i.e. $|w| > 0$. From $w \in L^*$ we have $w = w_1 w_2 \cdots w_l$ such that $w_1, w_2 \cdots w_l$ are in L. There exists a minimal n such that $|w_n| > 0$ and for all $m < n$, $|w_m| = 0$. Let $w_n = b_1 b_2 \cdots b_p$. We have that $b_2 \cdots a_p \in res_{b_1}(L)$. Furthermore, we have that $w_{n+1} \cdots w_l \in L^*$. We also have $a_1 = b_1$ and $w = a_1 a_2 \cdots a_k = w_n \cdots w_l$. Therefore, we have $a_2 \cdots a_k \in res_{a_1}(L) \cdot L^*$.

"$\Leftarrow$" We do a case distinction on the disjunction.

1. $w = \varepsilon$. Then $w \in L^*$ by definition.

2. $a_2 \cdots a_k \in res_{a_1}(L) \cdot L^*$. By lemma 1.1.1 we have n such that $a_2 \cdots a_{n-1} \in res_{a_1}(L)$ and $a_n \cdots a_k \in L^*$. By definition of res, we have $a_1 \cdots a_{n-1} \in L$. Furthermore, we also have $a_n \cdots a_k = w_1 w_2 \cdots w_l$ such that $w_1, w_2 \cdots w_l$ are in L. We chose $w_0 := a_1 \cdots a_{n-1}$. It follows that $w = w_0 w_1 \cdots w_l$ with $w_0, w_1, \cdots w_l$ in L. Therefore, $w \in L^*$.

□

Listing 3.2: Formalization of lemma 1.1.2

```
Lemma starP : forall {L v},
  reflect (exists2 vv, all [predD L & eps] vv & v = flatten vv)
          (v \in star L).
```

Theorem 3.1.1. *The decidable languages are closed under concatenation, Kleene star, union, intersection and complement.*

Proof. We have already given algorithms for every operator. It remains to show that they are correct. For concatenation and the Kleene star, we have shown in lemma 1.1.1 and 1.1.2 that the formalization is equivalent to the formal definition. The remaining operators are applied directly to the decision functions. □

3.2 Regular Languages

Definition 3.2.1. *The set of regular languages REG is defined to be exactly those languages generated by the following inductive definition.*

- $\emptyset \in REG$,
- $\forall a \in \Sigma. \{a\} \in REG$,
- $\forall L_1, L_2 \in REG. L_1^* \in REG, L_1 \cup L_2 \in REG, L_1 \cdot L_2 \in REG.$

3.2.1 Regular Expressions

Regular expressions mirror the definition of regular languages very closely. We will consider **extended regular expressions** that include negation (Not), intersection (And) and . (Dot), which is a single-symbol wildcard. We take the implementation from Coquand and Siles's development ([**?**]), which is also based on SSReflect and comes with helpful infrastructure for our proofs.

Listing 3.3: Regular Expressions

```
Inductive regular_expression :=
 | Void
 | Eps
 | Dot
 | Atom of symbol
 | Star of regular_expression
 | Plus of regular_expression & regular_expression
 | And of regular_expression & regular_expression
 | Conc of regular_expression & regular_expression
 | Not of regular_expression .
```

Standard regular expressions, boolean predicate

We will later prove that this definition is equivalent to the inductive definition of regular languages in 1.2.1. In order to do that, we introduce a predicate on regular expressions that distinguishes **standard regular expressions** from **extended regular expressions** (as introduced above). Standard regular expression consist only of $Void$, Eps, $Atom$, $Star$ and $Plus$.

```
Fixpoint standard (e: regular_expression char) :=
  match e with
      | Not _ => false
      | And _ _ => false
      | Dot => false
      | _ => true
  end.
```

Connect standard reg-exp to reg. languages

3.2.2 Deciding Language Membership

We make use of **derivatives of regular expressions** ([?]) to decide if a word $w \in \Sigma^*$ is contained in the language $\mathcal{L}(r)$ of the regular expression r. Derivatives are themselves regular expressions and are computed with respect to a single input character. In order to define derivatives, we first define a related concept.

Definition 3.2.2. *The derivative* $der\, a\, r$ *of* r *w.r.t. to* a *is defined such that*

$$\forall w \in \Sigma^*.\, w \in \mathcal{L}(der\, a\, r) \Leftrightarrow w \in residual\, a\, \mathcal{L}(r).$$

A suitable implementation is provided by Coquand and Siles.

Listing 3.4: Derivatives of Regular Expressions

```
Fixpoint der x e :=
 match e with
```

```
| Void => Void
| Eps => Void
| Dot => Eps
| Atom y => if x == y then Eps else Void
| Star e1 => Conc (der x e1) (Star e1)
| Plus e1 e2 => Plus (der x e1) (der x e2)
| And e1 e2 => And (der x e1) (der x e2)
| Conc e1 e2 => if has_eps e1 then
   Plus (Conc (der x e1) e2) (der x e2)
   else Conc (der x e1) e2
| Not e1 => Not (der x e1)
end.
```

Theorem 3.2.1. *For all r, w and a, we have that $w \in der\, a\, r$ if and only if $w \in residual\, a$.*

Proof. We prove the claim by induction over r. Two cases are non-trivial: □ Proof

Given the defining property of derivatives, we can easily see that a generalization of *der* to words suffices to decide language membership. We only need to check if the derivative w.r.t. to a given word accepts the empty word.

```
Fixpoint mem_der e u := if u is x :: v then mem_der (der x e) v else has_eps e.
```

Theorem 3.2.2. *The language of a regular expression r is decidable, i.e.*

$$w \in \mathcal{L}(r) \Leftrightarrow \varepsilon \in \mathcal{L}(mem\_der\, r\, w).$$

Proof. . □ Proof

Chapter 2

Conclusion

Chapter 3

References