

Formal Languages & Regular Languages

Exercise 1

Let the alphabet A be binary digits. $A = \{0, 1\}$

For the exercise, we consider each word of A^* to represent a number in \mathbb{N} , in the usual way:

0	represents	0
1	represents	1
110	represents	6
1010	represents	10

Note that leading zeros are ignored, and that the empty string ε is assigned the value 0.

0011	represents	3
ε	represents	0

Question 1.1

Define a function $f: A^* \rightarrow \mathbb{N}$ that converts words over the alphabet A into numbers in \mathbb{N} according to the above specification.

Solution:

$$f(w) = \sum_{0 \leq n < |w|} w_n \cdot 2^{|w| - 1 - n}$$

Question 1.2

Let E be the language of even numbers: $E = \{w \in A^* \mid f(w) \text{ is even}\}$

Prove that $EE = E$. To do so, prove that all elements of E are also elements of EE , and that all elements of EE are also elements of E .

Solution:

First direction. Let w be an element of E . We observe that $w = \varepsilon w$. Since $\varepsilon \in E$, we have that $\varepsilon w \in EE$, and thus that $w \in EE$. Qed.

Second direction. For this direction, we observe that E is the set of all words which are either empty, or end with a 0.

Let w be an element of EE . We thus have that $w = uv$ for some u in E and v in E . We have two cases, either v is ε or v is a non-empty word ending with 0 (from our observation).

In case $v = \varepsilon$, then $w = uv = u\varepsilon = u$, and thus $w \in E$. In case v is non-empty and ends with a 0, then uv is also non-empty and also ends with a 0. Therefore $w \in E$. Qed.

Question 1.3

Prove that $E^* = E$. You may find the fact you have proven in question 1.2 to be useful here.

Solution:

We have, by definition, that $E^* = \bigcup_{n \geq 0} E^n$. Also, we have that $E^0 = \{\epsilon\}$ and $E^n = EE^{n-1}$ for $n > 0$. We remark that $E^n = E$ for all $n > 0$. We prove it by induction on n :

For $n = 1$, we have that $E^1 = EE^0 = E$.

For $n > 1$, we have that $E^n = EE^{n-1}$. By induction hypothesis, $E^{n-1} = E$. Thus, we have that $E^n = EE$. From question 1.2, we know that $EE = E$. Thus, $E^n = E$.

Therefore, we have that $\bigcup_{n \geq 0} E^n = E^0 \cup \bigcup_{n \geq 1} E^n = E^0 \cup \bigcup_{n \geq 1} E = E^0 \cup E = \{\epsilon\} \cup E = E$.

Therefore, $E^* = E$. Qed.

Question 1.4

Build a regular expression whose language is E .

Solution:

$(0|1)^*0 \mid \epsilon$

Note:

Note that concatenation binds stronger than disjunction, which means the above regular expression should be read as $((0|1)^*0) \mid \epsilon$ and not as $(0|1)^*(0 \mid \epsilon)$.

Also, when we want to match any letter of the alphabet A , we can use A itself as a regular expression. Indeed, as is the case with all finite sets, there always exists a regular expression for A , which is simply the disjunction of all letters of A . The solution could then be written as $A^*0 \mid \epsilon$.

Exercise 2

Let A be some alphabet and let $f: A^* \rightarrow \{\text{true}, \text{false}\}$ be a computable function from A^* to true or false. Let L be the language defined by f .

$$L = \{ w \in A^* \mid f(w) = \text{true} \}$$

Find an algorithm that, given a word over the alphabet A , decides whether the word is part of L^* , the Kleene closure of L . Your algorithm may of course invoke f , but only a number of times polynomial in the size of the input word.

Solution:

```
def inKleeneClosure[A](
  inLanguage: List[A] => Boolean,
  word: List[A]): Boolean = {

  // The array marked will contain true at some index i
  // if we have found a sequence of words in the language
  // that starts at index 0 and ends at that index (exclusive).
  val marked: Array[Boolean] = Array.fill(word.length + 1)(false)

  // The empty sequence of words starts at index 0
  // and ends at index 0, therefore we mark it.
  marked(0) = true

  // For every possible start position.
  for (start <- 0 until (word.length - 1)) {
    // If we have found a way to end a sequence
    // of words at that point.
    if (marked(start)) {
      // Then for all words starting at that point...
      for (end <- (start + 1) to word.length) {
        val subword = word.slice(start, end)
        // We check if the subarray is a word of our language.
        if (inLanguage(subword)) {
          // If so, we mark that we have found a sequence
          // that ends at the end index.
          marked(end) = true
        }
      }
    }
  }

  // We return whether we were able to mark the end index.
  marked(word.length)
}
```

Exercise 3

There are basic properties of formal languages that you will frequently encounter when working with such objects.

One such property is *nullability*. We say that a language $L \subseteq A^*$ is *nullable* if it contains the empty word ε . This property looks innocent enough, but the nullability of a language will often be relevant.

Another such property is the so-called *first* set of a language. We define the *first* set of a language $L \subseteq A^*$ to be the set of characters in A that appear at the start of a word in L .

$$\text{first}(L) := \{ w_0 \mid w \in L \}$$

Question 3.1

Give a recursive function to compute the nullability of a regular expression.

Solution:

```

nullable( $\varepsilon$ ) = true
nullable( $\emptyset$ ) = false
nullable(a) = false
nullable( $e_1 \mid e_2$ ) = nullable( $e_1$ ) || nullable( $e_2$ )
nullable( $e_1 e_2$ ) = nullable( $e_1$ ) && nullable( $e_2$ )
nullable( $e^*$ ) = true

```

Question 3.2

Give a recursive function to compute the *first* set for regular expressions.

Solution:

```

first( $\varepsilon$ ) = {}
first( $\emptyset$ ) = {}
first(a) = { a }
first( $e_1 \mid e_2$ ) = first( $e_1$ )  $\cup$  first( $e_2$ )
first( $e_1 e_2$ ) = if (nullable( $e_1$ )) first( $e_1$ )  $\cup$  first( $e_2$ ) else first( $e_1$ )
first( $e^*$ ) = first(e)

```

Exercise 4

Let A be some alphabet and L be a language over A . We define the derivative of a language L with respect to an element x of A to be the set of all words w such that xw is in L .

$$\frac{dL}{dx} = \{ w \in A^* \mid xw \in L \}$$

Question 4.1

Prove that whenever a language is regular, then its derivation is also regular. To do so, build a recursive function that return a regular expression for the derivation of L with respect to x given a regular expression for L .

Solution:

$$\frac{d\epsilon}{dx} = \emptyset$$

$$\frac{d\emptyset}{dx} = \emptyset$$

$$\frac{da}{dx} = \text{if } (a = x) \ \epsilon \ \text{else } \emptyset$$

$$\frac{d(e_1 | e_2)}{dx} = \frac{de_1}{dx} \mid \frac{de_2}{dx}$$

$$\frac{d(e_1 e_2)}{dx} = \text{if } (\text{nullable}(e_1)) \ \frac{de_1}{dx} e_2 \mid \frac{de_2}{dx} \ \text{else } \frac{de_1}{dx} e_2$$

$$\frac{d(e^*)}{dx} = \frac{de}{dx}$$

Question 4.2

Can you come up with an algorithm to decide whether a word is part of the language of a regular expression using regular expression derivation?

Solution:

Inputs: word w , regular expression e .

```
while ( $w \neq \epsilon$ ) {  
     $x := w_0$  // get first letter of  $w$   
     $w := w_{[1, |w|)}$  // remove first letter of  $w$   
     $e := \frac{de}{dx}$  // get derivative of  $e$  by  $x$   
}  
return nullable( $e$ )
```