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

Solution:

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

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 $E^0 = \{\varepsilon\}$ 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 $\bigcup_{n \geq 0} E^n = E^0 \cup \bigcup_{n \geq 1} E^n = E^0 \cup \bigcup_{n \geq 1} E = E^0 \cup E = \{\varepsilon\} \cup E = E$.
Therefore, $E^* = E$. Qed.

Question 1.4

Build a regular expression whose language is E.

Solution:

(0|1)*0 | ε

Note:

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

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 | ε.

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)
}
```