Homework 1
Languages

Version du September 20, 2020

This homework has to be returned tomorrow, on Tuesday, at the beginning of the tutorial.

Please note that it is a mandatory assignment, that should be written on a clean sheet of paper and handed at the beginning of the next tutorial. A student who fails to hand their paper will be considered to be absent from the class. Each day, you will have to hand a different assignment. It should take no more than two hours to read your notes and write your daily paper.

Exercise 1 (On operators)**.** *We consider an input alphabet* $\Sigma = \{a, b\}$*. Define the following languages using only a finite number of finite words and the language operators union* $\cup$*, concatenation* $\cdot$*, and Kleene's star* $\star$*. Other operators such as* $\cap$*,* $\overline{\ }$*,* Suff*,* Pref*, Fac are not allowed.*

As an example, $\text{Pref}(\{a\}\{b\}^\star)$ *can be rewritten as* $\{\varepsilon\} \cup \{a\} \cup \{a\}\{b\}^\star$*.*

$$
\begin{aligned}
L_1 &= \text{Suff}(\{a\}\{b\}^\star) \\
L_2 &= Fac(\{a\}\{b\}^\star) \\
L_3 &= (\{a\}\{b\}^\star\{a\}^\star) \cap (\{a\}^\star\{b\}^\star\{a\}) \\
L_4 &= \overline{\{a\}^\star} \\
L_5 &= \overline{\{a\}\{b\}^\star} \cap \{a\}^\star
\end{aligned}
$$

Exercise 2 (A prefix language)**.** *L is said to be a prefix language if* $\forall (u, v) \in L^2, u \neq v \implies u \notin \text{Pref}(v)$*, i.e. no word of L is prefix to another word of L.*

For two prefix languages L_1 *et* L_2*, prove that:*

1. $\varepsilon \in L_1 \implies L_1 = \{\varepsilon\}$*;*
2. $L_1 \cap L_2$ *is prefix;*
3. $L_1 L_2$ *is prefix;*
4. $L_1 \cup L_2$ *may not be prefix.*

Exercise 3 (A mathematical keypad)**.** *Consider a keypad with 11 keys: ten numerical keys "*`0`*"..."*`9`*", and a key "*`E`*" that confirms the input.*

You must program the keypad so that the door can only open if the two following conditions are met:

— *the "*`E`*" has just been pressed;*

— *the number (represented by a sequence of digits) typed before the key "*`E`*" (i.e. typed after the second to last "*`E`*", or since the keypad has been turned on if the key "*`E`*" hasn't been pressed yet) is a multiple of 7.*

The number typed can be arbitrarily long. As an example, "`123456789123456789E`*" is a multiple of 7, hence, a valid passcode. This is not true of "*`123456789E`*". As a consequence, there is an infinite number of correct passcodes.*

By the end of the week, you should be able to say to your guests something like "The infinite set of the passcodes accepted by this keypad is rational, it can therefore be recognized by a finite-state machine." (or the other way round; it's not like anyone at the dining table would listen to a raving lunatic anyway).

Obviously, some correct passcodes would be so long that they would not fit in the keypad's puny RAM. There should therefore exist an algorithm such that only a finite number of input digits are read at any given time.

The following constraints actually apply to our current case: one can only press a single key at a time, and the keypad can't know the total size of the whole input. The keypad uses a very small amount of RAM, it can't store the whole input stream then interpret it later as it reads the key "`E`*".*

The program should be able to analyse the input stream on the fly, on a digit-per-digit basis, and should use a very small amount of RAM. The latter constraint implies that recursive calls are forbidden.

Fill in the gaps in the following program sketch. It is written in C, but you can choose another language if you feel so as long as you keep the underlying algorithm the same.

```
// Global variables (fill in as needed).
...


for (;;) // infinite loop
{
  // get_key waits for a key press then returns a value between 0 and 9,
  // or -1 if the key E has been pressed
  int key = get_key();

  if (key == -1)
    {
       if (/* Fill in the blank */)
         {
           open_door();
         }
       // Keep in mind that the next sequence of digits will be matched to another number

       ...
    }
  else // 0 <= key <= 9.
    {
       // intepret the last input digit

       ...
    }
}
```