ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Networked Systems Group (NSG)

HS 2020

Prof. L. Vanbever / M. Apostolaki, T. Holterbach, R. Schmid
based on Prof. R. Wattenhofer's material

Discrete Event Systems

Solution to Exercise Sheet 5

1 Pumping length

We know that the minimum pumping length p of a language $L = L_1 \cup L_2$ is at most $p \leq max\{p_1, p_2\}$, where p_1 and p_2 are the minimum pumping length of L_1 and L_2, respectively. Here, let $L1 = 1^*0^+1^+0^*$ and $L2 = 111^+0^+$.

- The minimum pumping length of L_2 cannot be 4 because 1110 cannot be pumped. Now consider the string s that belongs to L_2 and that has a size of 5. If $s = 11110$, then it can be divided into xyz where $x = 111$, $y = 1$ and $z = 0$ and thus can be pumped. If $s = 11100$, then it can be divided into xyz where $x = 111$, $y = 0$ and $z = 0$ and thus can be pumped. Similarly, all longer words can be pumped. The minimum pumping length for $L2$ is thus 5.
- A string s of size 3 and belonging to L1 can always be pumped.

Considering the word 1110, observe that it can also not be pumped in $L = L_1 \cup L_2$. In conclusion, the minimum pumping length of L is 5.

2 The art of being regular

L is not regular. We show it using the pumping lemma. We start by choosing a string in L. Let $w = 100^p\#10^p$. Then $w \in L$ since x (100^p) is equal to 2y (where y is 10^p) for $p >= 0$. We must consider three cases for where y can fall:

a) y = 1 Pump out. Arithmetic is wrong. The left side is 0 but right side isn't.

b) y = 10* Pump out. Arithmetic is wrong.

c) y = 0^p Pump out. Arithmetic is wrong. Decreased left side but not right. So, in particular, it is no longer the case that $x \geq y$ (required since $y \not\emptyset$).

Bonus Questions:

- Show whether L is context-free.
  L is not context-free. To see this, observe that

$$\begin{aligned} L &= \{x\#y \mid x + y = 3y\} \\ &= \{x\#y \mid x = 2y\} \\ &= \{w0\#w \mid w \in 1(0 \cup 1)^*\} \end{aligned}$$

- Show whether $L' = \{x\#y \mid x + \text{reverse}(y) = 3 \cdot \text{reverse}(y)\}$ is context-free.
  The reverse()-function takes an integer as a bitstring and reverses the order of its bits.
  L' is context-free. To see this, observe that it contains all palindromes with the string "0#" pushed in the middle. L' is generated by the grammar: $S \to 1S1 \,|\, 0S0 \,|\, 0\#$.

3 Counter Automaton

a) A counter automaton is basically a finite automaton augmented by a counter. For every regular language $L \in \mathcal{L}_{reg}$, there is a finite automaton A which recognizes L. We can construct a counter automaton C for recognizing L by simply taking over the states and transitions of A and *not* using the counter at all. Clearly C accepts L. This holds for every regular language and therefore, $\mathcal{L}_{reg} \subseteq \mathcal{L}_{count}$.

b) Consider the language L of all strings over the alphabet $\Sigma = \{0, 1\}$ with an equal number of 0s and 1s. We can construct a counter automaton with a single state q that increments/decrements its counter whenever the input is a 0/1. If the value of the counter is equal to 0, it accepts the string. Hence, L is in $\mathcal{L}_{count}$. On the other hand, it can be proven (using the pumping lemma) that L is not in $\mathcal{L}_{reg}$ and it therefore follows $\mathcal{L}_{count} \nsubseteq \mathcal{L}_{reg}$.

Some languages where the (non-finite) frequency of one or several symbols depends on the frequency of other symbols can be recognized by counter automata. Such languages cannot be recognized by finite automata.

c) First, we show that a pushdown automaton can simulate a counter automaton. Hence, PDAs are at least as powerful as CAs! The simulation of a given CA works as follows. We construct a PDA which has exactly the same states as the CA. The transitions also remain between the same pairs of states, but instead of operating on an INC/DEC register, we have to use a stack. Concretely, we store the state of the counter on the stack by pushing '+' and '−' on the stack. For instance, a counter value '3' is represented by three '+' on the stack, and similarly a value '−5' by five '−'. Therefore, when the CA checks whether the counter equals 0, the PDA can check whether its stack is empty.

In the following, we give just one example of how the transitions have to be transformed. Assume a transition of the counter automaton which, on reading a symbol s, increments the counter—independently of the counter value. For the PDA, we can simulate this behavior with three transitions: On reading s and if the top element of the stack is '−', a minus is popped; if the top element is a '+', another '+' is pushed; and if the stack is empty, also a '+' is pushed.

Hence, we have shown that the PDA is at least as powerful as the CA, and it remains to investigate whether both CA and PDA are equivalent, or whether a PDA is stronger. Although it is known that the PDA is actually more powerful, the proof is difficult: There is no pumping lemma for CAs for example such that we can prove that a given context-free language cannot be accepted by a CA. However, of course, if you have tackled this issue, we are eager to know your solution... :-)