HS 2012

Prof. C. Stamm / K.-T. Foerster, T. Langner, J. Seidel

Discrete Event Systems

Solution to Exercise Sheet 5

1 Counter Automaton

a) A counter automaton is basically a finite automaton augmented by a counter. For every regular language $L \in L_{reg}$, 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, $L_{reg} \subseteq L_{count}$.

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 L_{count}. On the other hand, it can be proven (using the pumping lemma) that L is not in L_{reg} and it therefore follows $L_{count} \nsubseteq L_{reg}$.

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... :-)

2 Tandem Pumping

a) Use the tandem pumping lemma to show that the language is *not* context free. For example, consider the word $w = a^p b^{p+1} c^{p+2}$. Clearly, $w \in L$. The tandem pumping lemma requires that w can be written as $w = uvxyz$ with $|vy| \geq 1$ and $|vxy| \leq p$. For context free languages, it must hold that $uv^ixy^iz \in L$ for all $i \geq 0$.

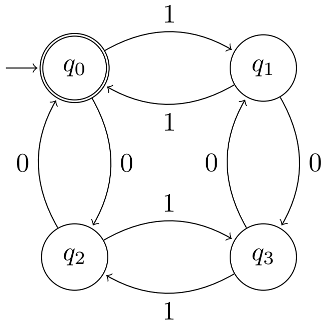
The window vxy can be applied at several locations on w. If it entirely covers the a region, then either v or y is at least one a. Therefore, pumping v and y increases the number of as in the resulting word, which violates the language definition.

If the window vxy starts in the area of the a's and ends in the area of b's, then v or y contains at least an a or a b. Again, pumping v and y increases the amount of this symbol, which results in a string not contained in the language. Similarly, if vxy only covers the b region, v or y contains at least one b, which produces strings not in L while pumping.

If the window vxy starts in the b area and ends in the c area, we have several cases: a) If either v or y contains both b and c, pumping w produces words not in L. If $v \in b^+$ and $y = \varepsilon$, pumping will produce words with too many b's. If $v \in b^+$ and $y \in c^+$, or if $v = \varepsilon$ and $y \in c^+$, we set i to 0 to obtain a string not in L.

If the window vxy entirely covers the c region, v or y contains at least one c. Thus, setting $i = 0$ removes at least one c, and the resulting string contains not enough cs to be in L.

b) This language is regular as the following corresponding DFA shows. Because the set of regular languages is a subset of the context-free languages, L is context-free.



3 Context Free Grammars

a) For reasons of brevity, we only give the productions of the grammar.

First, we create an equal number of symbols for w and z using rule (2), and then an equal number of symbols for x and y using rule (3).

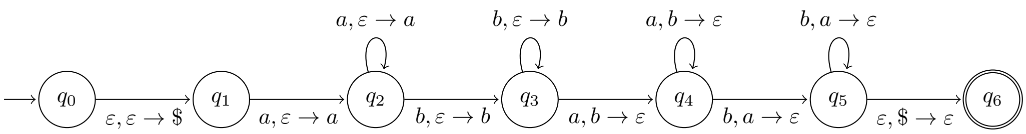
$$S \to A \tag{1}$$
$$A \to YAY \mid \#B\# \tag{2}$$
$$B \to YBY \mid \# \tag{3}$$
$$Y \to a \mid b \tag{4}$$

b) If $|w| = |y|$ and $|x| = |z|$, the resulting language is not context free, thus a CFG does not exist. This can be seen using the tandem pumping lemma as follows.

Let the word considered be $s = a^p\#a^p\#a^p\#a^p \in L$ with $|s| = 4p + 3 \geq p$. For any division $s = defgh$ with $|eg| \geq 1$ and $|efg| \leq p$, the pumpable regions e and g can never consist of boths as from w and y or both x and z because of the condition $|efg| \leq p$. Hence, any pumping would inevitably only modify the number of as in one part thereby creating a word $s' \notin L$. Therefore, L cannot be context free.

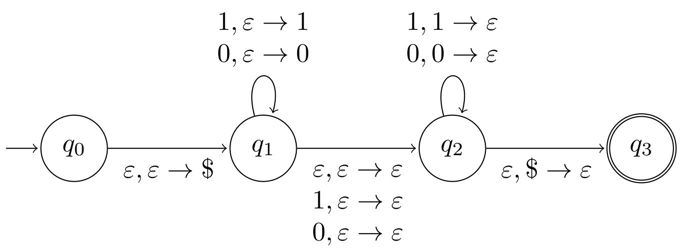
4 Push Down Automata

a) The PDA first reads all a from the input until it reads a b. For each a it reads, it pushes an a on the stack. Then, the PDA reads all b from the input until there comes an a. Again, for each b on the input, it pushes a b on the stack. Then, the automaton reads a from the input, but only if it can pop a b from the stack. Finally, it reads b from the input as long as it can pop an a from the stack.



b) This PDA should recognize all palindromes. However, we don't know where the middle of the word to recognize is. Therefore, we have to construct a non-deterministic automaton that decides itself when the middle has been reached.

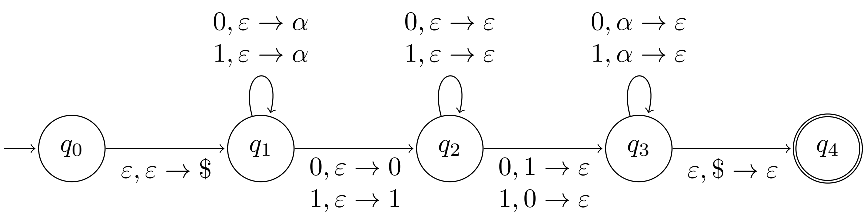
Note that we need to support words of even and odd length. Words of even length have a counterpart for each letter. However, the center letter of an odd word has no counterpart.



c) Consider the word w to be an array of symbols. If $w \in L$, there is at least one offset c, such that $w[c] \neq w[|w| - c]$. That is, there are two symbols x and y in w s.t. $x \neq y$ and the distance of x from the start of w equals the distance of y from the end of w.

The PDA reads $c - 1$ symbols, and stores a token α on the stack for each read symbol. Then, it reads the c-th symbol, and puts the symbol onto the stack. Afterwards, the PDA allows to read arbitrarily many symbols from the input, and does not modify the stack. Then, when only c symbols are left on the input stream, the PDA requires that the symbol on the stack must differ from the one on the input. Finally, the PDA reads the remaining $c - 1$ symbols and accepts if the stack is empty.

Note that this is again a non-deterministic PDA, as we do not know the value of c.



5 Designing Turing Machines

The proposed Turing machine decrements the value of a until $a = 0$. In each step, it adds a '1' to the output:

1 Move the TM head to the right of a and place a \$ sign. We will use this marker to return to the LSB of a.

2 Look at the LSB of a. If it is '1', we change it to 0 (transition between q_1 and q_3) and move to the right. Then, we continue moving to the right until we hit a $\square$, which is changed to a '1' (transition q_4 to q_5). Finally, we move back to the LSB of a.

3 If the LSB of a is '0', we search for the first '1' in a from the right (loop on q_1 and transition from q_1 to q_3).

3.1 If we find a '1', we change it to '0'. While moving back to the \$ symbol, we change all '0' to '1' (self-loop on q_3). Then, we proceed as in point 2 after passing the \$ symbol.

3.2 If we don't find a '1' in a at all (transition q_1 to q_6), we start the cleanup procedure: Remove all 0 on the right of the \$ symbol, and finally remove the \$ symbol itself and move to the right of u.