HS 2008 Prof. Dr. R. Wattenhofer / Raphael Eidenbenz / Roland Flury

Discrete Event Systems
Sample Solution to Exercise 5

1 Counter Automaton

- 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 recognized 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, $L_{reg} \subset L_{count}$.
- 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}.

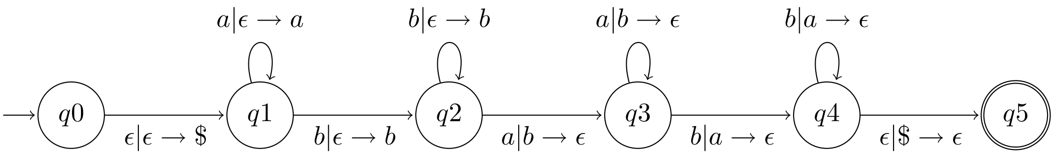
  On the other hand, it can be proven (using the pumping lemma) that L is not in L_{reg} and it therefore follows $L_{count} \notin L_{reg}$.
- First, we show that a pushdown automaton can simulate a counter automaton. Hence, PDA's are at least as powerful as CA's! The simulation of a given CA works as 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 a 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 CA's 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... :-)

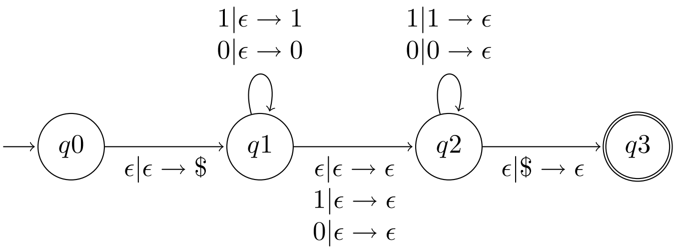
2 Push Down Automaton

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.

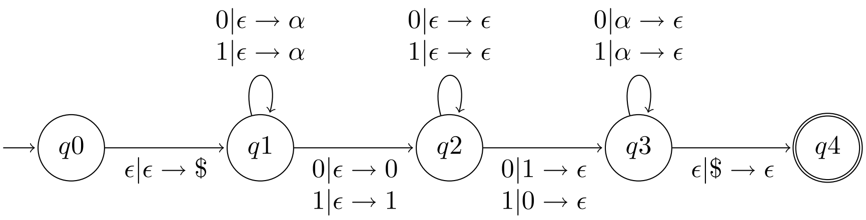
Note that we need to support words of even and odd length. Words of even length have a counter-part for each letter. However, the center letter of an odd word has no counterpart.



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 to 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.



3 Context Free Grammars

a) If x is not a permutation of y, then x and y contain a different number of a or b.

$$
\begin{array}{lcll}
S & \to & D & \textit{x and y differ in number of a} \\
 & \to & E & \textit{x and y differ in number of b} \\
D & \to & BaDaB \mid BaC\#B \mid B\#CaB & \\
E & \to & AbEbA \mid AbC\#A \mid A\#CbA & \\
B & \to & bB \mid \epsilon & \\
A & \to & aA \mid \epsilon & \\
C & \to & aC \mid bC \mid \epsilon &
\end{array}
$$

b) We can distinguish 2 cases: either $|x| \neq |y|$ or there is an offset i, such that $x[i] \neq y[i]$, thinking of x and y as arrays.

$$
\begin{array}{lcll}
S & \to & E & |x| \neq |y|
\end{array}
$$

$$
\begin{array}{rcll}
 & \rightarrow & AaC & |x| = |y| \textit{ and } \exists i : x[i] = b \textit{ and } y[i] = a \\
 & \rightarrow & BbC & |x| = |y| \textit{ and } \exists i : x[i] = a \textit{ and } y[i] = b \\
E & \rightarrow & DED & \\
 & \rightarrow & \#DC & \textit{right side is longer} \\
 & \rightarrow & DC\# & \textit{left side is longer} \\
D & \rightarrow & a \mid b & (a|b) \\
C & \rightarrow & DC \mid \epsilon & (a|b)^* \\
A & \rightarrow & DAD \mid bC\# & \\
B & \rightarrow & DBD \mid aC\# &
\end{array}
$$

Note that for the case $|x| = |y|$, we did not *enforce* that the two strings have equal length. But for the case they have equal length, they differ. (Thus, this grammar is ambiguous.)

4 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 $uv^i xy^i z \in L \forall i \geq 0$.

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 a 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 $y = \epsilon$, pumping will produce words with too many b's. If $v \in b^+$ and $y \in c^+$, or if $v = \epsilon$ and $y \in c^+$, we set i to 0 to obtain an string not in L.

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

b) This language is regular, see Figure 1. Because the set of regular languages is a subset of the context-free languages, the language is also context-free.

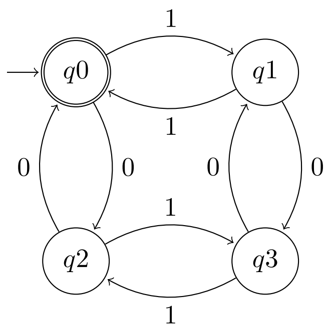
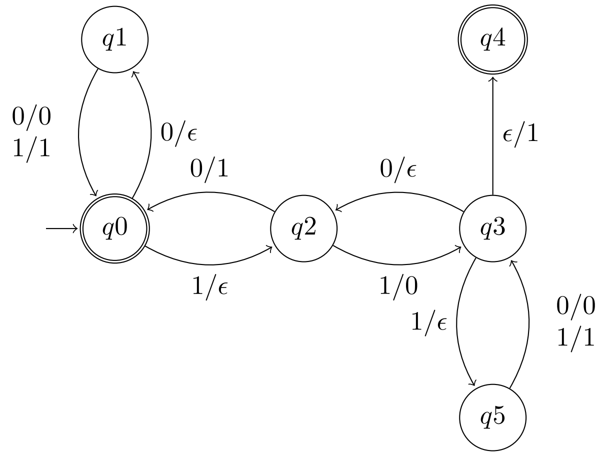


Figure 1: DFA for $L = \{x \mid x \in \{0,1\}^*$, and x contains an even number of '0' and an even number of '1'$\}$

c) Consider the word $w = 0^p 1^p \# 0^p 1^p \in L$. If the language is context free, we can apply the tandem pumping lemma. In order to keep the property that $|x| = |y|$, we must pump the same number of symbols on the left and right of #. Thus, the only reasonable place to place the sub-string vxy is such that v lies to the left of # and y to the right of #. But because $|vxy| \leq p$, v only contains 1 and y only contains 0. Therefore, for any string that we may pump (except for $i = 1$), the number of '0's x does not equal the number of '0's in y (and similarly for the number of '1's.) Therefore, the LHS and RHS of # are not permutations and the pumped strings are not in L. Thus, L is not context free.

5 Transducer and Turing Machine

a) The proposed automaton (which is deterministic!) reads two successive symbols (bits) of the input and outputs the sum. If there is a carry-over, we end up in state $q3$, where the output is adapted accordingly.



b) The machine performs the following actions:

1 Move the head to the LSB of b. For convenience of explanation, assume there is a variable i, initially set to 0. After this step, the TM head points to $b[i]$.

2 Replace the digit at the head with A or B, if the digit is a 0 or a 1, respectively. (That's how we store the value of digit $b[i]$ and can find back later on.)

3 Move to the left until we find the + sign. Then, continue moving left until we hit the first digit. (Note: this digit corresponds to $a[i]$). Depending on the value of this digit, go into state $q5$ or $q6$, and remove the digit $a[i]$, by writing a $\square$.

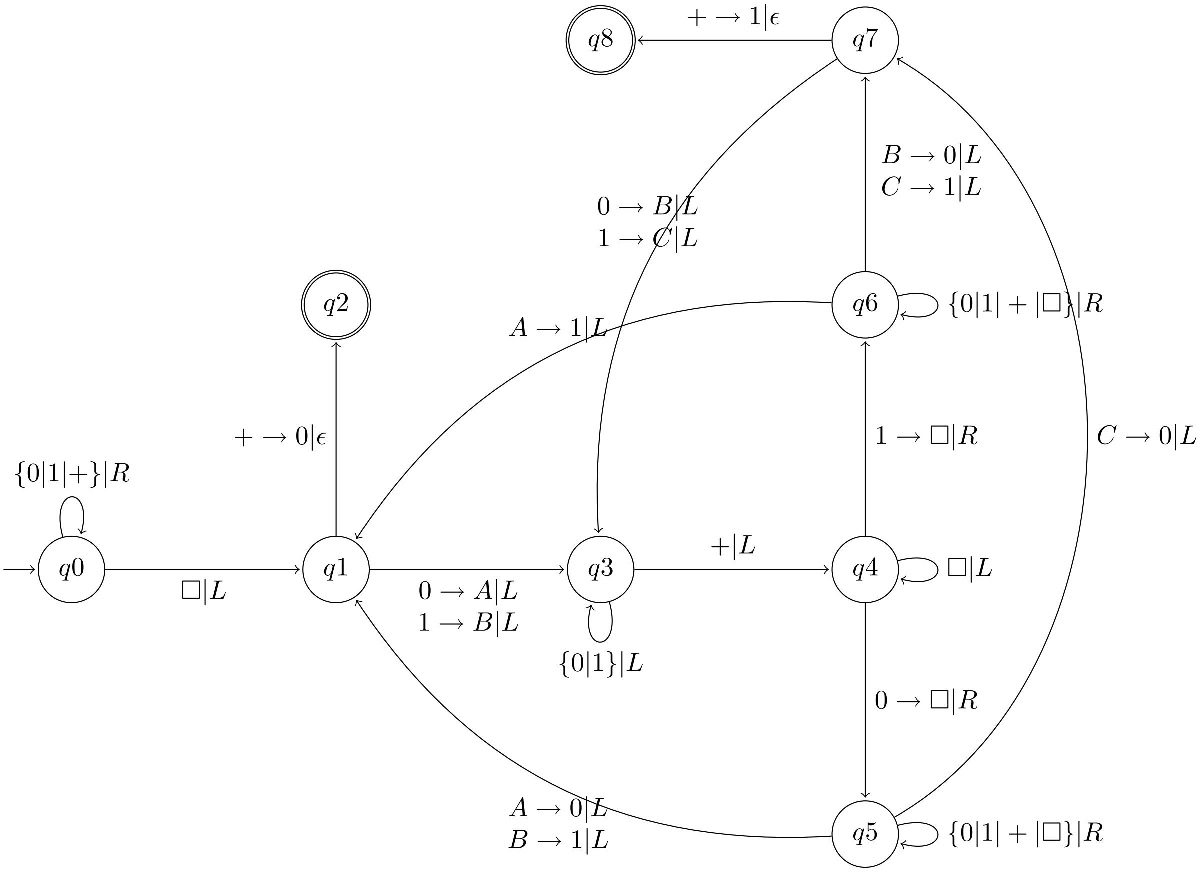
4 Move right until we hit an A or B (or C, which we explain later). At that point, we have the information of $a[i]$ and $b[i]$ and can determine the sum. If $a[i] + b[i] \geq 2$ (we get a reminder), go to state $q7$. (Note that $q1$ corresponds to $q7$: we're in $q7$ if there is a reminder, otherwise we're in $q1$.)

(5) Now, we're done with the digit at offset i. Increment i by one. (This is no action of the TM, it is only for the sake of explanation.)

6 Continue until we're in $q1$ or $q7$ and read a + sign, in which case we write the current reminder and terminate (accept).

6' Some more explanation to $q7$: In this state, we have a carry-over from the previous sum. Thus, $b[i]$ plus this carry over may already sum up to 2, in which case we write a C on the tape.

We use the following notation for transitions: $\alpha \to \beta|\gamma$: read α from the tape at the current position, then write a β and finally move left if $\gamma = L$ or move right if $\gamma = R$. We abbreviate transitions of the form $\alpha \to \alpha|\gamma$ and write $\alpha|\gamma$ (these transitions do not modify the content of the tape).



$+ \to 1|\epsilon$
$q8$
$q7$
$B \to 0|L$
$C \to 1|L$
$0 \to B|L$
$1 \to C|L$
$q2$
$q6$
$\{0|1|+|\Box\}|R$
$A \to 1|L$
$+ \to 0|\epsilon$
$1 \to \Box|R$
$C \to 0|L$
$\{0|1|+\}|R$
$q0$
$q1$
$q3$
$+|L$
$q4$
$\Box|L$
$\Box|L$
$0 \to A|L$
$1 \to B|L$
$\{0|1\}|L$
$0 \to \Box|R$
$A \to 0|L$
$B \to 1|L$
$q5$
$\{0|1|+|\Box\}|R$

c) 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 $q1$ and $q3$) and move to the right. Then, we continue moving to the right until we hit a □, which is changed to a '1' (transition $q4$ to $q5$). 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 $q1$ and transition from $q1$ to $q3$).

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 $q3$). 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 $q2$ to $q6$), 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.