Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
Philipp Bamberger, Yannic Maus

Theoretical Computer Science - Bridging Course
Summer Term 2018
Exercise Sheet 5

for getting feedback submit (electronically) before the start of the tutorial on 26th of November 2018.

Exercise 1: Constructing a Turing Machine *(3 Points)*

Consider alphabet $A = \{1, 2, \ldots, 9\}$. We call a string S over A a *blue* string, if and only if the string consisting of the odd-positioned symbols in S is the reverse of the string consisting of the even-positioned symbols in S. For example $S = 14233241$ is a blue string since the substring of the odd-positioned symbols is 1234 which is the reverse of the substring of the even-positioned symbols, i.e., 4321.
Design a Turing machine which accepts all blue strings over A. You do not need to provide a formal description of the Turing machine but your description has to be detailed enough to explain every possible step of a computation.

Sample Solution

On input S, first go through all symbols. If it is an odd number, reject. Else repeat the following: Go left until you reach the first unmarked symbol, mark it, go right to the last unmarked symbol, mark it, and compare both symbols. If they are different, reject.
Accept if all symbols are marked.

Exercise 2: *(4+2+2 Points)*

(a) Design a Turing Machine that decides the language $L := \{0^n 1^n \mid n \geq 1\}$. Explain your choice (you are supposed to explicitly construct the Turing machine).

(b) Give the sequence of configurations of your Turing machine run on the string 0011.

(c) Give the sequence of configurations of your Turing machine run on the string 0010.

Remark: Here, you need to solve part a) to solve part b) and c). We would try to avoid such exercises in the exam.

Sample Solution

(a) Alternately, the TM will change a 0 to an X and then a 1 to a Y until all 0s and 1s have been matched. In more detail, starting at the left end of the input, the TM enters a loop in which it changes a 0 to a X and moves to the right over whatever 0s and Ys it sees, until it comes to a 1. It changes the 1 to a Y and moves left, over Y 0s and 00s, until it finds an X. At that point, it looks for a 0 immediately to the right, and if it finds one, changes it to X and repeats this process, changing a matching 1 to a Y . The formal specification is $M = (Q; \Sigma; \Gamma; q_0; q_{reject}; q_{accept})$, where:

- $Q := \{q0, q_1, q_2, q_3, q_r, q_a\}$.
- $\Sigma = \{0, 1\}$.
- $\Gamma = \{0, 1, X, Y, \perp\}$

The transition function δ is given by

Q	0	1	X	Y	$\perp$
q_0	$(q_1; X; R)$	$(q_r; 1; R)$	$(q_r; X; R)$	$(q_3; Y; R)$	$(q_r; t; R)$
q_1	$(q_1; 0; R)$	$(q_2; Y; L)$	$(q_r; X; R)$	$(q_1; Y; R)$	$(q_r; t; R)$
q_2	$(q_2; 0; L)$	$(q_r; 1; R)$	$(q_0; X; R)$	$(q_2; Y; L)$	$(q_r; t; R)$
q_3	$(q_r; 0; R)$	$(q_r; 1; R)$	$(q_r; X; R)$	$(q_3; Y; R)$	$(q_a; t; R)$
q_r	-	-	-	-	-
q_a	-		-	-	-

Furthermore we have $q_{reject} := q_r$, $q_{accept} := q_a$.

(b)

$$\begin{aligned} q_0 0011 &\rightarrow Xq_1 011 \rightarrow X0q_1 11 \\ Xq20Y1 &\rightarrow q_2 X0Y1 \rightarrow Xq_0 0Y1 \\ XXq_1 Y1 &\rightarrow XXYq_1 1 \rightarrow XXq_2 YY \\ Xq_2 XYY &\rightarrow XXq_0 YY \rightarrow XXYq_3 Y \\ XXYYq_3 \perp &\rightarrow XXYYtq_a \perp \end{aligned}$$

(c)

$$\begin{aligned} q_0 0010 &\rightarrow Xq_1 010 \rightarrow X0q_1 10 \\ Xq_2 0Y0 &\rightarrow q_2 X0Y0 \rightarrow Xq_0 0Y0 \\ XXq_1 Y0 &\rightarrow XXYq10 \rightarrow XXY0q_1 t \\ & XXY0tq_r \perp \end{aligned}$$

Exercise 3: Random Questions *(2+2 Points)*

(a) Does the fact that the Halting Problem is not decidable mean that we can never tell if a program we have written is going to halt? Explain.

(b) Describe how a Turing machine with arbitrary tape alphabet Γ_0 can be simulated by a Turing machine with tape alphabet $\Gamma_1 = \{0, 1, \square\}$ that never writes the symbol $\square$ on the tape.

Sample Solution

(a) No, e.g., any Turing machine that simulates a DFA halts and if we write such a program and prove its correctness we know that it does.

(b) We use 0 and 1 to encode all symbols of Γ_0. Wlog let $a_1, \ldots, a_k$ be the symbols in Γ_0. A simple way to encode these symbols is to write $1^i = 1 \ldots 1$ for a_i and use the symbol 0 to separate different symbols.

Exercise 4: PDA to Turing Machine *(10 Points)*

Let a k-PDA be a pushdown automaton that has k stacks. Thus a 0-PDA is an NFA and a 1-PDA is a conventional PDA. We already know that 1-PDAs are more powerful (recognize a larger class of languages) than 0-PDAs.

(a) (5 points) Show that 2-PDAs are more powerful than 1-PDAs. *Hint: Find a suitable language that cannot be recognized by a 1-PDA but can be recognized by a 2-PDA*

(b) (5 points) Show that 3-PDAs are not more powerful than 2-PDAs. *Hint: Simulate a Turing machine tape with two stacks.*

Sample Solution

(a) It is easy to see that if a language can be recognized by a 1-PDA, it must be recognized by some 2-PDA. Now we show by giving an example that 2-PDA recognizes some language which 1-PDA can not. We know that the language $\{a^n b^n c^n \mid n \geq 1\}$ is not context-free, so that it cannot be recognized by a 1-PDA. Now we show that a 2-PDA recognizes this language. Suppose T_1 and T_2 be the two stacks in the 2-PDA. The 2-PDA will do the following: For each a it reads, push 'a' in both T_1 and T_2. For each b it reads, pop 'a' from T_1 and for each c it reads, pop 'a' from T_2. At any step, if it discovers the input string is not in correct order, i.e., not in the form–first all a, then all b, then c, the machine reject the input string. If the both the stacks T_1 and T_2 become empty at the end, we accept the input string. If not, reject it.

(b) If a 2-PDA can be used to simulate a Turing Machine, then it is clear that a 3-PDA is no more powerful than a 2-PDA, since a Turing machine can simulate a 3-PDA. This is because it is easy to simulate a 3-PDA by a 3-tape TM and we know that every multi-tape TM has an equivalent single tape TM. Therefore, we only show that a 2-PDA can simulate a Turing Machine.

Suppose T_1 and T_2 be the two stacks in the 2-PDA. Consider an arbitrary position of the Turing Machine tape-head. Then the first stack T_1 contains the tape symbols (including the state) to the left of the current head position, and the second stack T_2 contains tape symbols to the right. That is, assume at any time the Turing Machine configuration is $w_1 q a w_2$, where w_1 is the string to the left of the TM head, q is the current TM state, 'a' is the symbol under the head and w_2 is the string to the head's right. Then the TM configuration $w_1 q a w_2$ is represented by the 2-PDA configuration as $w_1 q$ in T_1 (with q at the top of the stack), and $a w_2$ in the reverse order in T_2 (i.e., with the symbol 'a' on top). Then the 2-PDA does the following:

1. For each step of the TM, the 2-PDA pops the top of T_1 (which is the TM state), and the top of T_2 (which is the symbol currently under the simulated machine's head).

2. This information is all we need to make a TM transition. The 2-PDA simulates $\delta(q, a) = (q', b, L)$ by (i) pushing b onto T_2 (ii) popping from T_1 and pushing the symbol obtained into T_2 (iii) pushing q' into T_1. It simulates $\delta(q, a) = (q', b, R)$ by pushing b and q', in that order, into the top of T_1. If at any time, the 2-PDA finds that T_2 is empty, it first pushes a blank symbol into T_2, and then performs in the same way.

Thus it can be seen that a 2-PDA can simulate a TM. Therefore the 3-PDA is no more powerful than 2-PDA.