ETH zürich
Departement Informatik

Theoretische Informatik

Prof. Dr. Juraj Hromkovič
Dr. Hans-Joachim Böckenhauer
`https://courses.ite.inf.ethz.ch/theo_inf_21`

Exemplary Solutions – Sheet 11

Zürich, December 17, 2021

Solution to Exercise 30

(a) Suppose that L_1 is context-free. Then we consider the number $n_1 = n_{L_1}$ from the pumping lemma for context-free languages and the word $z = 10^{n_1}\#1^{n_1}\#1^{n_1+1} \in L_1$. To avoid ambiguity, we refer to 10^{n_1} and 1^{n_1} as the first and second summand of the word, respectively, and refer to 1^{n_1+1} as the result. The pumping lemma yields a decomposition $z = uvwxy$ satisfying the three conditions of the pumping lemma. In particular, $|vx| \geq 1$ and $|vwx| \leq n_1$.

Now we distinguish three cases based on the relative position of v and x with respect to z.

If vx contains a symbol $\#$, then uwy does not have the proper form. Hence, in the following, we assume that vx consists only of zeros and ones.

If vx contains no part of the result, then one of the two summands in uwy (or both) becomes shorter, but the result z does not change. Because both summands have a leading 1, the condition on words in the language does not hold anymore which is a contradiction to the condition (iii).

If vx contains at least one symbol of the result, then vx cannot contain any symbol of the first summand due to the condition (ii). Hence, the binary length of the first summand in uwy is unchanged, but the binary length of the result is strictly shorter compared to z. Because both summands have a leading 1, the condition on words in the language does not hold anymore, independently of whether vx contains part of the second summand.

No more cases can occur and thus we get a contradiction to the pumping lemma. Hence, our assumption was false and L_1 is not context-free.

(b) Suppose that L_2 is context-free. Then we consider the number $n_2 = n_{L_2}$ from the pumping lemma for context-free languages and the word $z = a^{n_2}b^{n_2}ca^{n_2}b^{n_2} \in L_2$. The pumping lemma yields a decomposition $z = uvwxy$ satisfying the three conditions of the pumping lemma. In particular, $|vx| \geq 1$ and $|vwx| \leq n_2$.

Now we distinguish several cases based on the relative position of v and x with respect to z.

If vx contains the symbol c, then uwy contains no more c and thus $uwy \notin L_2$ which is a contradiction to the condition (iii) from the pumping lemma.

If vx only contains symbols from the prefix $a^{n_2}b^{n_2}$ of z, then the subword before c in uv^2wx^2y is longer than the subword after c. Hence, $uv^2wx^2y \notin L_2$ which is a contradiction to the condition (iii) from the pumping lemma.

If vx only contains symbols from the suffix $a^{n_2}b^{n_2}$, then the subword before c in uwy is longer than the subword after c. Hence, $uwy \notin L_2$ which is a contradiction to the condition (iii) from the pumping lemma.

The only remaining case is that vx contains symbols before c as well as symbols after c. Because of $|vwx| \leq n_2$, vx can contain no symbol a from the prefix a^{n_2} and no symbol b from the suffix b^{n_2}. Then uv^2wx^2y contains more symbols b before c than after c. Hence, $uv^2wx^2y \notin L_2$ which is a contradiction to the condition (iii) from the pumping lemma.

We get a contradiction to the pumping lemma in every case. Hence, our assumption was false and L_2 is not context-free.

Solution to Exercise 31

As L_1 is context-free and L_2 is regular, L_1 is accepted by a pushdown automaton $M_1 = (Q_1, \{a, b\}, \Gamma, \delta_1, q_{0,1}, Z_0)$ and L_2 is accepted by a (nondeterministic) finite automaton $M_2 = (Q_2, \{a, b\}, \delta_2, q_{0,2}, F)$.

Now we can essentially run both automata in parallel, using the product construction just like when combining two finite automata. The only additional difficulty is that M_1 accepts once the stack becomes empty and M_2 accepts upon reaching an accepting state. The new automaton being a pushdown automaton has to accept by emptying the stack.

To this end, we only allow to empty the stack if M_2 simultaneously makes a transition to an accepting state. To be able to recognize that the stack becomes empty, we have to guarantee that Z_0 is only used once at the very bottom of the stack. To this end, we first transform M_1 into an equivalent pushdown automaton $M_1' = (Q_1', \{a, b\}, \Gamma', \delta_1', q_{0,1}', Z_0')$ where

$$\begin{aligned}
Q_1' &= Q_1 \cup \{q_{0,1}'\}, q_{0,1}' \notin Q_1,\\
\Gamma' &= \Gamma \cup \{Z_0'\}, Z_0' \notin \Gamma,\\
\delta_1'(q_{0,1}', \lambda, Z_0') &= \{(q_{0,1}, Z_0'Z_0)\},\\
\delta_1'(q, \lambda, Z_0') &= \{(q, \lambda)\} \text{ for all } q \in Q_1,\\
\delta_1'(q, x, Z) &= \delta_1(q, x, Z) \text{ for all } q \in Q_1, x \in \{a, b\} \cup \{\lambda\}, \text{ and } Z \in \Gamma.
\end{aligned}$$

The pushdown automaton M_1' uses Z_0' as the symbol at the very bottom of the stack. In the only possible initial step, it puts Z_0 on top of it and changes its state to $q_{0,1}$, i.e., it reproduces the initial configuration of M_1 (with an additional symbol Z_0' at the very bottom of the stack). Then it behaves exactly like M_1 until Z_0' is read again. In the case when Z_0' is read (i.e., it is at the top of the stack), it is removed from the stack yielding an empty stack and finishing the computation. This happens if and only if everything else has been removed from the stack, i.e., the stack of M_1 has been emptied. Hence, the two automata are equivalent.

Using the product construction on M_1' and M_2, we construct the following pushdown automaton $M = (Q, \{a, b\}, \Gamma', \delta, q_0, Z_0')$ where

$$Q = Q_1' \times Q_2,$$

$$\begin{aligned} q_0 &= (q'_{0,1}, q_{0,2}), \\ \delta(q_0, \lambda, Z'_0) &= \{((q'_{0,1}, q_{0,2}), Z'_0 Z_0)\}, \\ \delta((p,q), \lambda, Z'_0) &= \{((p,q), \lambda, \lambda)\} \text{ for all } p \in Q_1, q \in F, \end{aligned}$$

and, for all $(p,q) \in Q$, $x \in \{a,b\}$, and $Z \in \Gamma$, we set

$$\begin{aligned} \delta((p,q), x, Z) &= \{((p',q'), \beta) \mid (p',\beta) \in \delta'_1(p,x,Z), q' \in \delta_2(q,x)\} \text{ as well as} \\ \delta((p,q), \lambda, Z) &= \{((p',q), \beta) \mid (p',\beta) \in \delta'_1(p,\lambda,Z)\}. \end{aligned}$$

The construction implies, for every word $w \in \{a,b\}^*$, that an accepting computation

$$((q'_{0,1}, q_{0,2}), w, Z'_0) \vdash_M \cdots \vdash_M ((p,q), u, \alpha) \vdash_M \cdots \vdash_M ((p',q'), \lambda, \lambda)$$

of M on w corresponds to the accepting computations

$$(q'_{0,1}, w, Z'_0) \vdash_{M'_1} \cdots \vdash_{M'_1} (p, u, \alpha) \vdash_{M'_1} \cdots \vdash_{M'_1} (p', \lambda, \lambda)$$

and

$$(q_{0,2}, w) \vdash_{M_2} \cdots \vdash_{M_2} (q, u) \vdash_{M_2} \cdots \vdash_{M_2} (q', \lambda)$$

of M'_1 and M_2 on w, respectively, and vice-versa. One only has to make sure that the steps of M'_1 and M reading an empty word are such that the state of M_2 is left unchanged.

Overall, we obtain

$$L(M) = L(M'_1) \cap L(M_2) = L_1 \cap L_2,$$

thus $L_1 \cap L_2$ is context-free.

Solution to Exercise 32

In the following, we show how a Turing machine can be simulated by a two-stack pushdown automaton. The simulation is based on the following idea: the read/write head of the Turing machine splits its working tape into two parts that are simulated by the two stacks. The top of one of the stacks corresponds to the read/write head of the Turing machine and a step to the left or to the right corresponds to copying from one stack to the other one.

Let M be a Turing machine. Then we simulate M by a two-stack pushdown automaton A as follows: Because the Turing machine does not have a separate input tape, we first have to copy the entire input of A onto the stacks. We perform this copying by first writing each input symbol on the first stack. Then we push a blank symbol $\sqcup$ onto the second stack. To read the first input symbol at the beginning of the actual simulation of M, we move the content of the first stack (except for the initial stack symbol Z_0) one symbol after another onto the second stack. The corresponding transitions (and also all subsequent ones) are λ-transitions, i.e., A ignores its own input from now on. Finally, we push the endmarker ¢ of M's tape onto the first stack.

Now the actual simulation can begin. We always interpret the symbol at the top of the first stack as the cell of M's tape that is currently scanned by the read/write head. The rest of the first stack contains the part of M's tape to the left of the read/write head. The second stack contains the part of M's tape to the right of the read/write head. Replacing the symbol at the current position on M's tape can be easily simulated. A step to the left on M's tape corresponds to moving the symbol at the top of the first stack onto the second stack, a step to the right on M's tape corresponds to moving the symbol at the top of the second stack onto the first stack. If $\sqcup$ is encountered on the second stack, then this blank symbol is not moved onto the first stack, but copied onto the first stack.

Because both stacks contain an initial stack symbol that is not changed during the simulation, the simulation cannot abort due to an empty stack. If the simulated TM M reaches the accepting state q_{accept}, then A should accept its input, thus A erases both stacks and halts. If the simulation of M reaches the rejecting state q_{reject}, then A halts without emptying the stacks and thus A does not accept its input. If the computation of M never ends, then the simulation does not end either and thus A does not accept its input.