Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

WS 2005/06

Prof. R. Wattenhofer / Thomas Moscibroda / Stefan Schmid

Discrete Event Systems
Exercise 6: Sample Solution

1 Chomsky Normal Form

There are two ways of solving this problem. One way is to systematically follow the procedure on page 99 of the textbook. The other one is to design a chomsky normal form from scratch that generates the same language. In the following, we will present both solutions. For the systematic approach, we begin by introducing a new start symbol S_0 that leads to the former start symbol A.

$$\begin{aligned} S_0 &\rightarrow A \\ A &\rightarrow BAB \mid B \mid \epsilon \\ B &\rightarrow 00 \mid \epsilon \end{aligned}$$

Next, we replace all ϵ-rules except for the new start state. As a result, we obtain the following context free grammar:

$$\begin{aligned} S_0 &\rightarrow A \mid \epsilon \\ A &\rightarrow BAB \mid B \mid AB \mid BA \mid A \mid BB \\ B &\rightarrow 00 \end{aligned}$$

In the next step, we remove all *unit rules* of the form $A \rightarrow A$, $S \rightarrow A$, $A \rightarrow B$, and so forth. This procedure leads to the following CFG:

$$\begin{aligned} S_0 &\rightarrow BAB \mid 00 \mid AB \mid BA \mid BB \mid \epsilon \\ A &\rightarrow BAB \mid 00 \mid AB \mid BA \mid BB \\ B &\rightarrow 00 \end{aligned}$$

Next, we have to replace all rules having three or more variables on the right side. In our case, we have to get rid of the rules $S_0 \rightarrow BAB$ and $A \rightarrow BAB$.

$$\begin{aligned} S_0 &\rightarrow BA_1 \mid 00 \mid AB \mid BA \mid BB \mid \epsilon \\ A &\rightarrow BA_1 \mid 00 \mid AB \mid BA \mid BB \\ A_1 &\rightarrow AB \\ B &\rightarrow 00 \end{aligned}$$

Finally, we introduce a variable C that maps to a single terminal symbol 0 as follows.

$$\begin{aligned} S_0 &\rightarrow BA_1 \mid CC \mid AB \mid BA \mid BB \mid \epsilon \\ A &\rightarrow BA_1 \mid CC \mid AB \mid BA \mid BB \\ A_1 &\rightarrow AB \\ B &\rightarrow CC \\ C &\rightarrow 0. \end{aligned}$$

The grammar given in the exercise produces all strings with an even number (possibly zero) of 0s. Therefore, another way of solving the exercise is to design a context free grammar in chomsky normal form generating the same language. For instance, the following (much simpler) grammar does the job.

$$\begin{aligned} S_0 &\rightarrow AA \mid BB \mid \epsilon \\ B &\rightarrow AA \mid BB \\ A &\rightarrow 0. \end{aligned}$$

2 CFL Closure Properties

Given two CFLs L and L', $L, L' \subseteq A^*$, consider any grammars G and G' that generate L and L', respectively. The idea is to combine G and G' appropriately in order to obtain grammars for $L \cup L'$, LL', and L^*. Consider the union of L and L', $L \cup L'$, and let $G = (V, A, P, S)$ and $G' = (V', A', P', S')$. The following grammar generates $L \cup L'$:

$$G(L \cup L') = (V \cup V' \cup \{S_0\}, A \cup A', P \cup P' \cup \{S_0 \rightarrow S|S'\}, S_0). \tag{1}$$

The concatenation can be proven as follows:

$$G(LL') = (V \cup V' \cup \{S_0\}, A \cup A', P \cup P' \cup \{S_0 \rightarrow SS'\}, S_0). \tag{2}$$

Finally, for the Kleene star, let $G = (V, A, P, S)$, we obtain $G(L^*)$ as follows:

$$G(L^*) = (V \cup \{S_0\}, A, P \cup \{S_0 \rightarrow S_0S_0|S, S_0 \rightarrow \epsilon\}, S_0). \tag{3}$$

3 Context Sensitive Languages

a) Assume that L is context free and consider the word $0^p1^p0^p1^p \in L$, where p is the pumping number. We can write $0^p1^p0^p1^p = uvxyz$ with a non-empty pumpable area $|vx| \geq 1$ and $|vwx| \leq p$. This implies that vwx is in the first 0^p1^p (case A), or in the middle 1^p0^p (case B), or in the last 0^p1^p (case C).

We know that after tandem-pumping it holds that $uv^ixy^iz \in L$. Choose $i = 0$, which in case A yields the word $0^k1^l0^p1^p$, in case B the word $0^p1^k0^l1^p$, and in case C the word $0^p1^p0^k1^l$ for some $k < p$ or some $l < p$, because at least one character has been removed. None of these words is in L, which yields the contradiction.

b) The following context-sensitive grammar describes L.

The idea is the following: We use the non-terminal U to mark the right end of the word. Furthermore, for every bit in the first part of the word zz we produce a corresponding non-terminal for the second part, i.e., A represents 0, B represents 1, C represents 00, D represents 01, E represents 10, and F represents 11.

$$S \rightarrow TU|00|11|\epsilon \qquad T \rightarrow 0TA|1TB|00C|01D|10E|11F$$

Note that non-terminals storing the information about the first word are still in the wrong ("palindrom") order. Hence, we need the following additional rules:

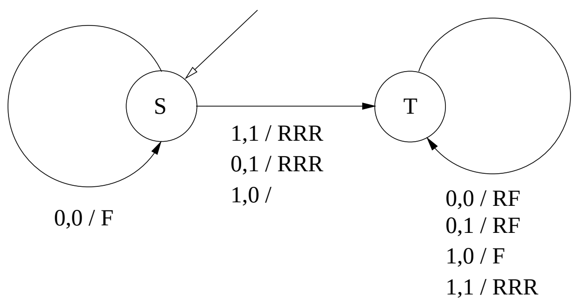
$$\begin{array}{cccccc} A0 \rightarrow 0A & A1 \rightarrow 1A & B0 \rightarrow 0B & B1 \rightarrow 1B & C0 \rightarrow 0C & C1 \rightarrow 1C \\ D0 \rightarrow 0D & D1 \rightarrow 1D & E0 \rightarrow 0E & E1 \rightarrow 1E & F0 \rightarrow 0F & F1 \rightarrow 1F \\ AU \rightarrow 0U & BU \rightarrow 1U & CU \rightarrow 00 & DU \rightarrow 01 & EU \rightarrow 10 & FU \rightarrow 11 \end{array}$$

The non-terminal A or B adjacent to the right end of the word (non-terminal U) is turned into the corresponding bit. As the non-terminals can overhaul terminals (but not other terminals), this procedure clearly inverts the order of the non-terminals yielding the desired second half of the word zz.

If one of the non-terminals C, D, E, or F arrives at the right end U, there are definitely no A's and B's left, and the construction is complete.

4 Transducer-Robot

The following finite state transducer solves the problem. The transducer has two states S (Seek)



and T (Track). The robot starts in the Seek state, not knowing anything about its position. The goal of the Seek state is to find a wall. Once a wall is found, the tracking works by following the *wall to the rear-right* condition. That is, there is always a piece of wall to the right of the robot, or diagonally to the rear-right, or both.

As long as the robot has not sensed any piece of wall ($r = 0, h = 0$), it moves forward with F. If it senses a wall, either ahead or to the right, it positions itself so as to fulfill the *wall to the rear-right* condition. The actions in the tracking state then guarantee that the *wall to the rear-right* condition is never invalidated. Finally, the robot will follow the inside of the wall endlessly, remaining in the tracking state.