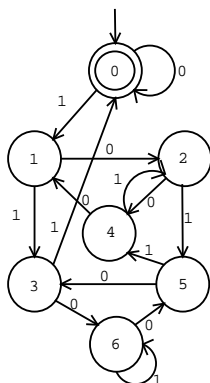


Exercise 1. Finite Automata

a. The FA is constructed using the same principles as exercise 4 of HW 1:



b. In general the machine does **NOT** accept binary numbers divisible by 7 written LSB first. As an example consider the binary number 1001101 (MSB) = 77. If written LSB first, this number is 1011001 = 64 + 16 + 8 + 1 = 89 ≡ 5 mod 7 which will not be accepted by the machine in a.

Exercise 2. Regular languages

Suppose \mathcal{L} is a regular set. Define

$$\mathcal{L}_{odd} \equiv \{a_1 a_3 \dots a_{2n-1} \mid a_1 a_2 \dots a_{2n} \in \mathcal{L}\}$$

Prove that \mathcal{L}_{odd} is regular.

There are (at least) two good strategies for this problem. Both assume that we have a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that accepts \mathcal{L} , and both prove the problem by constructing an M' that accepts \mathcal{L} .

1. Power-Set DFA

Create a new machine

$$M' = (2^Q, \Sigma, \delta', \{q_0\}, F')$$

where the states of the new machine are sets of states from the old machine. Consider the “double-step” function, which describes the set of states of M that can be reached from state s by reading a string $a?$, where $?$ is any single character in Σ :

$$D(s, a) = \bigcup_{c \in \Sigma} \{\delta(\delta(s, a), c)\}$$

We define our new transition function in terms of D :

$$\delta'(S, a) = \bigcup_{s \in S} D(s, a)$$

All that’s left is to construct F' , which is the set of states that contain a final state:

$$F' = \{S \mid S \in 2^Q, S \cap F \neq \emptyset\}$$

2. Even-Odd NFA

Create a new machine

$$M' = (Q', \Sigma, \Delta, \{q_0\}, F)$$

where $|Q'| = 2|Q|$. Let $n = |Q|$. We'll divide the states in Q' into two halves, the "even" states s_0, \dots, s_{n-1} and the "odd" states s_n, \dots, s_{2n-1} . For each transition $\delta(s_i, c) = s_j$ in M we add a normal transition from the even state to the odd state and also add an ϵ transition from the odd state to the even state. More precisely, we add these transitions to Δ :

$$\begin{aligned} s_i &\xrightarrow{c} s_{j+n} \\ s_{i+n} &\xrightarrow{\epsilon} s_j \end{aligned}$$

Finally, we note that F is unchanged from M (i.e. states in F map to even states in M').

Notice that even states can only be reached after an even number of transitions, and odd states can only be reached after an odd number of transitions. For each character in the input and each (even) state that has been reached in the NFA, a single normal transition is taken to an odd state, followed by a set of epsilon transitions back to even states. In this way the machine "guesses" at the missing even characters in $a_1 a_3 \dots a_{2n-1}$.

Note that simply changing "every other" transition in M into an ϵ transition will not work. If M contains a cycle of odd length then *every* transition in that cycle must be both a normal transition and an ϵ transition, which is nonsense.

Exercise 3. Context-free grammars

$$S \rightarrow AB \mid BA \mid A \mid B,$$

$$A \rightarrow CAC \mid a,$$

$$B \rightarrow CBC \mid b,$$

$$C \rightarrow a \mid b.$$

These productions generate two types of strings: (1) All strings of odd length (2) Strings of the form $xayubv$ or $ubvxay$ where $x, y, u, v \in \{a, b\}^*$, $|x| = |y|$, and $|u| = |v|$.

Exercise 4. Pushdown automata

Give a 1-state PDA that accepts $L = a^n b^m \mid n = m \vee n = 2m$

It's pretty easy to write a context-free grammar for this language, using a non-terminal X for the $n = m$ case and Y for the $n = 2m$ case:

$$S ::= X \mid Y$$

$$X ::= aXb \mid \epsilon$$

$$Y ::= aaYb \mid \epsilon$$

Now we can translate this to a grammar in Griebach Normal Form. We have to take out the ϵ productions (we'll fix that later) and ensure that every production is made of a terminal followed by zero or more non-terminals. To accomplish this we expand the beginnings of the X and Y productions inline

in the S production. (Note that there are other acceptable GNF grammars for this language):

$$\begin{aligned} S &::= aX \mid aAY \\ X &::= aXB \mid b \\ Y &::= aAYB \mid b \\ A &::= a \\ B &::= b \end{aligned}$$

Next we follow the recipe given in the book and in class for converting a GNF to a 1-state PDA that accepts by empty stack, being careful to add back an ϵ production to accept the $m = n = 0$ case.:

$$M = (\{q\}, \{a, b\}, \{S, X, Y, A, B\}, \delta, q, S, \{\})$$

$$\delta = \{ \begin{aligned} &((q, \epsilon, S), (q, \epsilon)), \\ &((q, a, S), (q, X)), \\ &((q, a, S), (q, AY)), \\ &((q, a, X), (q, XB)), \\ &((q, b, X), (q, \epsilon)), \\ &((q, a, Y), (q, AYB)), \\ &((q, b, Y), (q, \epsilon)), \\ &((q, a, A), (q, \epsilon)), \\ &((q, b, B), (q, \epsilon)) \end{aligned} \}$$

Exercise 5. Context-free languages

We use closure of intersection between a CFL and a RL (Regular Language) in our proof. Assume L is a CFL and consider the language defined by the following:

$$L \cap a^*b^*c^* = a^n b^n c^n, n \geq 0$$

From the lecture notes (and Kozen) it was shown that $a^n b^n c^n$ is not a CFL. However we know that the intersection between a CFL and a RL is a CFL. Thus we have a contradiction and we conclude that L cannot be a CFL.

Exercise 6. Turing machines

The basic idea behind our machine is that given $1^n\#$, it first puts a zero after the $\#$ and then adds 1 to the binary number after the $\#$ n times. We mark the 1's to keep track of how many of times we've added 1. At the end, we need to unmark them. As in Kozen, we assume that the first cell on the tape is some end-of-tape marker. Our machine performs the following steps.

1. Move the head to the cell immediately after the $\#$ and write a zero. Move back to the first cell. Continue to step 2.
2. Move the head to the right until we read either a 1 or the $\#$. If we read a 1, continue to step 3. Otherwise, we're reading the $\#$ and we go to step 6.
3. The machine is currently reading a 1. Replace the 1 with a X, and then move the head to the right until it reaches the first blank cell on the tape. Continue to step 4.
4. Move the head left one cell. This brings us to the last symbol in the string m after the $\#$. (We know that m is non-empty because of step 1.) We now need to add 1 to m , so we loop over the following 3 cases.

- (a) If the machine is reading a 0, replace it with a 1 and go to step 5.
 - (b) If the machine is reading a 1, replace it with a 0 and repeat this loop.
 - (c) If the machine is reading the #, we need to create a space between the # and m so that we can write the carry. We can do this using the “making space” algorithm from class. Write a 1 into the space, and go to step 5.
5. Move the head to the beginning of the tape. Go back to step 2.
6. The machine is currently reading the #. By this point, the string m after the # is the MSB representation of n in binary, and all the 1's have been marked. Repeatedly move the head left, replacing each X with a 1. The machine halts when it reaches the end of tape marker.