CMPS 130: HW 2

Kevin Wang

February 18, 2019

Solution to Problem 1

Definition 1.1. A language $\mathcal{L}$ is a context-free language (CFL) if there exists some CFG G, such that $L(G) = \mathcal{L}$. [1]

Theorem 1. *All regular languages are context-free languages.*

Proof. Let DFA $D = (Q, \Sigma, \delta, q_0, F)$ accept some regular language, $\mathcal{L}$, such that $L(D) = \mathcal{L}$.

Let PDA $P = (Q, \Sigma, \Gamma, \delta', q_0, \$, F)$ be constructed from DFA D, where

$$\delta : Q \times \Sigma \cup \{\epsilon\} \times \Gamma \longrightarrow 2^{Q \times (\Gamma^* \cup \{\epsilon\})}$$
$$\delta(q, a, \epsilon) = (q', \epsilon)$$

such that the stack of P is not used. Therefore, P is functionally equivalent to DFA D and accepts language $\mathcal{L}$, such that $L(P) = \mathcal{L}$.

By Definition 1.1, $\mathcal{L}$ is a context-free language. Thus, all regular languages are CFL's. □

[1] lec7.pdf

Solution to Problem 2

Definition 2.1. A language $\mathcal{L}$ is a context-free language (CFL) if there exists some CFG G, such that $L(G) = \mathcal{L}$. [2]

Lemma 2.1. *Every regular language is a context-free language.* [3]

Lemma 2.2 (Closure Under Union)**.** *Given CFL's $\mathcal{A}$ and $\mathcal{B}$, then $\mathcal{A} \cup \mathcal{B}$ is also a context-free language.* [4]

Theorem 2. *The language $\mathcal{L} = \{x | x$ is not of the form ww, $w \in \{0,1\}^*\}$ is a CFL.*

Proof. Let $\mathcal{L} = \mathcal{L}_{odd} \cup \mathcal{L}_{even}$ where:

1. $\mathcal{L}_{odd} = \{x | x \in \mathcal{L}, |x| \text{ is odd}\}$
2. $\mathcal{L}_{even} = \{xy | xy \in \mathcal{L}, |x| = |y|, x \neq y\}$

We first observe that $\mathcal{L}$ accepts all words of odd length, as words of form ww have even lengths. We know that $\mathcal{L}_{odd}$ is a regular language[5]. By Lemma 2.1, $\mathcal{L}_{odd}$ is a CFL.

Next, we construct a CFG $G = (V, \Sigma, P, S)$ that accepts a language $L(G)$:

1. $V = \{S, X, Y\}$
2. $\Sigma = \{0, 1\}$
3. $P = \{X \longrightarrow 0X0|0X1|1X0|1X1|0,\ Y \longrightarrow 0Y0|0Y1|1Y0|1Y1|1\}$
4. $S = XY|YX$

We note that for each production, $|X| = |Y|$, and thus $|XY|$ must be even. Furthermore, because $X = \cdots|0$ and $Y = \cdots|1$, there is always at least a single symbol difference between X and Y, and thus, $X \neq Y$. Therefore, by construction of G, $L(G) = \mathcal{L}_{even}$. By Definition 2.1, we know that $\mathcal{L}_{even}$ is a CFL.

Thus, because context-free languages are closed under union (Lemma 2.2), $\mathcal{L}$ is also a CFL.

□

[2] lec7.pdf
[3] lec7.pdf
[4] Wikipedia, Context-free-language, Closure
[5] DFA from lec2.pdf

Solution to Problem 3

Lemma 3.1. *Let M_N be a non-deterministic Turing Machine. There exists a deterministic Turing Machine M_D that simulates M_N, such that $L(M_D) = L(M_N)$.*[6]

Theorem 3. *A Turing Machine with an extra bit in the transition function adds no additional power.*

Proof. Observe that with an extra bit in the transition function, each state–symbol pair has 2 possible transitions. The 2 transitions depend on new bit D being $RIGHT$ or $LEFT/\sqcup$. As such, the extension results in a non-deterministic Turing Machine as there is more than one explicit transition for each state-symbol pair. Note that there is still a finite number of computational transitions.Let NTM N's computational tree be represented by a binary tree with root q_0 and leaf nodes $f \in F$. Left branches are transition functions $\delta(q_i, \rho, LEFT)$ while right branches are transition functions $\delta(q_i, \rho, RIGHT)$.

We can construct a deterministic Turing Machine to simulate N by using Breadth-First Search on N's computational tree. We start with 4 tapes:

1. Memory Tape: stores input string
2. Address Tape: stores the transitions of N's computational tree
3. Direction Tape: stores direction the head moved the last time it visited the cell
4. Scratch Tape: used for running the simulation of N

The Turing Machine then runs the simulation. First, the input string is read from memory into scratch. For each iteration of the simulation, we read D from the parallel cell from the direction tape. We then run BFS to find a valid transition that accepts the state–symbol pair as well as D. Then, we update the direction value to D'.

As there exists a TM that simulates N, the extended Turing Machine is not more powerful. □

[6] lec10.pdf

Solution to Problem 4

Theorem 4.1. *A Turing Machine with a 2-dimensional tape has the same power as a standard Turing Machine.*

Proof. First, we will construct a Turing Machine with 2 tapes:[7]

1. Memory Tape: stores rows of symbols separated by # and is bounded by $ on either end
2. Counter Tape: tracks the column of the head by adding/removing markers every move R/L

Note that a Turing Machine with 2 tapes is equivalent in power to a standard Turing Machine.

When the 2D Turing Machine moves LEFT, we move left on the memory tape. If the cell is at the leftmost boundary of the 2D table, the standard TM will have read in $ or #. The head then moves right by one step, returning to the initial symbol of the row.

When the 2D Turing Machine moves RIGHT, we move right on the memory tape. If the cell is at the current rightmost boundary of the 2D table, the standard TM will have read in $ or #. A blank, ␣, is subsequently inserted by shifting the symbol – and everything right of it – right by one cell. This is equivalent to adding a blank at the end of a row in the 2D Turing Machine's table.

When the 2D Turing Machine moves DOWN, we note the current head column and move left until a # is met. Then we move to the previous # or $ and make moves right until we are at the column specified from before. However, if we move left and a $ is met instead, we return to the original position of the same row (using the column value) as the table is not expandable downwards.

When the 2D Turing Machine moves UP, we note the current head column and move right until a # is met. Then we make moves right until we are at the same column specified from before. However, if we move right and a $ is met instead, we insert a # followed by n blanks, ␣, (where n is the column index) by shifting the $ $n + 1$ cells right. This is equivalent to adding a new row to the top of the 2D Turing Machine's table.

As the 2-dimensional Turing Machine can be simulated by a 2-tape standard Turing Machine, the 2D Turing Machine is not more powerful than a standard TM. □

[7] lec10.pdf