Homework 6

CSC 445-01: Theory of Computation

Matthew Mabrey, Luke Kurlandski

April 26, 2021

4.3

A DFA will recognize Σ^* iff every reachable state is a final state. In a similar fashion to Theorem 4.4, we design the TM T to test whether or not this is the case and decide ALL_{DFA}.

T = "On input A, where A is a DFA:

1. Mark A's start state
2. Do until no new state is marked:
   (a) Mark any state that can be reached via the transition function from a marked state
3. If every marked state is a final state, then *accept*; Else any marked state is not a final state, then *reject*."

4.4

Every CFG has an equivalent Chomsky Normal Form. The rules of a Chomsky Normal CFG state that the only way to generate the empty string is through the rule $S \to \epsilon$. Using this fact, we design a TM T to decide $A\epsilon_{CFG}$.

T = "On input G, where G is a CFG

1. Convert G to a Chomsky Normal Form G' with rules R'
2. If the rule $S \to \epsilon$ is in R', then *accept*; Else *reject*."

4.11 (using a CFG)

We construct a TM I to decide INFINTE$_{PDA}$.

I = "On input M, where $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is a PDA that recognizes $L(P)$:

1. Construct a Context Free Grammar for $L(P)$, $G' = (V', \Sigma', R', S')$
2. Convert G' into Chomsky Normal Form grammar, $G = (V, \Sigma, R, S)$
3. Conduct a search for recursion in the rules, R
   - Use a BFS to prevent "getting stuck", as opposed to DFS
   - For some arbitrary non terminal $A \in V$, if the derivation $A \to uAv$ exists, for $u, v \in \{V \cup \Sigma\}^*$, then recursion exists in the rules
4. If recursion is found *accept*; Else *reject*

4.11 (shaky)

We construct a TM I to decide INFINTE$_{PDA}$.

I = "On input M, where $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ is a PDA that recognizes L_P:

1. Let $k = |\mathscr{P}(Q \times \Gamma_\epsilon)|$
2. Construct DFA, D, to recognize $L_D = \{w \mid |w| > k\}$
3. Construct DFA, M, to recognize $L_M = L_D \cap L_P$
4. Use the E_{DFA} decider Theorem 4.4 and *reject* if $L_M = \emptyset$; Else *accept* if $L_M \neq \emptyset$

Required: For a PDA to accept an infinite number of strings, there cannot exist an upper limit on the length of the strings the PDA accepts, ie, it must accept strings of arbitrary length.

Our PDA has a transition function $\delta : Q \times \Sigma_\epsilon \Gamma_\epsilon \rightarrow \mathscr{P}(Q \times \Gamma_\epsilon)$. Therefore, at any time, the PDA may be in any of $|\mathscr{P}(Q \times \Gamma_\epsilon)|$ different configurations of state and stack. We choose the integer k to be this size. We build a language L_D that contains all words longer than k. Using L_D, we build another language L_M that contains the elements in the language recognized by our PDA L_P that are longer than k. If the language of the PDA L_P contains no words longer than k, then L_M will be empty and our TM will reject. If L_P contains some word longer than k, we accept because that word may be pumped with the pumping lemma for CFGs, meaning that the PDA will accept an infinte number of strings.

5.1

First we define

$$EQ_{CFG} = \{< G_1, G_2 > | G_1 \text{ and } G_2 \text{ are equivalent context free grammars}\}$$
$$ALL_{CFG} = \{< G > | G \text{ is a CFG and } L(G) = \Sigma^*\}$$

We will use a proof by contradiction to show that EQ_{CFG} is undecidable.

Suppose that EQ_{CFG} were decidable by some TM, R. Then we could use R to construct a TM, S, that decides ALL_{CFG}. We describe S in the following paragraph.

On input G, where G is a CFG:

1. Run $< G, G_{\Sigma^*} >$ on R, where $L(G_{\Sigma^*}) = \Sigma^*$
2. *accept* if R accepts; Else *reject*

In summary, machine S uses machine R to compare an input grammar, G, to a grammar whose language is Σ^*. The result of R's computation then determines if $L(G) = \Sigma^*$.

However, we know from Theorem 5.13 that ALL_{CFG} is undecidable. Therefore, we have a contradiction and EQ_{CFG} cannot be decidable.

5.4

No.

We revisit the definition of mapping reducibility. If $A \leq_m B$, then there is a computable function f where

$$w \in A \text{ iff } f(w) \in B$$

f is defined such that some Turing Machine, on input w, halts with the output $f(w)$ on its tape.

However, just because the function $f(w)$ produces strings that belong to the regular language B does not nessecitate that the input strings $w \in A$ form a regular language themselves. So A does not need to be regular for B to be regular.

For further evidence, we provide a counter example. Suppose A is context free language with an alphabet $\Sigma = \{0, 1\}$. If we map all members of A to 1 and all non members of A to 0, then $A \leq_m B$ where B is a regular language with one member: the single character '1'.

5.9 (incomplete)

First we define

$$T = \{< M > | M \text{ is a TM that accepts } w^R \text{ whenever it accepts } w\}$$
$$A_{TM} = \{< M, w > | M \text{ is TM and accepts } w\}$$

We will use a proof by contradiction to show that T is undecidable.

Suppose that T were decidable by some TM, R. Then we could use R to construct a TM, S, that decides A_{TM}. We describe S in the following paragraph.

On input $< M, w >$,

1. Do stuff. Not sure what tho.

However, we know that A_{TM} is undecidable. Therefore, we have a contradiction and T cannot be decidable.

5.22 (shaky on ltr)

We will prove

$$\text{A is Turing-recognizable} \leftrightarrow A \leq_m A_{TM}$$

left-to-right direction

We will prove

$$\text{A is Turing-recognizable} \rightarrow A \leq_m A_{TM}$$

If A is Turing recognizable, then some TM T_A recognizes it. We design a TM T that writes the concatenation of a word, w, and T_A on its tape if T_A accepts w. We describe the TM T:

On input w:

1. Run w on T_A
2. If T_A accepts, write $< T_A, w >$ on the tape and accept. Else write the empty string.

We see that every word in A gets mapped to $< T_A, w >$, ie, some member of A_{TM}. This fulfils the mapping reduction from A to A_{TM}.

right-to-left direction

We will prove

$$A \leq_m A_{TM} \rightarrow \text{A is Turing-recognizable}$$

We know from theorem 5.28 that if $A \leq_m B$ and B is Turing-recognizable, then A is Turing recognizable. A_{TM} is a Turing-recognizable language, thus A must be as well.