Exam 2

Francisco J. Díaz Riollano
Student ID: 802-15-2172

November 30, 2018

Question 1

1. (20 points) Demonstrate that a deterministic Turing machine can be simulated with a pushdown automaton with two stacks.

Definitions:
Turing Machine, $TM = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$.
1) Q is the set of states
2) Σ is the input alphabet not containing the blank symbol.
3) Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4) $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
5) q_0 is the start state
6) q_{accept} is the accept state
7) q_{reject} is the reject state, where $q_{reject} \neq q_{accept}$

2-Stack Pushdown Automata, $P = (Q, \Sigma, \Gamma_{\varepsilon L}, \Gamma_{\varepsilon R}, \delta, q_0, F)$
1) Q is the set of states
2) Σ is the input alphabet not containing the blank symbol.
3) $\Gamma_{\varepsilon L} = \Gamma \cup \varepsilon$ where Γ is the left stack alphabet
4) $\Gamma_{\varepsilon R} = \Gamma \cup \varepsilon$ where Γ is the right stack alphabet
5) $\delta : Q \times \Sigma_\varepsilon \times \Gamma_{\varepsilon R} \times \Gamma_{\varepsilon L} \to (Q \times \Gamma_{\varepsilon L} \times \Gamma_{\varepsilon R})$, where $\Gamma_{\varepsilon L}$ is the the left stack with ε and $\Gamma_{\varepsilon R}$ is the right stack with ε
6) $q_0 \in Q$ is the start state
7) $F \subseteq Q$ is the set of accept states.

The idea is to show that the transition function in a TM; $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is equivalent to the transition function of a 2-Stack Pushdown Automata; $\delta : Q \times \Sigma_\varepsilon \times \Gamma_{\varepsilon R} \times \Gamma_{\varepsilon L} \to Q \times \Gamma_{\varepsilon L} \times \Gamma_{\varepsilon R}$.

The idea is to used the endowed stacks for tracking or counting the number of left or right movements, where each movement count corresponds to a stack in the 2-stack pushdown automata. We define the transition function of the pushdown automata on which the movement to the left of the TM, say $\delta(q_1, 0) = (q_2, \times, L)$ for example, to be equivalent to pushing 0 to the left stack $\Gamma_{\varepsilon L}$ and popping on the right stack stack $\Gamma_{\varepsilon R}$. Vice versa, moving to the right in the TM would be equivalent to pushing the symbol it read to the right stack and performing a pop from the left stack. Thus in general moving to the left would equate to popping from the right and pushing to the left and moving to the right would equate to pushing to right and popping the left stack.

A turing machine cannot read an empty string from the tape alphabet it does however, have $\sqcup$. Thus operations involving $\sqcup$ in the TM would be equivalent to operations involving ε on the 2-Stack Pushdown Automata.

As for the starting, accepting, rejecting states.

In the 2-Stack Push Down Automata q_0 would be the same as q_0 in the TM. F, the subset of accepting states in the 2-Stack Pushdown Automata, must contain q_{accept} state of the TM.

For the rejecting state q_{reject} of the TM must be a state in $Q - F$ of the 2-Stack Push Down Automata in order to be equivalent.

Question 2

2. (a) (10 points) Define formally the concepts of decidable language and Turing-recognizable language, and show that a decidable language is Turing-recognizable but not necessarily the other way around.

1) Decidable languages:
A language L over Σ is said to be to be decidable if there exist a Turing Machine T, such that $L = L(T)$. Such a language is said to be decidable, if on input w, the machine *accepts* if $w \in L$ and the machine $rejects$ otherwise. Often such a machine is called a decider of the language L.

2) Turing recognizable language:
A language L over Σ is said to be recognizable if there exists a Turing Machine T, $L = L(T)$ but T is not necessarily a decider. Meaning such a machine may not necessarily tell whether on input $w \in L$ will *accept* or $reject$.

3) Proof that a decidable language is recognizable but not the other way around:

We have two directions to prove. The first will consist of proving that if a language is decidable it is also recognizable. The other proof will consist of proving that if a language is recognizable does not necessarily imply the language is decidable. The later could be done by proving the existence of language that is recognizable but is not decidable.

a) We will show this, by proving there exist such a language, A_{TM}, that decides whether a Turing Machine is Turing recognizable.

$U =$ "On input $< M, w >$, where M is a TM and w is a string:

1) Simulate M on w.
2) If M ever enters its accept state, *accept*.
3) If M ever enters its reject state,$reject$"

It is said that the Turing Machine U is recognizes A_{TM}, thus every decidable language is turing-recognizable.

b) The other proof will consist of showing that if language is recognizable, it is not necessarily a decidable language.

We assume A_{TM} is decidable and obtain a contradiction.
Suppose H is a decider for A_{TM}. Namely $H(< M, w >)$, *accepts* if M accepts w, $rejects$ if M does not accept w. Now we will construct a new machine D which determines what M does when given its own description M.

$D =$ " On input $< M >$, where M is a TM:

1. Run H on input $< M, < M >>$.
2. Output the opposite of what H outputs; that is, if H accepts $reject$,

if H accepts $reject$.
Now run D on itself.

$D(< D >)$: $accept$ if D does not accept $< D >$,
$reject$ if D accepts $< D >$.
No matter what D does, it is forced to do the opposite, thus cannot decide, forcibly of course but it proves the point. Thus we have a contradiction.

(b) (30 points) Let INF_A = {< A >: A is a finite state automaton and $L(A)$ is infinite }. Is INF_A decidable? Answer with a demonstration or a counterexample.

We want prove that given an Automata, A has an a language $L(A)$, which is infinite.
We cannot test all $w \in \Sigma^*$. However, we have the pumping lemma to help us. A language that satisfies the pumping lemma of finite state machines is of the form xy^iz, where $i > 0$ and $p = |y| > 0$. If we can find a decider such that decides on languages that have such a string with a least length p then, our language is infinite by the pumping lemma.

M = On input A
Let p be the number of states in A (the cardinality of the set of states $|Q|$)
Let Σ be the alphabet of the automata.
Let $k \in \mathbb{N}, k > 1$ (for simplicity choose k =2).
For each string $w \in \Sigma^*$ such that $p \leq |w| \leq k \times p$
    Run $M_{DFA}(< A, w >)$, where M_{DFA} decides whether A accepts w or not
    If M_{DFA} accepts
        Accept
End For Loop
Reject

If this machine *Accepts* it means it has found such a p of the pumping lemma, thus $L(A)$ is infinite. *Rejects* otherwise.

3. (a) (10 points) Define formally the concept of computable function.

A computable function is a function whose values can be computed with a Turing Machine. More formally, a function $f : \Sigma^* \rightarrow \Sigma^*$ is said to be computable, if there is a Turing Machine M which on input $w \in \Sigma^*$, will halt with $f(w)$ on the machine's tape. The machine M may accept or reject w. Also computable functions may not be described with a closed formula. This will be left to reduction which may or may not happen or even exist for that matter.

(b) (30 points) Prove or disprove with a counterexample that the reductions of

(a) A_{T_M} to $HALT$

Proof that A_{TM} is reducible to $HALT$. We begin by constructing a function, f that maps from $< M, w >$ to $< M', w' >$. Where $< M, w > \in A_{TM}$ and $< M', w' > \in HALT$. The machine that computes the function f described above is described as

f = On input $< M, w >$
1) Build a new machine M'
M' = On input x
1. Run M on x
2. If M accepts then *accept*
3. If M rejects then enter a loop
2) Output: $< M', w' >$

Since we have developed a TM that describes f, we can conclude f is a computable function.

(b) E_{TM} to EQ_{TM} are both computable.
Proof that E_{TM} to EQ_{TM} is reducible and computable:

The reduction is quite simple. Testing emptiness of a Turing Machine will be a special case of testing the equivalence of two Turing Machines. We would be testing the equivalence of whether the given Turing Machine also rejects absolutely everything it is given, thus having an empty language.
We begin by constructing a function f that maps from $< M >$ to $< M, M1 >$. Where $< M, M1 > \in EQ_{TM}$ and $< M > \in E_{TM}$.

Let f be the Turing machine that decides E_{TM} and R be the Turing Machine that decides EQ_{TM}.

f = On input $< M >$,where M is a Turing Machine.

1. Run R on input $< M, M1 >$, where $M1$ is a the Turing Machines that rejects all inputs it is given.

2. If R accepts, then *accept*

3. If R rejects, then $reject$

Output $< M, M1 >$