**QUESTION 1.**

Idea: the encoding of an NFA $M$ is a member of $ODD_{NFA}$ as long as $M$ never accepts an even length string (including $\epsilon$). Presented with the encoding of $M$, a TM can determine the alphabet, $\Sigma$, and it can also construct a DFA that accepts all even length strings over $\Sigma$. (An example algorithm is: create 2 states, $q_0$ and $q_1$, make the $s_0$ the only final state, make a transition from $q_0$ to $q_1$ on every symbol in $\Sigma$ and make a transition from $q_1$ to $q_0$ on every symbol in $\Sigma$.) We also have an algorithm for constructing an equivalent DFA from an NFA (Theorem 1.39) and an algorithm for constructing a DFA for the intersection of two regular languages (Sipser page 46). We will use this machinery to reduce $ODD_{NFA}$ to $E_{DFA}$ which we know to be decidable. Construct $A$, a TM for $ODD_{NFA}$ as follows.

$A=\text{“On input } <M> \text{ where } M \text{ is an NFA} \hfill$

1. Construct an equivalent DFA, $M'$, from $M$.
2. Construct a DFA, $N$, that accepts all even length strings over the alphabet $\Sigma$ of $M$.
3. Construct a DFA, $P$, that accepts $L(N) \cap L(M')$.
4. Run $T$, the decider for $E_{DFA}$, on $P$. If it accepts, accept. If it rejects, reject.”

As $A$ is a decider and it decides $ODD_{NFA}$, $ODD_{NFA}$ is decidable. □

Another approach is to classify states as having an even, odd, or both even and odd length path to the final state. (We ignore the implementation details, but note that we can choose to “mark” symbols in such a way that we can keep track of four classifications - “even,” “odd,” “even, odd,” “none” - of the states of $M$ on the tape.) A sketch of an algorithm is as follows:

$A'=\text{“On input } <M> \text{ where } M \text{ is an NFA} \hfill$

1. Begin by marking all final states “even” and all other states “none.”
2. Repeat until no new markings occur:
   - For every transition $r_j \in \delta(r_i, a)$, if $r_j$ is “even,” add “odd” to the classification of $r_i$; if $r_j$ is “odd,” add “even” to the classification of $r_i$; if $r_j$ is “even, odd”, add “even, odd” to the classification of $r_i$; otherwise, add nothing to $r_i$.
   - For every transition $r_j \in \delta(r_i, \epsilon)$, if $r_j$ is “even,” add “even” to the classification of $r_i$; if $r_j$ is “odd,” add “odd” to the classification of $r_i$; if $r_j$ is “even, odd”, add “even, odd” to the classification of $r_i$; otherwise, add nothing to $r_i$.
3. If the start state is classified “odd,” or “none” accept; otherwise, reject. (The start state is classified “none” if there is no path from start to a final state, in which case every string accepted by $M$ is indeed odd.)
2

CSE 555 HW 1 SAMPLE SOLUTION

QUESTION 2.

Idea: we rely on the fact that context-free languages are closed over regular intersection (intersection with regular languages). Again, we use an algorithm that builds a DFA for the language of all even-length strings over $\Sigma$. We then use a product construction to take the intersection of this DFA and $M$ (we need to be careful since $M$’s transition function needs to account for stack operations, $\epsilon$, and can also map to multiple states, but it can be done). We then use the procedure in Lemma 2.27 which shows we can convert a PDA to a CFG. Overall, we are reducing the problem $ODD_{PDA}$ to $E_{CFG}$ which we know to be decidable.

B=“On input $<M>$ where $M$ is a PDA

1. Construct a DFA, $N$, that accepts all even length strings over the alphabet $\Sigma$ of $M$.
2. Construct a PDA, $P$, that accepts $L(N) \cap L(M)$.
3. Convert $P$ to a CFG, $G$.
4. Run $R$, the decider for $E_{CFG}$, on $G$. If it accepts, accept. If it rejects, reject.”

As B is a decider and it decides $ODD_{PDA}$, $ODD_{PDA}$ is decidable. □

Another approach is to classify variables in terms of the number of terminals they can produce. Classes are “none” (or non-productive), “even,” “odd,” and “even, odd.” (Again, we ignore implementation, but note that we could mark the variables in some way to denote the class, such as with no dot, a single dot, a double dot, or a triple dot.) Then, a sketch of the algorithm is as follows:

$B’=“On input <M> where M is a PDA

1. Convert the PDA to a CFG in CNF.
2. Mark all variables as “odd” for which there is some rule $A \rightarrow a, A \in V, a \in \Sigma$ and mark all other variables “none.” If $S \rightarrow \epsilon$, add mark “even” for $S$.
3. Repeat until no new markings occur:
   For every rule, $A \rightarrow BC$, determine a classification of $A$ based on the classifications of $B$ and $C$ and add it to $A$. (Details needed for all possible combinations, but the idea is both even implies “even,” both odd implies “even,” etc.)
4. If the start symbol is classified “odd,” or “none” accept; otherwise, reject.

QUESTION 3.

We show that $A_{TM}$ is mapping reducible to $A_{1}^{1}_{TM}$ and so $A_{1}^{1}_{TM}$ cannot be decidable. Assume, for the purpose of obtaining a contradiction, that $A_{1}^{1}_{TM}$ is decidable. Then, there is a TM, $C$, that decides $A_{2}^{1}_{TM}$. We will also build a TM, $D$ that takes input $w$ and converts it from any base (alphabet) to a unary representation, $w'$. (Without deciding exact implementation details, we know that a TM can add and multiply, so given some mapping $f : \Sigma \rightarrow N$, we can algorithmically convert the encoding of $w$ from $\Sigma$ to unary.) We also use a Turing machine $U$ that takes a unary input and writes as output the corresponding character string in $\Sigma$. Now, let $E$ be a TM that works as follows:
E=“On input \(< M, w >\)
1. Run D on \(w\) to obtain \(w'\).
2. Form a machine \(M'\) that first runs \(U\) and then transfers control to \(M\).
3. Run C on \(< M', w' >\). If C accepts, accept; otherwise, reject.

Then E is a decider for \(A_{TM}\), so we have a contradiction. □

**Question 4.**

First we consider the stronger requirement in the hint. Assume, to the contrary, it is the case that whenever a language \(L\) is Turing-enumerable, there is a Turing machine that enumerates \(L\) so that the strings output satisfy \(|w_{i+1}| \geq |w_i|\) for all \(i \geq 1\). We use the enumerator, E, to build a decider \(D\) for \(L\). On input \(< M, w \rangle\), \(D\) employs E as a subroutine. \(D\) compares the length of string output by E to length of \(w\). If E ever prints \(w\), \(D\) accepts. On the first string printed by E that is strictly longer than the length of \(w\), \(D\) rejects. Since E enumerates strings in non-decreasing order, E will either print \(w\), at which time \(D\) accepts, or it will print a string longer than \(w\), at which point \(D\) rejects, and one of the two occurs in a finite amount of time. (Note: if E never prints a string longer than \(w\), then \(L\) is a finite language, and all finite languages are regular and therefore decidable.) This proves that every (strongly) enumerable language is decidable, which is a contradiction.

Now we treat the weaker length requirement. We use the unary converter \(D\) (implementing Turing-computable function \(d\)) from question 3 again. \(D\) establishes that \(L \leq_m UL\), the language of unary representations of members of \(L\). Now define a second Turing computable function \(p\) that maps the input string \(1^n\) to the output string \(11^{10^n}\). (Here ‘1’ is the symbol used, but “11” means the number eleven.) Let \(PL = \{p(w) : w \in UL\}\). Then the existence of \(p\) establishes that \(UL \leq_m PL\). Any enumeration of \(PL\) meeting the weak length restriction must list strings in non-decreasing order of length, because no two strings of different length can have lengths that are within a factor of 10. As before, we can conclude that an enumeration of \(PL\) so that the strings output satisfy \(|w_{i+1}| \geq \frac{1}{10}|w_i|\) for all \(i \geq 1\) yields a decider for \(PL\). This in turn leads to a decider for \(UL\), and to a decider for \(L\). So if we start with a recognizable but not decidable language for \(L\), we find that \(PL\) cannot be enumerated under the weaker constraint on lengths.

We have yet not established that \(PL\) is itself recognizable. One way to do this is to observe that computing logarithms base 11 of powers of 11 is Turing computable, and so is the conversion back to the original symbol set. Alternatively, we could just build an enumerator for \(PL\) from one for \(L\), by outputting \(p(d(w))\) whenever the enumerator for \(L\) goes to output \(w\). (Be careful with this argument in general; it works in this case because every string in \(PL\) is of the form \(p(d(w))\) for some \(w \in L\).)

**Question 5a.**

Assume to the contrary that \(L\) is Turing-recognizable. Then there is some TM that recognizes \(L\). Then there is some enumerator, \(E\), that enumerates \(L\). We will build an
enumerator, $E'$, for $\overline{A_{TM}}$ that works as follows. $E'$ runs $E$ as a subroutine. For every string $x\#y$ that $E$ prints on its tape, $E'$ trims $x\#$ from the front and prints $y$ on its tape. Then, $E'$ enumerates all $y \in \overline{A_{TM}}$. Then, from $E'$, we can build a recognizer for $\overline{A_{TM}}$, so it is recognizable, which is a contradiction.

Another approach to this problem is very similar to the approach shown in 5b.

**Question 5b.**

Assume to the contrary that $L$ is co-Turing-recognizable. Then, there is some TM, $R$, that recognizes $\overline{L} = \{x\#y | x \in \overline{A_{TM}} \text{ or } y \in \overline{A_{TM}} \} \cup \{\text{the input is incorrectly formatted}\}$. Build a recognizer, $P$, which has a string $<N,z>$ hardcoded into it where that $N$ is an encoding of a TM and $z$ is a string that is not accepted by $N$ (that is, $<N,z> \notin A_{TM}$).

$P$ works as follows:

1. Run $R$ on $<M,w> \# <N,z>$. If it accepts, accept.

Note that $P$ accepts whenever $R$ accepts, which is either if $<M,w> \in \overline{A_{TM}}$ or $<N,z> \in A_{TM}$, but we’ve chosen $<N,z> \notin A_{TM}$. Then, $P$ only accepts if $<M,w> \in \overline{A_{TM}}$, so $P$ is a recognizer for $\overline{A_{TM}}$, which gives a contradiction.

Another approach to this problem is very similar to the approach shown in 5a.