CS 341 Homework 21
Undecidability

1. Which of the following problems about Turing machines are solvable, and which are undecidable? Explain your answers carefully.
(a) To determine, given a Turing machine M, a state q, and a string w, whether M ever reaches state q when started with input w from its initial state.
(b) To determine, given a Turing machine M and a string w, whether M ever moves its head to the left when started with input w.
(c) To determine, given two Turing machines, whether one semidecides the complement of the language semidecided by the other.
(d) To determine, given a Turing machine M, whether the language semidecided by M is finite.

2. Show that it is decidable, given a pushdown automaton M with one state, whether $L(M) = \Sigma^*$. (Hint: Show that such an automaton accepts all strings if and only if it accepts all strings of length one.)

3. Which of the following problems about context-free grammars are solvable, and which are undecidable? Explain your answers carefully.
(a) To determine, given a context-free grammar G, is $\varepsilon \in L(G)$?
(b) To determine, given a context-free grammar G, is $\{\varepsilon\} = L(G)$?
(c) To determine, given two context-free grammars G_1 and G_2, is $L(G_1) \subseteq L(G_2)$?

4. The nonrecursive languages L that we have discussed in class all have the property that either L or the complement of L is recursively enumerable.
(a) Show by a counting argument that there is a language L such that neither L nor its complement is recursively enumerable.
(b) Give an example of such a language.

Solutions

1. (a) To determine, given a Turing machine M, a state q, and a string w, whether M ever reaches state q when started with input w from its initial state. This is not solvable. We can reduce H to it. Essentially, if we can tell whether a machine M ever reaches some state q, then let q be M's halt state (and we can massage M so it has only one halt state). If it ever gets to q, it must have halted. More formally:

L_1 = H = {s = "M" "w" : M halts on input string w}

$\Downarrow \quad \tau$

(?M_2) L_2 = {s : "M" "w" "q" : M reaches state q when started with input w from its initial state}

Let τ' create, from M the machine M* as follows. Initially M* equals M. Next, a new halting state H is created in M*. Then, from each state that was a halting state in M, we create transitions in M* such that for all possible values of the current tape square, M* goes to H. We create no other transitions to H. Notice that M* will end up in H in precisely the same situations in which M halts.

Now let τ("M" "w") = τ'("M") "w" "H"

So, if M_2 exists, then M_1 exists. It invokes τ' to create M*. Then it passes "M*", "w", and "H" to M_2 and returns whatever M_2 returns. But M_1 doesn't exist. So neither does M_2.

(b) To determine, given a Turing machine M and a string w, whether M ever moves its head to the left when started with input w. This one is solvable. We will assume that M is deterministic. We can build the deciding machine D as follows. D starts by simulating the operation of M on w. D keeps track on another tape of each configuration of M that it has seen so far. Eventually, one of the following things must happen:

1. M moves its head to the left. In this case, we say yes.
2. M is stuck on some square s of the tape. In other words, it is in some state p looking at some square s on the tape and it has been in this configuration before. If this happens and M didn't go left yet, then M simply hasn't moved off of s. And it won't from now on, since it's just going to do the same thing at this point as it did the last time it was in this configuration. So we say no.
3. M moves off the right hand edge of the input w. So it is in some state p looking at a blank. Within k steps (if k is the number of states in M), M must repeat some state p. If it does this without moving left, then again we know that it never will. In other words, if the last time it was in the configuration in which it was in state p, looking at a blank, there was nothing to the right except blanks, and it can't move left, and it is again in that same situation, it will do exactly the same thing again. So we say no.

(c) To determine, given two Turing machines, whether one semidecides the complement of the language semidecided by the other. This one is not solvable. We can reduce to it the problem, "Given a Turing machine M, is there any string at all on which M halts?" (Which is equivalent to "Is $L(M) = \varnothing$?") In the book we show that this problem is not solvable. What we'll do is to build a machine M* that semidecides the language Σ^*, which is the complement of the language $\varnothing$. If we could build a machine to tell, given two Turing machines, whether one semidecides the complement of the language semidecided by the other, then to find out whether any given machine M accepts anything, we'd pass M and our constructed M* to this new machine. If it says yes, then M accepts $\varnothing$. If it says no, then M must accept something. Formally:

$L_1 =$ {s = "M" M halts on some string w}

$\Downarrow \quad \tau$

(?M_2) $L_2 =$ {s = "M_1" "M_2" : M_1 decides the complement of the language semidecided by M_2}

M accepts strings over some input alphabet Σ. Let τ' construct a machine M* that semidecides the language Σ^*. Then τ("M") = "M" "τ'(M)".

So, if M_2 exists, then M_1 exists. It invokes τ' to create M*. Then it passes "M" and "M*" to M_2 and returns the opposite of whatever M_2 returns (since M2 says yes if $L(M) = \varnothing$ and M1 wants to say yes if $L(M) \neq \varnothing$). But M_1 doesn't exist. So neither does M_2.

(d) To determine, given a Turing machine M, whether the language semidecided by M is finite. This one isn't solvable. We can reduce to it the problem, "Given a Turing machine M, does M halt on ε?" We'll construct, from M, a new machine M*, which erases its input tape and then simulates M. M* halts on all inputs iff M halts on ε. If M doesn't halt on ε, then M* halts on no inputs. So there are two situations: M* halts on all inputs (i.e., L(M*) is infinite) or M* halts on no inputs (i.e., L(M*) is finite). So, if we could build a Turing machine M_2 to decide whether L(M*) is finite or infinite, we could build a machine M_1 to decide whether M halts on ε. Formally:

$L_1 =$ {s = "M" M halts on ε}

$\Downarrow \quad \tau$

(?M_2) $L_2 =$ {s = "M" is finite}

Let τ construct the machine M* from "M" as described above.

So, if M_2 exists, then M_1 exists. It invokes τ to create M* which accepts a finite language precisely if M accepts ε. But M_1 doesn't exist. So neither does M_2.

2. M only has one state S. If S is not a final state, then $L(M) = \varnothing$, which is clearly not equal to Σ^*, so we say no. Now suppose that S is a final state. Then M accepts ε. Does it also accept anything else? To accept any single character c in Σ, there must be a transition ((S, c, ε), (S, ε)). In other words, we must be able to end up in S with an empty stack if, looking at an empty stack, we see c. If there is not such a transition for every element c of Σ, then we say no, since we clearly cannot get even all the one character strings in Σ^*. Now, suppose that all those required transitions do exist. Then, we can stay in S with an empty stack (and thus accept) no matter what character we see next and no matter what is on the stack (since these transitions don't check the stack). So, if M accepts all strings in Σ^* of length one, then it accepts all strings in Σ^*. Note that if M is deterministic, then if it does have all the required transitions it will have no others, since all possible configurations are accounted for

3. (a) To determine, given a context-free grammar G, is $\varepsilon \in L(G)$ This is solvable by using either top down or bottom up parsing on the string ε.

(b) To determine, given a context-free grammar G, is $\{\varepsilon\} = L(G)$ This is solvable. By the context-free pumping theorem, we know that, given a context-free grammar G generating a language L(G), if there is a string of length greater than B^T in L, then vy can be pumped out to create a shorter string also in L (the string must be shorter since $|vy| > 0$). We can, of course, repeat this process until we reduce the original string to one of length less than B^T. This means that if there any strings in L, there are some strings of length less than B^T. So, to see whether $L = \{\varepsilon\}$, we do the following: First see whether $\varepsilon \in L(G)$ by parsing. If not, we say no. If ε is in L, then we need to determine whether any other strings are also in L. To do this, we test all strings in Σ^* of length up to B^{T+1}. If we find one, we say no, $L \neq \{\varepsilon\}$. If we don't find any, we can assert that $L = \{\varepsilon\}$. Why? If there is a longer string in L and we haven't found it yet, then we know, by the pumping theorem, that we could pump out vy until we got a string of length B^T or less. If ε were not in L, we could just test up to length B^T and if we didn't find any elements of L at all, we could stop, since if there were bigger ones we could pump out and get shorter ones but there aren't any. However, because ε is in L, what about the case where we pump out and get ε? That's why we go up to B^{T+1}. If there are any long strings that pump out to ε, then there is a shortest such string, which can't be longer than B^{T+1} since that's the longest string we can pump out (by the strong version of the pumping theorem).

(c) To determine, given two context-free grammars G_1 and G_2, is $L(G_1) \subseteq L(G_2)$ This isn't solvable. If it were, then we could reduce the unsolvable problem of determining whether $L(G_1) = L(G_2)$ to it. Notice that $L(G_1) = L(G_2)$ iff $L(G_1) \subseteq L(G_2)$ and $L(G_2) \subseteq L(G_1)$. So, if we could solve the subset problem, then to find out whether $L(G_1) = L(G_2)$, all we do is ask whether the first language is a subset of the second and vice versa. If both answers are yes, we say yes. Otherwise, we say no. Formally:

$L_1 = \{s : s = G_1\ G_2,\ G_1 \text{ and } G_2 \text{ are context-free grammars, and } L(G_1) = L(G_2)\}$

$\Downarrow \quad \tau$

(?M_2) $L_2 = \{s : s = G_1\ G_2,\ G_1 \text{ and } G_2 \text{ are context-free grammars, and } L(G_1) \subseteq L(G_2)\}$

If M_2 exists, then $M_1(G_1\ G_2) = M_2(G_1\ G_2)$ AND $M_2(G_2\ G_1)$. To write this out in our usual notation so that the last function that gets applied is M_2, is sort of tricky, but it can, of course be done: Don't worry about doing it. If you can write any function for M_1 that is guaranteed to be recursive if M_2 exists, then you've done the proof.

4. (a) If any language L is recursively enumerable, then there is a Turing machine that semidecides it. Every Turing machine has a description of finite length. Therefore, the number of Turing machines, and thus the number of recursively enumerable languages, is countably infinite (since the power set of a countable set is countably infinite). If, for some language L, its complement is re, then it must have a semideciding Turing machine, so there is a countably infinite number of languages whose complement is recursively enumerable. But there is an uncountable number of languages. So there must be languages that are not recursively enumerable and do not have recursively enumerable complements.

(b) L = {"M" : M halts on the input 0 and M doesn't halt on the input 1}.
The complement of L = {"M" : M doesn't halt on the input 0 or M halts on the input 1}. Neither of these languages is recursively enumerable because of the doesn't halt piece.