Main topics of the week:
- Countable and uncountable sets
- Diagonalization proof of uncountable
- Undecidable halting problem $A_{TM}$
- Complement of $A_{TM}$ is not even Turing recognizable
- Review for final

Review problems for final.

Let $G$ be the grammar

$$
S \rightarrow aS \mid aA \mid a \\
A \rightarrow aAb \mid ab
$$

Give a concise description of $L(G)$ and a PDA to accept $L(G)$.

We observe that the symbol $A$ generates $a^nb^n$ for all $n \geq 1$. The rules for $S$ using $aA$ and $a$ simply add another ‘a’ in front, so that gives us strings of the form $aa^n b^n$ for all $n \geq 0$. The rule $aS$ for $S$ allows an arbitrary number of $a$’s to be added at the beginning, so $L(G) = \{ a^m b^n \mid m > n \geq 0 \}$.

Show that the C programming language is not a context free language.

Proof: Suppose that $C$ is a CFL. By the pumping lemma for CFLs, there must be a pumping length $p$. Consider the legal C program:

```c
f(){int aa...a;aa...a;aa...a;}
```

In this program, the variable is $a^p$, i.e., the symbol ‘a’ repeated $p$ times. Suppose we realize this string as $uvxyz$, where $|vxy| \leq p$. If $vy$ contains the blank or any of the symbols preceding it, then we certainly have a syntax error in $uxz$ or at least an undeclared variable. If $vy$ contains the last semi-colon or closing brace, then again $uxz$ will have a syntax error. So $vxy$ must be between the space and the last semi-colon. If $vy$ contains a semi-colon and possibly some characters before and after, again $uxz$ will have an undeclared variable since one set of the $a$’s will combine to be at least $p+1$ $a$’s while the other stays the same. If $vxy$ contains no semi-colon, then $uvxyzz$ will have a second identifier, so again the program will not be legal since something will be undeclared. This exhausts all the possibilities; so legal C programs are not a context free language.
Let \( A = \{<R,S> \mid R \text{ and } S \text{ are regular expressions and } L(R) \subseteq L(S) \} \) Show that \( A \) is decidable.

**Proof:** We describe a TM that decides \( A \):

1. On input \(<R,S>\) where \( R \) and \( S \) are regular expressions,
   1) Convert the regular expression \( R \) to an equivalent DFA \( A \), using the procedure for converting a regular expression to an NFA, and the procedure for converting an NFA to a DFA.
   2) Likewise convert the regular expression \( S \) to a DFA \( B \).
   3) Using the procedure from an exercise, convert \( B \) to a DFA \( C \) that recognizes the complement of \( L(S) \).
   4) Using the procedure from the problem on the midterm (just like the one for the union of regular languages), construct a DFA \( D \) from \( A \) and \( C \) to recognize \( L(R) \cap \neg L(S) \).
   5) Run the TM for \( \text{E}_{\text{DFA}} \) on \( D \) to determine if \( L(R) \cap \neg L(S) = \emptyset \).
   6) If it is empty, accept, otherwise reject.

Note that each stage of our TM uses procedures that we know to halt, so this is a decider. By determining if the intersection of \( L(R) \) with the complement of \( L(S) \) is empty, we determine whether \( L(R) \subseteq L(S) \) or not.

Let \( S = \{<M> \mid M \text{ is a DFA that accepts } w^R \text{ whenever } M \text{ accepts } w\} \). Show that \( S \) is decidable.

**Proof:** We describe a TM that decides \( S \). Basically we test to see whether the language recognized by \( M \) is the same as the reverse of that language.

1. On input \(<M>\), where \( M \) is a DFA,
   1) Construct a DFA \( N \) to recognize \( \{w^R \mid w \in L(M)\} \). (Recall that we saw in a problem how to construct an NFA to do this by adding a single accept state and reversing all the arrows, and we know how to convert an NFA to a DFA).
   2) Run the TM for \( \text{EQ}_{\text{DFA}} \) with input \(<M,N>\).

If it accepts, accept, otherwise reject."