Problems for Week Nine

Regular vs. Context-Free

For each of the following languages, determine whether it is (a) regular or (b) context-free but NOT regular, and prove that your choice is correct. (Note: if you choose (a), you may want to exhibit an automaton or a regular expression—I recommend choosing whichever you feel *less* comfortable with. If you choose (b), observe that you will need to prove two things.)

i. $\Sigma = \{a, b\}$ and $L = \{(ab)^n \mid n \in \mathbb{N}\}$.

ii. $\Sigma = \{a, b\}$ and $L = \{(ab)^n a^n \mid n \in \mathbb{N}\}$.

iii. $\Sigma = \{a, b\}$ and $L = \{(ab)^n a^m \mid n, m \in \mathbb{N}$ and the total number of a's is even $\}$.

iv. $\Sigma = \{a, b\}$ and $L = \{ w \in \Sigma^* \mid$ every prefix of w has at least as many a's as b's $\}$. (This one's tricky!)

Turing Machines

Although much of our discussion of Turing machines takes place at a high level, it's still instructive to try to design Turing machines at the level of individual states.

i. Let $\Sigma = \{0, 1\}$ and let $L = \{ w \in \Sigma^* \mid w$ is a palindrome $\}$ (recall that a palindrome is a string that's the same when read forwards and backwards). Draw a state-transition diagram of a TM for L.

ii. Draw the state-transition diagram for a TM whose language is $\{ a^n b^n c^n \mid n \in \mathbb{N} \}$.

The Story So Far

From the “lava diagram” in lecture, you probably noticed that

$$\mathbf{REG} \subseteq \mathbf{R} \subseteq \mathbf{RE}$$

Here, **REG** is the class of all regular languages, **R** is the class of all decidable languages, and **RE** is the class of all recognizable languages.

On Problem Set Eight, you’ll show that **REG** ⊆ **R**.

i. Show that **REG** ≠ **R**.

ii. Show that **R** ⊆ **RE**. *(Hint: What's the definition of* ***R****? What's the definition of* ***RE****? Expand out the requisite terms and see what you find.)*

Closure Properties of R

This question explores various closure properties of **R**. Because **R** corresponds to decidable problems, languages in **R** are precisely the languages for which you can write a method

```
bool inL(string w)
```

such that

- for any string $w \in L$, calling *inL(w)* returns true.
- for any string $w \in L$, calling *inL(w)* returns false.

This means that we can reason about closure properties of the decidable languages by writing actual pieces of code.

i. Let L_1 and L_2 be decidable languages over the same alphabet Σ. Prove that $L_1 \cup L_2$ is also decidable. To do so, suppose that you have methods *inL1* and *inL2* matching the above conditions, then show how to write a method *inL1uL2* with the appropriate properties. Then, briefly justify why your construction is correct.

ii. Repeat problem (i), except proving that the **R** languages are closed under concatenation.

Decidable Languages

All regular languages are decidable, but below is a purported proof that the regular language described by the regular expression *a*b* is undecidable:

> ***Theorem:*** *a*b* is undecidable.
>
> ***Proof:*** By contradiction; assume *a*b* is decidable. Let *D* be a decider for it. Consider what happens when we run *D* on a string of infinitely many *a*'s followed by a *b*, and on a string of infinitely many *a*'s. Let's call this first string *x* and the second string *y*. Since *D* is a decider, it halts on all inputs, and therefore cannot run for an infinitely long time. Therefore, *D* must halt before reading the last character of *x* and the last character of *y*. Because *x* and *y* are the same except for their last character, we see that *D* must have the same behavior when run on *x* and when run on *y*. If *D* accepts *x*, then *D* also accepts *y*, but *y* is not in the language *a*b*. Otherwise, *D* rejects *x*, but *x* is in the language *a*b*. Both cases contradict the fact that *D* is a decider for *a*b*. We have reached a contradiction, so our assumption must have been wrong. Thus *a*b* is undecidable. ■

What's wrong with this proof?