Extra Practice Problems 12

We got a number of requests over Piazza for more practice problems on the Myhill-Nerode theorem, designing CFGs, and proving undecidability and unrecognizability. Here's some more practice problems on those topics. Hope this helps!

Problem One: The Myhill-Nerode Theorem

Many of the languages you designed CFGs for on Problem Set Eight are not regular.

i. Let $\Sigma = \{\texttt{a}, \texttt{b}\}$ and let $L = \{ w \in \Sigma^* \mid w \text{ is a palindrome and } |w| \geq 10 \}$. (Recall that a palindrome is a string that's the same when read forwards and backwards.) For example, we have $\texttt{abaaaaaaba} \in L$ and $\texttt{abbbababbba} \in L$, but $\texttt{abba} \notin L$ and $\texttt{bbbbbbbbba} \notin L$. Prove that L is not a regular language.

ii. Let $\Sigma = \{\texttt{a}, \texttt{b}\}$ and let $L = \{ w \in \Sigma^* \mid w \text{ is } \textit{not} \text{ a palindrome} \}$, the language of strings that are not the same when read forwards and backwards. For example, $\texttt{aab} \in L$ and $\texttt{baabab} \in L$, but $\texttt{aba} \notin L$, $\texttt{bb} \notin L$, and $\varepsilon \notin L$. Prove that L is not a regular language.

iii. Let $\Sigma = \{\texttt{1}, \texttt{+}, \texttt{=}\}$ and let $L = \{ \texttt{1}^m\texttt{+}\texttt{1}^n\texttt{=}\texttt{1}^{m+n} \mid m, n \in \mathbb{N} \}$. For example, the strings $\texttt{111+1=1111}$ and $\texttt{+1=1}$ are in the language L, but $\texttt{1+11=11}$ is not, nor is $\texttt{1+1+1=111}$. Prove that L is not a regular language.

iv. Let Σ be an alphabet containing these symbols:

$$\varnothing \quad \mathbb{N} \quad \{ \quad \} \quad , \quad \cup$$

Let $L = \{ w \in \Sigma^* \mid w \text{ is a syntactically valid string representing a set} \}$. Prove that L is not a regular language.

Problem Two: Nonregular Languages via a Different Path

The Myhill-Nerode theorem is a powerful tool for proving that languages aren't regular, but it might not be the easiest way to prove that a given language isn't regular. This problem explores a different route you can take to prove that various languages aren't regular.

i. Prove that if L_1 is a language, L_2 is a regular language, and $L_1 \cap L_2$ is not regular, then L_1 is not regular.

ii. Using your result from part (i), but without using the Myhill-Nerode theorem, prove that the language $L = \{ w \in \{\texttt{a}, \texttt{b}\}^* \mid w \text{ has the same number of } \texttt{a}\text{'s as } \texttt{b}\text{'s} \}$ is not regular.

Problem Three: Designing CFGs

Below is a list of alphabets and languages over those alphabets. Design a CFG for each of these languages.

i. Let $\Sigma = \{1, \geq\}$ and let $L = \{ 1^m \geq 1^n \mid m, n \in \mathbb{N} \text{ and } m \geq n \}$. Write a CFG for L.

ii. On Problem Set 8, you explored the language $L_1 = \{ 1^m+1^n=1^{m+n} \mid m, n \in \mathbb{N} \}$ over the alphabet {1, +, =} Consider the following generalization of this language, which we will call L_2, which consists of all strings describing unary encodings of two sums that equal one another. For example:

1 + 3 = 4 would be encoded as `1+111=1111`

4 = 1 + 3 would be encoded as `1111=1+111`

2 + 2 = 1 + 3 would be encoded as `11+11=1+111`

2+0+2+0=0+4+0 would be encoded as `11++11+=+1111+`

0=0 would be encoded as `=`

Notice that there can be any number of summands on each side of the `=`, but there should be exactly one `=` in the string; thus `1=1=1` $\notin L_2$. Write a CFG for L_2.

iii. Let $\Sigma = \{$`(`, `)`, `[`, `]`$\}$ and let $L = \{w \in \Sigma^* \mid w$ is a string of balanced parentheses and brackets$\}$. This means that all parentheses and brackets must match one another, and collectively they must obey the appropriate nesting rules. For example, `([])[]` $\in L$, but `([)]` $\notin L$. Write a CFG for L.

Problem Four: Formalizing the Lava Diagram

In the Guide to the Lava Diagram, we explored these two languages:

$$L_1 = \{ \langle M \rangle \mid M \text{ is a TM and } |\mathscr{L}(M)| \geq 2 \}$$

$$L_2 = \{ \langle M \rangle \mid M \text{ is a TM and } |\mathscr{L}(M)| = 2 \}$$

The Guide makes many claims about these languages, but never actually proves them.

i. Prove that L_1 is undecidable.

ii. Prove that L_2 is undecidable.

iii. Show that L_1 is recognizable by designing a verifier for it. Your verifier should be represented in pseudocode via a method with a signature like this one:

```
bool imConvincedIsInL1(TM M, Arg1Type arg1, Arg2Type arg2, …, ArgnType argn)
```

where the arguments beyond the first represent the certificate and can be of any type you'd like.

Problem Five: Self-Reference and RE

Since A_{TM} is an **RE** language, there's a *recognizer* for A_{TM}, which we can represent in software as a method willAccept. Consider the following program *P*:

```
int main() {
    string me = mySource();
    string input = getInput();

    if (willAccept(me, input)) {
        reject();
    } else {
        accept();
    }
}
```

Prove that this program loops on all inputs.

Problem Six: Threat Detection

There have been a ton of news articles about computer systems being attacked by independent actors and nation-states. You might wonder why our computers are so vulnerable – couldn't someone just write a program that analyzes a program's source code and determine whether it has any security problems?

Let's consider a simplified scenario. Imagine that there's a special method

```
void sendSecretDataTo(String emailAddress)
```

that, if called, sends an email containing a bunch of secret information to the specified email address. For example, you might call `sendSecretDataTo("john.roberts@supremecourt.gov")` to send all the secret data to Chief Justice John Roberts, or call `sendSecretDataTo("bad.actor@hackers.com")` to send all the secret data to evil hackers.

You are interested in whether it's possible to write a method

```
bool canLeakDataTo(String program, String emailAddress)
```

that takes as input the source code of a program and an email address, then returns true if there is some execution of `program` that causes the secret data to the specified email address and returns false otherwise. This program would let you check whether a particular program might ever leak data to a specified email address, which would make it easier to check whether the program is secure.

Is it possible to implement this method? If so, write code for the method, then prove that your code works as intended. If not, prove that it's not possible to implement this method.

(As a note, if you try implementing this method, you should do so in a way that doesn't call `sendSecretDataTo`, and if you try proving this method can't be written, you can assume that no correct implementation will ever call `sendSecretDataTo`.)