CS103 Handout 25S
Fall 2013 November 18, 2013

Solutions to Extra Practice Problems

Problem One: Binary Relations

Prove that $R = R^2$ if R is reflexive and transitive. Recall that $R = R^2$ iff for any $x, y \in A$, we have that xRy iff xR^2y.

Proof: Let R be a relation that is reflexive and transitive. We will prove that $R = R^2$. To show that $R = R^2$, we will prove that for any $x, y \in A$, that xRy iff xR^2y. To do so, we prove both directions of implication:

(⇒) First, we prove that for any $x, y \in A$, if xRy, then xR^2y. Consider any $x, y \in A$ where xRy. Since R is reflexive, we know that xRx holds for all $x \in A$. Therefore, since xRx and xRy, we know that xR^2y, as required.

(⇐) Next, we prove that for any $x, y \in A$, if xR^2y, then xRy. Consider any $x, y \in A$ where xR^2y. This means that there is some $z \in A$ such that xRz and zRy. Since R is transitive, since xRz and zRy, we know that xRy, as required.

Thus xRy iff xR^2y, so $R = R^2$, as required. ■

Why we asked this question: This question, in addition to being more practice on proofs with relations, is a good testbed for manipulating biconditionals. The overall statement to be proved is an implication, but the consequent is a biconditional. If you take the time to spell out each individual step in the proof and avoid jumping too far ahead, you can avoid accidentally getting the biconditional wrong.

Here's a question to ponder: does the other direction of the implication hold? That is, if $R = R^2$, is R necessarily reflexive and transitive?

Problem Two: Diagonalization

Let $L_{\text{co-D}} = \{ \langle M \rangle \mid M \text{ is a TM and } M \text{ rejects } \langle M \rangle \}$. Prove that $L_{\text{co-D}} \notin$ co-**RE** by using a proof by diagonalization.

Proof: By contradiction; assume that $L_{\text{co-D}} \in$ co-**RE**. This means that there is a co-recognizer R for $L_{\text{co-D}}$. Since R is a co-recognizer for $L_{\text{co-D}}$, we know that for any TM M that $\langle M \rangle \notin L_{\text{co-D}}$ iff R rejects $\langle M \rangle$. By definition of $L_{\text{co-D}}$, for any TM M we have $\langle M \rangle \notin L_{\text{co-D}}$ iff M does not reject $\langle M \rangle$. Therefore, we see that for any TM M, M does not reject $\langle M \rangle$ iff R rejects $\langle M \rangle$. This in turn holds if we choose $M = R$, so we have that R does not reject $\langle R \rangle$ iff R rejects $\langle R \rangle$, which is impossible. We have reached a contradiction, so our assumption must have been wrong. Thus $L_{\text{co-D}} \notin$ co-**RE**. ■

Why we asked this question: For any TM M, the language of that TM, $\mathscr{L}(M)$, is the set of all strings that the TM accepts. We can define the "co-language" of a TM, denoted co-$\mathscr{L}(M)$, to be the set of all strings that the TM doesn't reject. Since a co-recognizer for a language L should reject all strings not in L and not reject any strings in L, the co-language of M is the language for which M is a co-recognizer.

Given this, the language $L_{\text{co-D}}$ could be defined equivalently as

$$L_{\text{co-D}} = \{ \langle M \rangle \mid M \text{ is a TM and } \langle M \rangle \notin \text{co-}\mathscr{L}(M) \}$$

From here, it's a lot clearer where the parallel arises with L_{D} and why the proofs would be similar.

More to the point of why we asked this question: between lecture and the handouts, there are a total of three proofs that $L_{\text{D}} \notin$ **RE**. This problem asks you to prove that $L_{\text{co-D}} \notin$ co-**RE**, and the structure of the proof is quite similar. We hoped this would give you practice working with diagonal languages.

Problem Three: First-Order Logic

In what follows, let's assume the domain of discourse is a nonempty set of people, so all quantifer range over people.

Consider the predicate *Drinks*(*p*), which says that *p* is currently drinking. The *drinker's paradox* is the following statement:

$$\exists p.\ (Drinks(p) \to \forall q.\ Drinks(q))$$

This says "there is someone where if that person is drinking, then *everyone* is drinking."

i. Explain why the above statement is always true, regardless of who's drinking.

Proof: If everyone is drinking, the statement is true because the consequent of the implication is true, and anything implies a true statement. Otherwise, someone is not drinking. Choose *p* to be someone not drinking. Then the implication holds because the antecedent is false, and the overall statement is true. ■

ii. Is the above statement equivalent to the following statement?

$$(\exists p.\ Drinks(p)) \to (\forall q.\ Drinks(q))$$

This statement is not equivalent to the original. Suppose someone, but not everyone, is drinking. Then the antecedent of this statement is true but the consequent is false, so the statement is false. This statement more accurately says "either no one is drinking or everyone is drinking."

Why we asked this question: We've stressed the importance of pairing → with ∀ and ∧ with ∃, and we figured this problem would be a good way for you to understand some of the strange results that can happen when you start to mix the two. Part (ii) of this problem, in particular, was designed to make sure that you were comfortable understanding how parentheses can change the meaning of a first-order expression.

Problem Four: Regular Expressions

Below are some alphabets and languages over those alphabets. Design a regular expression for each of those languages.

i. Let Σ = {**a**, **b**} and let $L = \{ w \mid |w| \geq 1$ and w starts and ends with the same symbol. $\}$ Write a regular expression for L.

One possibility is

`Σ | aΣ*a | bΣ*b`

All strings of length one are in the language and are handled by the first part of the regular expression, while strings of length greater than one are handled by the last two parts (which match strings bookended by each character in Σ.)

ii. Let Σ = {**0** , **1**} and let $L = \{ \langle w_1, w_2 \rangle \mid w_1, w_2 \in \Sigma^* \}$. That is, L is the set of all pairs of strings encoded using the encoding format used on Problem Set Seven. Write a regular expression for L.

One possibility is

`(00|11)*01Σ*`

The idea here is to generate a string made of any number of copies of duplicated characters, then the separator **`01`**, then anything.

iii. Let Σ = {**a**, **b**} and let $L = \{ w \in \Sigma^* \mid w$ has an even number of **b**'s $\}$. Write a regular expression for L.

One possibility is

`a*(ba*ba*)*`

If there are an even number of **b**'s in the string, then the string can be formed by placing down two **b**'s at a time, filling in the gaps between them with **a**'s. There's a "fencepost problem" with this regular expression where there will always be one more block of **a**'s than there are **b**'s in the string, so we need to generate that extra block of **a**'s separately.

Why we asked this question: Parts (i) and (iii) of this question were designed to get you thinking about edge cases of regular expressions. Both of those problems require some amount of special handling to account for all strings. Part (ii) of this problem was designed to connect the material from regular expressions to the material on encodings that we're covering now.

Problem Five: Context-Free Grammars

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

One possibility is

$S \to \mathtt{a}S\mathtt{b} \mid A \mid B$

$A \to \mathtt{a}A \mid \mathtt{a}$

$B \to \mathtt{b}B \mid \mathtt{b}$

Intuitively, the start symbol generates balanced strings of **a**'s and **b**'s, then decides whether to add in extra **b**'s to imbalance the count or extra **a**'s.

ii. Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and let $L = \{ w \in \Sigma^* \mid \text{there are more } \mathtt{a}\text{'s than } \mathtt{b}\text{'s in } L \}$. Write a CFG for L.

One possibility is

$S \to E\mathtt{a}T$

$T \to E\mathtt{a}T \mid E$

$E \to \mathtt{a}E\mathtt{b}E \mid \mathtt{b}E\mathtt{a}E \mid \varepsilon$

This grammar is based on the grammar for “equal numbers of **l**'s and **r**'s” from Problem Set 6. Intuitively, any string with more **a**'s than **b**'s can be split apart into blocks of strings with the same number of **a**'s and **b**'s, with the additional **a**'s interspersed in-between them. The nonterminal E generates a block of characters with the same number of **a**'s and **b**'s in it. The nonterminals S and T, collectively, generate a string of the form $E(\mathtt{a}E)^+$, which represents a string of balanced blocks with at least one extra **a** interspersed between them.

iii. (Challenge problem!) Let $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ and let $L = \{\ xy \mid x, y \in \Sigma^*, |x| = |y|, \text{ and } x \neq y\ \}$. In other words, L is the language of all strings that whose first half is not the same as its second half. Write a CFG for L.

One possibility is

S → AB | BA

A → **aAa** | **aAb** | **bAa** | **bAb** | **a**

B → **aBa** | **aBb** | **bAa** | **bBb** | **b**

This one is tricky. The key insight is that any two strings that aren't equal must have some position at which they disagree. We can generate two strings of some fixed length that disagree at position k by using the regular expression $\mathbf{\Sigma}^n\mathbf{a}\mathbf{\Sigma}^m$ to generate one string and $\mathbf{\Sigma}^n\mathbf{b}\mathbf{\Sigma}^m$to generate the other, where $m, n \in \mathbb{N}$ are constants determining how many characters appear before and after the mismatched character. This means that we can reframe the language as this one:

$$L = \{\ xy \mid \text{there exists some } m, n \in \mathbb{N} \text{ such that } x = \mathbf{\Sigma}^n\mathbf{a}\mathbf{\Sigma}^m \text{ and } y = \mathbf{\Sigma}^n\mathbf{b}\mathbf{\Sigma}^m \text{ or vice-versa}\ \}$$

This is in turn equivalent to

$$L = \{\ \mathbf{\Sigma}^n\mathbf{a}\mathbf{\Sigma}^m\,\mathbf{\Sigma}^n\mathbf{b}\mathbf{\Sigma}^m \mid m, n \in \mathbb{N}\ \} \cup \{\ \mathbf{\Sigma}^n\mathbf{b}\mathbf{\Sigma}^m\,\mathbf{\Sigma}^n\mathbf{a}\mathbf{\Sigma}^m \mid m, n \in \mathbb{N}\ \}$$

The beautiful insight necessary to solve the problem is that any string of the form

$$\mathbf{\Sigma}^n\mathbf{a}\mathbf{\Sigma}^m\,\mathbf{\Sigma}^n\mathbf{b}\mathbf{\Sigma}^m$$

has the form

$$\mathbf{\Sigma}^n\mathbf{a}\mathbf{\Sigma}^{m+n}\mathbf{b}\mathbf{\Sigma}^m$$

which is in turn equivalent to

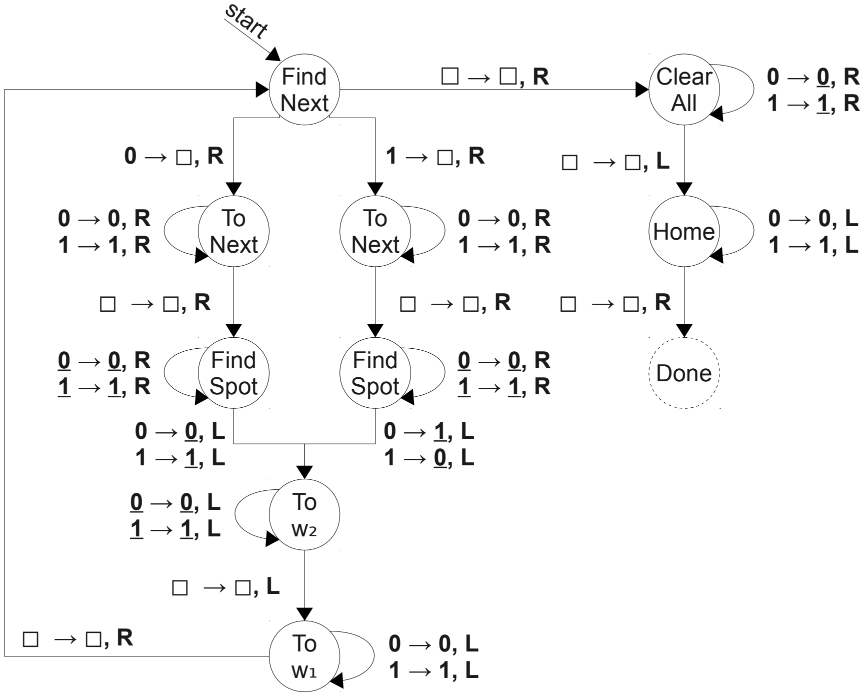
$$(\mathbf{\Sigma}^n\mathbf{a}\mathbf{\Sigma}^n)(\mathbf{\Sigma}^m\mathbf{b}\mathbf{\Sigma}^m)$$

Notice the structure of this string – it's an **a** surrounded on both sides by n characters, following be a **b** surrounded on both sides by m characters. All strings of this form are in L, and the only strings in L are strings that have this form, or the same form but with the **a** and **b** swapped.

The grammar given here works by deciding whether the **a** or **b** comes first, then surrounding each one of them independently with some number of symbols on each side.

Why we asked this question: CFGs can be tough to design, though there are many repeating patterns (using the nonterminals to keep track of some "state," building strings from both ends simultaneously, breaking the input apart into multiple pieces and building each independently, etc.) We hoped that these problems would help you get a sense for how to apply those concepts in practice.

Problem Six: Turing Machines



This TM works by repeatedly crossing off a character from the left string, finding the first character in the string on the right that it hasn't yet processed, then applying the XOR operator to the value it read in the first part and the new character. As it does so, it adds more and more marks to the string on the right. Once all characters from the first string have been consumed, the TM then unmarks all the characters in the right string and repositions the tape head to the front of that string.

Why we asked this question: There are many ways to solve this problem. The approach we outlined here uses constant storage (to remember which number has been read), extra tape symbols (to mark which characters in the second string have been used), and subroutines (one to do the XOR, one to clear all the marks). We hoped this would give you a better sense for how to combine together all the TM techniques we've seen so far!