Discussion Solutions 5

Problem One: Nonregular Languages

Let $L = \{ w \in \{\texttt{0}, \texttt{1}, \texttt{2}\}^* \mid w \text{ contains the same number of copies of the substrings } \texttt{01} \text{ and } \texttt{10} \}$. Prove that L is not a regular language.

Proof: Let $S = \{ (\texttt{012})^n \mid n \in \mathbb{N} \}$. S is infinite, since it contains one string for each natural number. Next, consider any two strings $(\texttt{012})^n$, $(\texttt{012})^m \in S$. Then $(\texttt{012})^n(\texttt{210})^n \in L$ because it contains n copies of $\texttt{01}$ and n copies of $\texttt{10}$, but $(\texttt{012})^m(\texttt{210})^n \notin L$ because it contains m copies of $\texttt{01}$ and n copies of $\texttt{10}$. Therefore, S is an infinite set of strings distinguishable relative to L, so by the Myhill-Nerode theorem L is not regular. ■

Why we asked this question: Many of the questions on Problem Set 6 involve the Myhill-Nerode theorem, and we wanted to give you more examples to draw from as you started to work on that problem set. We selected this question in particular because it relates back to Problem 1.ii on Problem Set Five, which asked you to show that a variation on this language was indeed regular.

Problem Two: Designing CFGs

i. Let Σ = { `p`, `∧`, `∨`, ¬, →, ↔, `(`, `)`, ⊤, ⊥ } and let *PL* = { *w* ∈ Σ* | *w* is a legal propositional logic formula using just the variable *p* }. Write a CFG for *PL*.

One option is the following:

S → `p` | S `∧` S | S `∨` S | ¬S | S → S | S ↔ S | `(`S`)` | ⊤ | ⊥

ii. Let Σ = {`0`, `1`} and consider the regular expression *R* = `(0|(10)*)*|10*`. Write a CFG *G* such that $\mathscr{L}(R) = \mathscr{L}(G)$.

One way to do this is to iteratively refine the regular expression down into a CFG. We start with

S → `(0|(10)*)*` | `10*`

To handle the stars on both sides, we'll introduce some new nonterminals:

S → A | `1`B
A → `(0|(10)*)`A | ε
B → `0`B | ε

To fix up the nested | in the first production of A, we add more nonterminals:

S → A | `1`B
A → CA | ε
B → `0`B | ε
C → `0` | `(10)*`

Finally, to fix up the last star, we add one more nonterminal:

S → A | `1`B
A → CA | ε
B → `0`B | ε
C → `0` | D
D → `10`D | ε

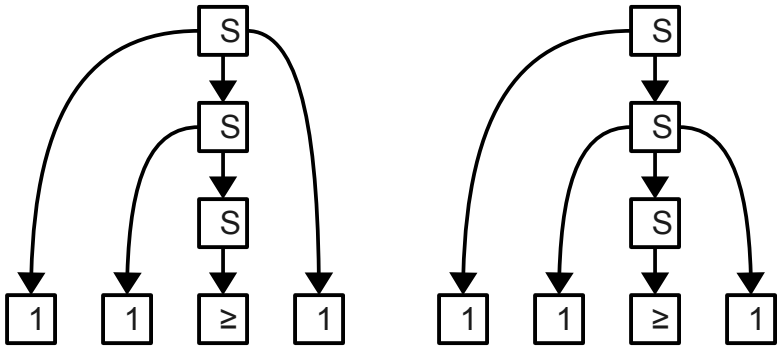
Why we asked this question: We asked part (i) of this question to get you accustomed to connecting context-free grammars with languages that you've seen before (in this case, the language of propositional logic formulas). We hoped that you would see how to use the syntax of propositional logic as a guide.

We asked part (ii) of this question to show you how to turn structures in regular expressions into equivalent structures in context-free grammars. Although CFGs don't support star or nested ORs, those concepts can be encoded simply once you know how to do it.

Problem Three: Uncertainty about Ambiguity

i. Show that this grammar is ambiguous by providing a string in *GE* and two different parse trees for that string.

Here are two parse trees for string **11≥1**:



ii. Rewrite this grammar so that it is unambiguous. Explain, but do not formally prove, why your new grammar is unambiguous.

One solution is

$$S \rightarrow \mathtt{1}S \mid T$$

$$T \rightarrow \mathtt{1}T\mathtt{1} \mid \mathtt{\geq}$$

The intuition is to build the string in two steps. First, lay down as many extra **1**'s as you would like at the front of the string. Next, lay down a balanced number of **1**'s, and finally place a **≥** in the middle.

Why we asked this question: Ambiguity is a tricky concept in CFGs and requires some diligence to remove. In this question, we wanted you to see how to "layer" the grammar into two pieces, one of which generates a balanced string and one of which generates an imbalanced string.