Section 08: Solutions

1. Regular Expressions

(a) Write a regular expression that matches base 10 numbers (e.g., there should be no leading zeroes).

Solution:

\[ 0 \cup (1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)(0 \cup 1 \cup 2 \cup 3 \cup 4 \cup 5 \cup 6 \cup 7 \cup 8 \cup 9)^* \]

(b) Write a regular expression that matches all base-3 numbers that are divisible by 3.

Solution:

\[ 0 \cup ((1 \cup 2)(0 \cup 1 \cup 2)^*0) \]

(c) Write a regular expression that matches all binary strings that contain the substring “111”, but not the substring “000”.

Solution:

\[ (01 \cup 001 \cup 1^*)^*(0 \cup 00 \cup \epsilon)111(01 \cup 001 \cup 1^*)^*(0 \cup 00 \cup \epsilon) \]

(d) Write a regular expression that matches all binary strings that do not have any consecutive 0’s or 1’s.

Solution:

\[ ((01)^*(0 \cup \epsilon)) \cup ((10)^*(1 \cup \epsilon)) \]

(e) Write a regular expression that matches all binary strings of the form $1^k y$, where $k \geq 1$ and $y \in \{0, 1\}^*$ has at least $k$ 1’s.

Solution:

\[ 1(0 \cup 1)^*1(0 \cup 1)^* \]

Explanation: While it may seem like we need to keep track of how many 1’s there are, it turns out that we don’t. Convince yourself that strings in the language are exactly those of the form $1x$, where $x$ is any binary string with at least one 1. Hence, $x$ is matched by the regular expression $(0 \cup 1)^*1(0 \cup 1)^*$.

2. CFGs

Write a context-free grammar to match each of these languages.
(a) All binary strings that start with 11.

**Solution:**

\[
S \rightarrow 11T \\
T \rightarrow 1T | 0T | \varepsilon
\]

(b) All binary strings that contain at most one 1.

**Solution:**

\[
S \rightarrow ABA \\
A \rightarrow 0A | \varepsilon \\
B \rightarrow 1 | \varepsilon
\]

(c) All strings over 0, 1, 2 with the same number of 1s and 0s and exactly one 2.

Hint: Try modifying the grammar from Section 8 2c for binary strings with the same number of 1s and 0s (You may need to introduce new variables in the process).

**Solution:**

\[
S \rightarrow 2T | T2 | ST | TS | 0S1 | 1S0 \\
T \rightarrow TT | 0T1 | 1T0 | \varepsilon
\]

\(T\) is the grammar from Section 8 2c. It generates all binary strings with the same number of 1s and 0s. \(S\) matches a 2 at the beginning or end. The rest of the string must then match \(T\) since it cannot have another 2. If neither the first nor last character is a 2, then it falls into the usual cases of matching 0s and 1s, so we can mostly use the same rules as \(T\). The main change is that \(SS\) becomes \(ST|TS\) to ensure that exactly one of the two parts contains a 2. The other change is that there is no \(\epsilon\) since a 2 must appear somewhere.
3. Recursively Defined Sets of Strings

For each of the following, write a recursive definition of the sets satisfying the following properties. Briefly justify that your solution is correct.

(a) Binary strings of even length.

Solution:

\[\begin{align*}
\text{Basis: } & \forall \varepsilon \in S. \\
\text{Recursive Step: } & \forall x \in S, \exists x00, x01, x10, x11 \in S. \\
\text{Exclusion Rule: } & \forall x \in S, \exists x \in S.
\end{align*}\]

"Brief" Justification: We will show that \(x \in S\) iff \(x\) has even length (i.e., \(|x| = 2n\) for some \(n \in \mathbb{N}\)). (Note: "brief" is in quotes here. Try to write shorter explanations in your homework assignment when possible!)

Suppose \(x \in S\). If \(x\) is the empty string, then it has length 0, which is even. Otherwise, \(x\) is built up from the empty string by repeated application of the recursive step, so it is of the form \(x_1x_2...x_n\), where each \(x_i \in \{00, 01, 10, 11\}\). In that case, we can see that \(|x| = |x_1| + |x_2| + ... + |x_n| = 2n\), which is even. Now, suppose that \(x\) has even length. If it's length is zero, then it is the empty string, which is in \(S\). Otherwise, it has length \(2n\) for some \(n > 0\), and we can write \(x\) in the form \(x_1x_2...x_n\), where each \(x_i \in \{00, 01, 10, 11\}\) has length 2. Hence, we can see that \(x\) can be built up from the empty string by applying the recursive step with \(x_1\), then \(x_2\), and so on up to \(x_n\), which shows that \(x \in S\).

(b) Binary strings not containing 10.

Solution:

If the string does not contain 10, then the first 1 in the string can only be followed by more 1s. Hence, it must be of the form \(0^m1^n\) for some \(m, n \in \mathbb{N}\).

\[\begin{align*}
\text{Basis: } & \forall \varepsilon \in S. \\
\text{Recursive Step: } & \forall x \in S, \exists 0x \in S \text{ and } x1 \in S. \\
\text{Exclusion Rule: } & \forall x \in S, \exists x \in S.
\end{align*}\]

Brief Justification: The empty string satisfies the property, and the recursive step cannot place a 0 after a 1 since it only adds 0s on the left. Hence, every string in \(S\) satisfies the property.

In the other direction, from our discussion above, any string of this form can be written as \(y = 0^m1^n\) for some \(m, n \in \mathbb{N}\). We can build up the string \(y\) from the empty string by applying the rule \(x \rightarrow 0x\) \(m\) times and then applying the rule \(x \rightarrow x1\) \(n\) times. This shows that the string \(y\) is in \(S\).

(c) Binary strings not containing 10 as a substring and having at least as many 1s as 0s.

Solution:

These must be of the form \(0^m1^n\) for some \(m, n \in \mathbb{N}\) with \(m \leq n\). We can ensure that by pairing up the 0s with 1s as they are added:

\[\begin{align*}
\text{Basis: } & \forall \varepsilon \in S.
\end{align*}\]
**Recursive Step:** If \( x \in S \), then \( 0x1 \in S \) and \( x1 \in S \).

**Exclusion Rule:** Each element of \( S \) is obtained from the basis and a finite number of applications of the recursive step.

**Brief Justification:** As in the previous part, we cannot add a 0 after a 1 because we only add 0s at the front. And since every 0 comes with a 1, we always have at least as many 1s as 0s.

In the other direction, from our discussion above, any string of this form can be written as \( xy \), where \( x = 0^n1^m \) and \( y = 1^{n-m} \), since \( n \geq m \). We can build up the string \( x \) from the empty string by applying the rule \( x \rightarrow 0x1 \) \( m \) times and then produce the string \( xy \) by applying the rule \( x \rightarrow x1 \) \( n-m \) times, which shows that the string is in \( S \).

(d) Binary strings containing at most two 0s and at most two 1s.

**Solution:**

This is the set of all binary strings of length at most 4 except for these:

\[
\begin{align*}
000, & \quad 1000, & \quad 0100, & \quad 0010, & \quad 0001, & \quad 0000, & \quad 111, & \quad 0111, & \quad 1011, & \quad 1101, & \quad 1110, & \quad 1111
\end{align*}
\]

Since this is a finite set, we can define it recursively using only basis elements and no recursive step.

### 4. More CFGs

Write a context-free grammar to match each of these languages.

(a) All binary strings that end in 00.

**Solution:**

\[
S \rightarrow 0S | 1S | 00
\]

(b) All binary strings that contain at least three 1's.

**Solution:**

\[
\begin{align*}
S \rightarrow & \quad TTT \\
T \rightarrow & \quad 0T | T0 | 1T | 1
\end{align*}
\]

(c) All binary strings with an equal number of 1’s and 0’s.

**Solution:**

\[
S \rightarrow 0S1S | 1S0S | \varepsilon
\]

and

\[
S \rightarrow SS | 0S1 | 1S0 | \varepsilon
\]

both work. Note: The fact that all the strings generated have the property is easy to show (by induction) but the fact that one can generate all strings with the property is trickier. To argue this that each of these is grammars is enough one would need to consider how the difference between the # of 0s seen and the # of 1’s seen occurs in prefixes of any string with the property.
(d) All binary strings of the form $xy$, where $|x| = |y|$, but $x \neq y$.

**Solution:**

\[
S \rightarrow AB \mid BA \\
A \rightarrow 0 \mid 0A0 \mid 0A1 \mid 1A0 \mid 1A1 \\
B \rightarrow 1 \mid 0B0 \mid 0B1 \mid 1B0 \mid 1B1
\]

**Explanation:** We will explain the forward direction (i.e., this grammar generates strings of the desired form); in particular, we will examine strings generated by the rule $AB$, as the other rule follows similarly.

An arbitrary string generated by $AB$ will look like $a_10a_2b_11b_2$, where $a_1, a_2, b_1, b_2 \in \{0, 1\}^*$, $|a_1| = |a_2| = k_1$, and $|b_1| = |b_2| = k_2$ for some $k_1, k_2 \in \mathbb{N}$. In particular, we can “repartition” the substring $a_2b_1$ into $a_2'b_1'$ s.t. $|a_2'| = k_2$ and $|b_1'| = k_1$. Letting $x = a_10a_2'$ and $y = b_1'b_2$, observe that $|x| = |y| = k_1 + k_2 + 1$ and $x$ and $y$ differ at the $(k_1 + 1)$-th character.
5. Reversing a Binary Tree

Consider the following definition of a (binary) Tree.

**Basis Step** Nil is a Tree.

**Recursive Step** If L is a Tree, R is a Tree, and x is an integer, then Tree(x, L, R) is a Tree.

The sum function returns the sum of all elements in a Tree.

\[
\begin{align*}
\text{sum}(\text{Nil}) & = 0 \\
\text{sum}(\text{Tree}(x, L, R)) & = x + \text{sum}(L) + \text{sum}(R)
\end{align*}
\]

The following recursively defined function produces the mirror image of a Tree.

\[
\begin{align*}
\text{reverse}(\text{Nil}) & = \text{Nil} \\
\text{reverse}(\text{Tree}(x, L, R)) & = \text{Tree}(x, \text{reverse}(R), \text{reverse}(L))
\end{align*}
\]

Show that, for all Trees T that

\[\text{sum}(T) = \text{sum}(\text{reverse}(T))\]

**Solution:**

For a Tree T, let \(P(T)\) be “\(\text{sum}(T) = \text{sum}(\text{reverse}(T))\)”. We show \(P(T)\) for all Trees T by structural induction.

**Base Case:** By definition we have \(\text{reverse}(\text{Nil}) = \text{Nil}\). Applying sum to both sides we get \(\text{sum}(\text{Nil}) = \text{sum}(\text{reverse}(\text{Nil}))\), which is exactly \(P(\text{Nil})\), so the base case holds.

**Inductive Hypothesis:** Suppose \(P(L)\) and \(P(R)\) hold for some arbitrary Trees L and R.

**Inductive Step:** Let x be an arbitrary integer. Goal: Show \(P(\text{Tree}(x, L, R))\) holds.

We have,

\[
\begin{align*}
\text{sum}(\text{reverse}(\text{Tree}(x, L, R))) & = \text{sum}(\text{Tree}(x, \text{reverse}(R), \text{reverse}(L))) & \text{[Definition of reverse]} \\
& = x + \text{sum}(\text{reverse}(R)) + \text{sum}(\text{reverse}(L)) & \text{[Definition of sum]} \\
& = x + \text{sum}(R) + \text{sum}(L) & \text{[Inductive Hypothesis]} \\
& = x + \text{sum}(L) + \text{sum}(R) & \text{[Commutativity]} \\
& = \text{sum}(\text{Tree}(x, L, R)) & \text{[Definition of sum]}
\end{align*}
\]

This shows \(P(\text{Tree}(x, L, R))\).

**Conclusion:** Therefore, \(P(T)\) holds for all Trees T by structural induction.