Problem Set 7

Due: Wednesday, March 5th by 11:00pm

Instructions

Solutions submission. You must submit your solution via Gradescope. In particular:

- Submit a *single* PDF file containing your solutions to Tasks 1–4, 6 (and optionally 7). Follow the prompt on Gradescope to link tasks to your pages.
- Task 5 should be submitted **only** on `grin`. The instructions for appear below that problem.
- Task 6 should be submitted **first** on `grin`. However, your grammar with explanations also need to be submitted as part of your PDF in Gradescope.

Task 1 – Weak in the Trees [16 pts]

Recall the definition of rooted binary trees (with no data) from lecture:

> **Basis Step**: $\bullet \in$ **Tree**
> **Recursive Step**: if $L \in$ **Tree** and $R \in$ **Tree**, then $\text{Tree}(L, R) \in$ **Tree**.

Note that, in lecture, we drew "$\text{Tree}(L, R)$" as a picture of a tree, whereas here we are using more normal (functional) notation, which should be easier for calculations.

Next, we define the following operation which swaps the left and right subtrees throughout a tree:

$$\begin{aligned} \text{reverse}(\bullet) &:= \bullet \\ \text{reverse}(\text{Tree}(L, R)) &:= \text{Tree}(\text{reverse}(R), \text{reverse}(L)) \quad \forall L, R \in \textbf{Tree} \end{aligned}$$

Now, consider the following claim about the mirroring operation:

$$\forall\, T \in \textbf{Tree}\, (\text{reverse}(\text{reverse}(T)) = T)$$

Prove this claim by structural induction.

Task 2 – List and Shout [20 pts]

Recall the definition of lists of numbers from lecture:

> **Basis Step**: $\texttt{nil} \in$ **List**
> **Recursive Step**: for any $a \in \mathbb{Z}$, if $L \in$ **List**, then $a :: L \in$ **List**.

For example, the list $[1, 2, 3]$ would be created recursively from the empty list as $1 :: (2 :: (3 :: \texttt{nil}))$. We will consider "::" to associate to the right, so $1 :: 2 :: 3 :: \texttt{nil}$ means the same thing.

The next problem uses two recursively-defined functions. The first is len, which calculates the length of the list. As we saw in lecture, it is defined recursively as follows:

$$\begin{array}{rcll} \text{len}(\texttt{nil}) & := & 0 & \\ \text{len}(a :: L) & := & 1 + \text{len}(L) & \forall a \in \mathbb{Z}, \forall L \in \textbf{List} \end{array}$$

The second function, echo-pos, which drops any non-positive numbers *and* replaces any positive numbers with two copies of that number, is defined by:

$$\begin{array}{rcll} \text{echo-pos}(\texttt{nil}) & := & \texttt{nil} & \\ \text{echo-pos}(a :: L) & := & \text{echo-pos}(L) & \text{if } a \leqslant 0 \quad \forall a \in \mathbb{Z}, \forall L \in \textbf{List} \\ \text{echo-pos}(a :: L) & := & a :: a :: \text{echo-pos}(L) & \text{if } a > 0 \quad \forall a \in \mathbb{Z}, \forall L \in \textbf{List} \end{array}$$

For example, from these definitions, we get $\text{echo-pos}(-1 :: 2 :: -3 :: \texttt{nil}) = 2 :: 2 :: \texttt{nil}$.

a) Write a chain of equalities, citing the appropriate definitions, showing that

$$\text{echo-pos}(1 :: -2 :: 3 :: \texttt{nil}) = 1 :: 1 :: 3 :: 3 :: \texttt{nil}$$

b) Write a chain of equalities, citing the appropriate definitions, showing that

$$\text{len}(1 :: 1 :: 3 :: 3 :: \texttt{nil}) = 4$$

c) Use structural induction to prove that

$$\forall L \in \textbf{List}\, (\text{len}(\text{echo-pos}(L)) \leqslant 2\,\text{len}(L))$$

Hint: since the definition of echo-pos is split into cases ($a > 0$ and $a \leqslant 0$), you will likely need to argue by (the same) cases in your inductive step!

While this is an English proof, remember how cases work formally to prove a proposition Q: first, prove $(a > 0) \to Q$; then, prove $(a \leqslant 0) \to Q$; and finally, state the tautology "$a > 0$ or "$a \leqslant 0$" and apply Cases. The only difference in English is that we want to tell the reader that we will use cases at the *beginning* of the proof, rather than at the end.

One way to say this in English is "We will argue by cases, $a > 0$ or $a \leqslant 0$.". After that, we prove the two implications. Finally, at the end, we state that "Since one of those cases must hold, we can see that Q holds in general'.'

Task 3 – Long Live the String [18 pts]

Recall the definition of strings over an alphabet Σ from lecture:

> **Basis Step**: $\varepsilon \in \Sigma^*$
> **Recursive Step**: for any $a \in \Sigma$, if $w \in \Sigma^*$, then $wa \in \Sigma^*$.

For example, with $\Sigma = \{a, b, c\}$, the string "abc" would be created recursively as $\varepsilon abc = ((\varepsilon a)b)c$

> The string concatenation operator $\bullet$ is defined recursively as follows:

$$\begin{array}{rcll} x \bullet \varepsilon & := & x & \forall x \in \Sigma^* \\ x \bullet (wa) & := & (x \bullet w)a & \forall a \in \Sigma, \forall x, w \in \Sigma^* \end{array}$$

Let Σ be a fixed alphabet. The following claim says that string concatenation is associative:

$$\forall x, y, z \in \Sigma^* \left((x \bullet y) \bullet z = x \bullet (y \bullet z)\right)$$

Prove this claim by structural induction on z.

Task 4 – A Few of My Favorite Strings [12 pts]

For each of the following, write a recursive definition of the set of strings satisfying the given properties. *Briefly* justify that your solution is correct. This should be 3-4 sentences per part explaining the intuition why (1) any string produced by the basis and recursive step is in the described set and (2) any string in the described set is in the recursive set. You may find the section examples helpful, but your explanations do not need to be as long as the section solutions.

a) Binary strings that start with 10 and have an even length.

b) Binary strings where every 1 is followed by an even number of 0s. That is, if s contains a 1, meaning it can be written as $s = x1 \bullet y$, for some strings x, y, then y *must* have an even number of 0s.

c) Binary strings with an odd number of 1s where every 1 is immediately followed by a 0.

Task 5 – Hope Strings Eternal [10 pts]

For each of the following, construct regular expressions that match the given set of strings:

a) Binary strings that start with 10 and have an even length.

b) Binary strings where every 1 is followed by an even number of 0s. That is, if s contains a 1, meaning it can be written as $s = x1 \bullet y$, for some strings x, y, then y *must* have an even number of 0s.

c) Binary strings with an odd number of 1s where every 1 is immediately followed by a 0.

d) Binary strings with at most three 1s.

e) Binary strings with at most three 1s **or** at least two 0s.

> Submit and check your answers to this question here:
>
> http://grin.cs.washington.edu
>
> Think carefully about your answer to make sure it is correct before submitting. You have only **5 chances** to submit a correct answer.

Task 6 – Glitz and Grammar [15 pts]

For each of the following, construct a context-free grammar that generates the given set of strings.

If your grammar has more than one variable (also called a non-terminal symbol), write a sentence describing what sets of strings you expect each variable in your grammar to generate. For example, if your grammar was:

$$\begin{aligned} \mathbf{S} &\rightarrow \mathbf{E} \mid \mathbf{O} \\ \mathbf{O} &\rightarrow \mathbf{EC} \\ \mathbf{E} &\rightarrow \mathbf{EE} \mid \mathbf{CC} \\ \mathbf{C} &\rightarrow 0 \mid 1 \end{aligned}$$

You could say "C generates binary strings of length one, E generates (non-empty) even length binary strings, and O generates odd length binary strings." It is also fine to use a regular expression, rather than English, to describe the strings generated by a variable (assuming such a regular expression exists). If you only use one variable (i.e., S), an explanation is not required but you should provide the grammar.

a) Binary strings matching the regular expression "$1(0 \cup 10)^*0 \cup 001$".

Hint: You can use the technique described in lecture to convert this RE to a CFG.

b) All strings of the form $y\#x$, with $x, y \in \{0,1\}^*$ and y a subsequence of x^R.

(Here x^R means the reverse of x. Also, a string w is a subsequence of another string z if you can delete some characters from z to arrive at w.)

c) All binary strings in the set $\{0^m1^n : m, n \in \mathbb{N} \text{ and } m > n\}$.

> Submit and check your answers to this question here:
>
> http://grin.cs.washington.edu
>
> Think carefully about your answer to make sure it is correct before submitting. You have only **5 chances** to submit a correct answer.
>
> **Note**: You must also include each grammar and sentences describing each new non-terminal, as described above, in Gradescope.

Task 7 – Extra Credit: With a Grammar, the Whole World is a Nail [0 pts]

Consider the following context-free grammar.

⟨**Stmt**⟩ → ⟨**Assign**⟩ | ⟨**IfThen**⟩ | ⟨**IfThenElse**⟩ | ⟨**BeginEnd**⟩
⟨**IfThen**⟩ → if condition then ⟨**Stmt**⟩
⟨**IfThenElse**⟩ → if condition then ⟨**Stmt**⟩ else ⟨**Stmt**⟩
⟨**BeginEnd**⟩ → begin ⟨**StmtList**⟩ end
⟨**StmtList**⟩ → ⟨**StmtList**⟩⟨**Stmt**⟩ | ⟨**Stmt**⟩
⟨**Assign**⟩ → `a := 1`

This is a natural-looking grammar for part of a programming language, but unfortunately the grammar is "ambiguous" in the sense that it can be parsed in different ways (that have distinct meanings).

a) Show an example of a string in the language that has two different parse trees that are meaningfully different (i.e., they represent programs that would behave differently when executed).

b) Give **two different grammars** for this language that are both unambiguous but produce different parse trees from each other.