CSE 311: Foundations of Computing I

Homework 7 (due May 29th at 11:00 PM)

Directions: *Write up carefully argued solutions to the following problems. Your solution should be clear enough that it should explain to someone who does not already understand the answer why it works. However, you may use results from lecture, the theorems handout, and previous homeworks without proof.*

1. Grammar School (15 points)

For each of the following, construct context-free grammars that generate the given set of strings.

If your grammar has more than one variable, we will ask you to write a sentence describing what sets of strings you expect each variable in your grammar to generate. For example, if your grammar was:

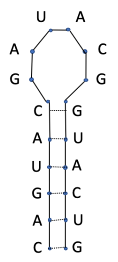
$$\begin{aligned} \mathbf{S} &\to \mathbf{E} \mid \mathbf{O} \\ \mathbf{O} &\to \mathbf{EC} \\ \mathbf{E} &\to \mathbf{EE} \mid \mathbf{CC} \\ \mathbf{C} &\to 0 \mid 1 \end{aligned}$$

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).

(a) [5 Points] Binary strings matching the regular expression "$(1 \cup 01 \cup 001)^*(0 \cup \varepsilon)$".
Hint: You can use the procedure described in lecture to convert the RE to a CFG.

(b) [5 Points] RNA strings that can can fold into a lollipop shape.

Each RNA string is a string over the alphabet $\{A, C, G, U\}$. The string CAGUACGAUACGGUACUG can fold into a lollipop shape like this:



The prefix CAGUAC and suffix GUACUG can together form the handle of the lollipop because they line up in such a manner that As are across from Us (or vice versa) and Cs are across from Gs (or vice versa). The candy part of the lollipop consists of the string GAUACG, which has length 6.

In general, an RNA string can fold into a lollipop shape if it can be written as xyz, where x and z contain at least two characters, y contains at least four characters, and the characters of x match up with those of z^R, in the sense that the the corresponding pairs of letters fall in the set $\{(A, U), (U, A), (C, G), (G, C)\}$.

(c) [5 Points] $\{0^m 1^n 0^{n+2m} : m, n \geq 0\}$

2. University Relations (15 points)

Consider the set of all students on campus. In each part of this problem, we define a relation R on this set. For each one, state whether R is or is not reflexive, symmetric, antisymmetric, and/or transitive. (No proofs.)

(a) [5 Points] $(a, b) \in R$ if students a and b take exactly the same set of classes in the Spring 2019 quarter.

(b) [5 Points] $(a, b) \in R$ if every class that is taken by a is also taken by student b.

(c) [5 Points] $(a, b) \in R$ if students a and b have no classes in common this quarter.

3. Distant Relations (10 points)

Let A be a set. Prove or disprove each of the following claims:

(a) [5 Points] If R and S are transitive relations on A, then $R \cap S$ is transitive.

(b) [5 Points] If R and S are transitive relations on A, then $R \cup S$ is transitive.

4. Lean, Mean String Machine [Online] (30 points)

For each of the following, create a *DFA* that recognizes exactly the language given.

(a) [5 Points] Binary strings where every 00 is immediately followed by a 1.

(b) [5 Points] Binary strings with at most two 0s.

(c) [5 Points] Binary strings with at least two 1s.

(d) [5 Points] Binary strings with at most two 0s **and** at least two 1s.

(e) [5 Points] Binary strings with either at most two 0s or at least two 1s but not both.

(f) [5 Points] $\{0^m 1^n 0^x : m \geq 0 \text{ and } n > 0 \text{ and } x \equiv n + 2m \pmod 3\}$

> Submit and check your answers to this question here:
>
> `https://grinch.cs.washington.edu/cse311/fsm`
>
> Think carefully about your answer to make sure it is correct before submitting. You have only 5 chances to submit a correct answer.

5. Association Football (40 points)

Let $\mathcal{A} = \{\ldots, p, q, r, \ldots\}$ be a fixed set of atomic propositions. We then define the set **Prop** as follows:

Basis Elements: for any $p \in \mathcal{A}$, $\texttt{Atomic}(p) \in$ **Prop**.

Recursive Step: if $A, B \in$ **Prop**, then $\texttt{Wedge}(A, B) \in$ **Prop**.

The set **Prop** represents parse trees of propositions, except that we only allow the propositions to be combined using one operator, "wedge" (the name of $\wedge$ in LaTeX).

The following function, "prop", converts a parse tree into the actual proposition it represents:

$$\begin{aligned} \text{prop}(\texttt{Atomic}(p)) &= p && \text{for any } p \in \mathcal{A} \\ \text{prop}(\texttt{Wedge}(A, B)) &= (\text{prop}(A)) \wedge (\text{prop}(B)) && \text{for any } A, B \in \textbf{Prop} \end{aligned}$$

where we drop the $(\dots)$ around prop(A) and prop(B) if they are simply atomic proposition. For example:

$$\begin{aligned} \text{prop}(\texttt{Wedge}(\texttt{Atomic}(p), \texttt{Wedge}(\texttt{Atomic}(q), \texttt{Atomic}(r)))) &= p \wedge (q \wedge r) \qquad \text{and} \\ \text{prop}(\texttt{Wedge}(\texttt{Wedge}(\texttt{Atomic}(p), \texttt{Atomic}(q)), \texttt{Atomic}(r))) &= (p \wedge q) \wedge r \end{aligned}$$

Note how the two different parse trees translate into syntactically-different[1] propositions. However, we know that the two propositions are actually equivalent, $p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r$, by associativity. Our ultimate goal in this problem is to prove the equivalence of all propositions that differ only in the placement of their parentheses, a fact which we cited earlier in the quarter but have not proven... until now.

To that end, we define a function "left" that takes a single parse tree and returns another one containing the same atomic propositions, in the same order, but with all the parentheses grouped to the left — i.e., something like $((((((p \wedge q) \wedge r) \wedge s) \wedge t) \wedge u) \wedge v) \wedge w$. It is defined as follows:

$$\begin{aligned} \text{left}(\texttt{Atomic}(p)) &= \texttt{Atomic}(p) \qquad \text{for any } p \in \mathcal{A} \\ \text{left}(\texttt{Wedge}(A, B)) &= \text{tilt}(\text{left}(A), B) \end{aligned}$$

For atomic propositions, left has no work to do (no parens to move). When given a $\texttt{Wedge}(A, B)$, left fixes the parentheses in the left subtree, A, recursively, but to add all the propositions from B into the expression with the parentheses grouped to the left, it relies on a second function, "tilt", which we define like this:

$$\begin{aligned} \text{tilt}(A, \texttt{Atomic}(p)) &= \texttt{Wedge}(A, \texttt{Atomic}(p)) \qquad \text{for any } A \in \textbf{Prop} \text{ and } p \in \mathcal{A} \\ \text{tilt}(A, \texttt{Wedge}(B, C)) &= \text{tilt}(\text{tilt}(A, B), C) \qquad \text{for any } A \in \textbf{Prop} \end{aligned}$$

Since it does need to change the first argument, A, which is already fixed by left, tilt is defined using the structure of the *second* argument only. We will give more intuition for how tilt works below. But first....

(a) [3 Points] Compute prop(left($\texttt{Wedge}(\texttt{Wedge}(\texttt{Atomic}(p), \texttt{Atomic}(q)), \texttt{Wedge}(\texttt{Atomic}(r), \texttt{Atomic}(s)))$)) from the definitions. Show your work.

(b) [3 Points] Compute prop(left($\texttt{Wedge}(\texttt{Atomic}(p), \texttt{Wedge}(\texttt{Atomic}(q), \texttt{Wedge}(\texttt{Atomic}(r), \texttt{Atomic}(s))))$)) from the definitions. Show your work.

(c) [2 Points] How do the resulting propositions compare (syntactically) if we apply left (and then prop) to the parse trees corresponding to the propositions $(p \wedge q) \wedge (r \wedge s)$ and $p \wedge (q \wedge (r \wedge s))$?
Hint: Use the results of parts (a) and (b). No additional computation is required.

From the definitions, we can see that prop(tilt$(A, \texttt{Atomic}(p))$) = prop($\texttt{Wedge}(A, \texttt{Atomic}(p))$) = (prop$(A)$) $\wedge\, p$, so when the second argument to tilt is an atomic proposition, p, tilt simply places this after prop(A) with parentheses on the left side (like we want).

When given a non-atomic proposition, $\texttt{Wedge}(A, B)$, the definition says that tilt$(A, \texttt{Wedge}(B, C))$ = tilt(tilt$(A, B), C)$. As tilt recurses over the parse tree, it will add each of the atomic propositions it finds in the leaves after prop(A) in the order they are reached by the recursion, and the definition, tilt(tilt$(A, B), C)$, says to *first* add all the atomic propositions found in B after prop(A) and *then* add all the atomic propositions found in C after all of those. Hence, the result will be prop(A) followed by all the propositions from $\texttt{Wedge}(B, C)$ *in the same order* as they appear in prop($\texttt{Wedge}(B, C)$), with parentheses around the left side at each step... just as we wanted.

Of course, the statements above, while intended to be helpful, are not a formal proof of correctness. Next, we ask you to formally prove some useful facts about these functions, facts that will help us work toward our goal of proving that the locations of parentheses in these propositional logic expressions are not important.

[1] I.e., they are different strings (even ignoring any differences in whitespace) due to the different placement of parentheses.

(d) [16 Points] Consider this claim: for any $A, B \in \textbf{Prop}$, we have $\text{prop}(\text{tilt}(A, B)) \equiv \text{prop}(\texttt{Wedge}(A, B))$. Prove this claim by structural induction **on** B.
Hint: For a fixed B, this is a "for all" claim over $A \in \textbf{Prop}$.
Hint: Your solution need not be long. My inductive step is a chain of less than 10 equivalences.

(e) [14 Points] Use structural induction to prove that $\text{prop}(\text{left}(A)) \equiv \text{prop}(A)$.
Hint: Use the fact proven in part (d). Again, your solution need not be long.

Finally, we return to the question we set out to address in this problem.

(f) [2 Points] Why does this tell us that the location of the parentheses are not important in any expression with just "$\wedge$"s and atomic propositions?

6. Extra Credit: Ambiguity (0 points)

Consider the following context-free grammar.

$$\begin{array}{ll}
\langle \textbf{Stmt} \rangle & \to \langle \textbf{Assign} \rangle \mid \langle \textbf{IfThen} \rangle \mid \langle \textbf{IfThenElse} \rangle \mid \langle \textbf{BeginEnd} \rangle \\
\langle \textbf{IfThen} \rangle & \to \text{if condition then } \langle \textbf{Stmt} \rangle \\
\langle \textbf{IfThenElse} \rangle & \to \text{if condition then } \langle \textbf{Stmt} \rangle \text{ else } \langle \textbf{Stmt} \rangle \\
\langle \textbf{BeginEnd} \rangle & \to \text{begin } \langle \textbf{StmtList} \rangle \text{ end} \\
\langle \textbf{StmtList} \rangle & \to \langle \textbf{StmtList} \rangle \langle \textbf{Stmt} \rangle \mid \langle \textbf{Stmt} \rangle \\
\langle \textbf{Assign} \rangle & \to \texttt{a := 1}
\end{array}$$

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) [0 Points] 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) [0 Points] Give **two different grammars** for this language that are both unambiguous but produce different parse trees from each other.