CSE 311: Foundations of Computing I

Section 8: Structural Induction and REs

1. Structural Induction I

Consider the following recursive definition of strings Σ^* over the alphabet Σ.

Basis Step: ε is a string

Recursive Step: If w is a string and $a \in \Sigma$ is a character, then wa is a string.

Recall the following recursive definition of the function len:

$$\begin{aligned} \mathsf{len}(\varepsilon) &= 0 \\ \mathsf{len}(wa) &= 1 + \mathsf{len}(w) \end{aligned}$$

Now, consider the following recursive definition:

$$\begin{aligned} \mathsf{double}(\varepsilon) &= \varepsilon \\ \mathsf{double}(wa) &= \mathsf{double}(w)aa. \end{aligned}$$

Prove that, for any string x, we have $\mathsf{len}(\mathsf{double}(x)) = 2\,\mathsf{len}(x)$.

2. Structural Induction II

Consider the following definition of a (binary) **Tree**:

Basis Step: $\bullet$ is a **Tree**.

Recursive Step: If L is a **Tree** and R is a **Tree** then $\texttt{Tree}(\bullet, L, R)$ is a **Tree**.

The function leaves returns the number of leaves of a **Tree**. It is defined as follows:

$$\begin{aligned} \mathsf{leaves}(\bullet) &= 1 \\ \mathsf{leaves}(\texttt{Tree}(\bullet, L, R)) &= \mathsf{leaves}(L) + \mathsf{leaves}(R) \end{aligned}$$

Also, recall the definition of size on trees:

$$\begin{aligned} \mathsf{size}(\bullet) &= 1 \\ \mathsf{size}(\texttt{Tree}(\bullet, L, R)) &= 1 + \mathsf{size}(L) + \mathsf{size}(R) \end{aligned}$$

Prove that $\mathsf{leaves}(T) \geq \mathsf{size}(T)/2$ for all $T \in$ **Trees**.

3. Regular Expressions

(a) Write a regular expression that matches base 10 non-negative numbers.
(Note that there should be no leading zeroes.)

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

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

4. CFGs

Construct CFGs for the following languages:

(a) All binary strings that end in 00.

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

(c) Propositional logic statements using only variables from a fixed alphabet $\mathcal{A} = \{\ldots, p, q, r, \ldots\}$ and only the operators $\neg$, $\wedge$, and $\vee$ as well as parentheses "(..)". (Assume no space characters.)

5. Structural Induction III

In this problem, we will prove De Morgan's Law for arbitrary propositions. For example, we will show that

$$\neg(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \equiv \neg p_1 \vee \neg p_2 \vee \cdots \vee \neg p_n.$$

is true for any $n \geq 1$.

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

Basis Elements For any $p \in \mathcal{A}$, $\texttt{Atomic}(p) \in \mathbf{Prop}$.

Recursive Step If $A, B \in \mathbf{Prop}$, then $\mathsf{Neg}(A), \mathsf{Wedge}(A, B), \mathsf{Vee}(A, B) \in \mathbf{Prop}$.

The set $\mathbf{Prop}$ represents parse trees of propositions. We allow the propositions to be combined using the operators, Wedge and Vee (the names of $\wedge$ and $\vee$ in LaTeX). We also allow negation of propositions with Neg.

Next, we define a function $\mathcal{T}$ that takes a parse tree (an element of $\mathbf{Prop}$) as input and returns the proposition that it represents.. Formally we define,

$$\begin{aligned}
\mathcal{T}(\texttt{Atomic}(p)) &= p && \text{for any } p \in \mathcal{A} \\
\mathcal{T}(\texttt{Wedge}(A, B)) &= (\mathcal{T}(A)) \wedge (\mathcal{T}(B)) && \text{for any } A, B \in \mathbf{Prop} \\
\mathcal{T}(\texttt{Vee}(A, B)) &= (\mathcal{T}(A)) \vee (\mathcal{T}(B)) && \text{for any } A, B \in \mathbf{Prop} \\
\mathcal{T}(\texttt{Neg}(A)) &= \neg\mathcal{T}(A) && \text{for any } A \in \mathbf{Prop}
\end{aligned}$$

The function flip takes a parse tree as input and returns another parse tree as follows:

$$\begin{aligned}
\mathsf{flip}(\mathsf{Atomic}(p)) &= \mathsf{Neg}(\mathsf{Atomic}(p)) && \text{for any } p \in \mathcal{A} \\
\mathsf{flip}(\mathsf{Wedge}(A, B)) &= \mathsf{Vee}(\mathsf{flip}(A), \mathsf{flip}(B)) && \text{for any } A, B \in \mathbf{Prop} \\
\mathsf{flip}(\mathsf{Vee}(A, B)) &= \mathsf{Wedge}(\mathsf{flip}(A), \mathsf{flip}(B)) && \text{for any } A, B \in \mathbf{Prop} \\
\mathsf{flip}(\mathsf{Neg}(A)) &= A && \text{for any } A \in \mathbf{Prop}
\end{aligned}$$

The function flip negates each atomic proposition and swaps $\vee$ with $\wedge$ (and vice versa) throughout the tree.

With those definitions in hand, use structural induction show that, for any $A \in \mathbf{Prop}$,

$$\mathcal{T}(\mathsf{Neg}(A)) \equiv \mathcal{T}(\mathsf{flip}(A)).$$

This proves that we can produce a proposition that is equivalent to negating the expression by, instead, flipping all $\wedge$s to $\vee$s (and vice versa) and negating atomic propositions recursively until we hit $\neg$s.