0. Structural Induction: CharTrees

Recursive Definition of CharTrees:

- Basis Step: `Null` is a CharTree.
- Recursive Step: If `L, R` are CharTrees and `c ∈ Σ`, then CharTree(`L, c, R`) is also a CharTree.

Intuitively, a CharTree is a tree where the non-null nodes store a char data element.

Recursive functions on CharTrees:

- The preorder function returns the preorder traversal of all elements in a CharTree.
  
  \[
  \text{preorder}(\text{Null}) = \varepsilon \\
  \text{preorder}(\text{CharTree}(L, c, R)) = c \cdot \text{preorder}(L) \cdot \text{preorder}(R)
  \]

- The postorder function returns the postorder traversal of all elements in a CharTree.
  
  \[
  \text{postorder}(\text{Null}) = \varepsilon \\
  \text{postorder}(\text{CharTree}(L, c, R)) = \text{postorder}(L) \cdot \text{postorder}(R) \cdot c
  \]

- The mirror function produces the mirror image of a CharTree.
  
  \[
  \text{mirror}(\text{Null}) = \text{Null} \\
  \text{mirror}(\text{CharTree}(L, c, R)) = \text{CharTree}(\text{mirror}(R), c, \text{mirror}(L))
  \]

- Finally, for all strings `x`, let the “reversal” of `x` (in symbols `x^R`) produce the string in reverse order.

Additional Facts:
You may use the following facts:

- For any strings `x_1, ..., x_k`: `(x_1 \cdot ... \cdot x_k)^R = x_k^R \cdot ... \cdot x_1^R`
- For any character `c`, `c^R = c`

Statement to Prove:
Show that for every CharTree `T`, the reversal of the preorder traversal of `T` is the same as the postorder traversal of the mirror of `T`. In notation, you should prove that for every CharTree, `T`: `[\text{preorder}(T)]^R = \text{postorder}(\text{mirror}(T))`.

There is an example and space to work on the next page.
Example for Intuition:

Let $T_i$ be the tree above.

$\text{preorder}(T_i) = \text{"abcd"}.$

$T_i$ is built as $(\text{null}, a, U)$

Where $U$ is $(V, b, W)$,

$V = (\text{null}, c, \text{null})$, $W = (\text{null}, d, \text{null}).$

This tree is mirror($T_i$).

$\text{postorder}($mirror$(T_i)) = \text{"dcba"},$

"dcba" is the reversal of "abcd" so $[\text{preorder}(T_i)]^R = \text{postorder}($mirror$(T_i))$ holds for $T_i$

Solution:

Let $P(T)$ be “$[\text{preorder}(T)]^R = \text{postorder}($mirror$(T))$”. We show $P(T)$ holds for all CharTrees $T$ by structural induction.

**Base case** ($T = \text{null}$): $\text{preorder}(T)^R = \epsilon^R = \epsilon = \text{postorder}(\text{null}) = \text{postorder}($mirror$(\text{null}))$, so $P(\text{null})$ holds.

**Inductive hypothesis**: Suppose $P(L) \land P(R)$ for arbitrary CharTrees $L, R$.

**Inductive step**: We want to show $P(\text{CharTree}(L, c, R))$, i.e. $[\text{preorder}($CharTree$(L, c, R))]^R = \text{postorder}($mirror($\text{CharTree}(L, c, R))$).

Let $c$ be an arbitrary element in $\Sigma$, and let $T = \text{CharTree}(L, c, R)$

$$\text{preorder}(T)^R = [c \cdot \text{preorder}(L) \cdot \text{preorder}(R)]^R$$

$\text{preorder}(T)^R = [c \cdot \text{preorder}(L) \cdot \text{preorder}(R)]^R$ \hspace{1cm} \text{defn of preorder}

$\text{preorder}(T)^R = \text{preorder}(R)^R \cdot \text{preorder}(L)^R \cdot c^R$ \hspace{1cm} \text{Fact 1}

$\text{preorder}(T)^R = \text{preorder}(R)^R \cdot \text{preorder}(L)^R \cdot c$ \hspace{1cm} \text{Fact 2}

$\text{preorder}(T)^R = \text{postorder}(\text{mirror}(R)) \cdot \text{postorder}(\text{mirror}(L)) \cdot c$ \hspace{1cm} \text{by I.H.}

$\text{preorder}(T)^R = \text{postorder}(\text{CharTree}(\text{mirror}(R), c, \text{mirror}(L)))$ \hspace{1cm} \text{recursive defn of postorder}

$\text{preorder}(T)^R = \text{postorder}(\text{mirror}(\text{CharTree}(L, c, R)))$ \hspace{1cm} \text{recursive defn of mirror}

$\text{preorder}(T)^R = \text{postorder}(\text{mirror}(T))$ \hspace{1cm} \text{defn of $T$}

So $P(\text{CharTree}(L, c, R))$ holds.

By the principle of induction, $P(T)$ holds for all CharTrees $T$. 
1. Structural Induction: Strings

Recursive Definition of a String:
- Basis Step: $\epsilon$ is a string
- Recursive Step: If $w$ is a string and $a$ is a character, $w \cdot a$ is a string
  (the string $w$ with the character $a$ appended on to the end)

Recursive functions on String:
Length:
\[
\begin{align*}
\text{len}(\epsilon) &= 0 \\
\text{len}(w \cdot a) &= 1 + \text{len}(w)
\end{align*}
\]
Reverse:
\[
\begin{align*}
\text{rev}(\epsilon) &= \epsilon \\
\text{rev}(w \cdot a) &= a \cdot \text{rev}(w)
\end{align*}
\]

Statement to Prove:
Prove that for any string $x$, $\text{len} (\text{rev}(x)) = \text{len}(x)$.

Solution:
For a string $x$, let $P(x)$ be "$\text{len}(\text{rev}(x)) = \text{len}(x)$". We prove $P(x)$ for all strings $x$ by structural induction on the set of strings.

Base Case ($x = \epsilon$): By definition of reverse, $\text{len}(\text{rev}(\epsilon)) = \text{len}(\epsilon)$. So $P(\epsilon)$ holds.

Inductive Hypothesis: Suppose $P(w)$ holds for some arbitrary string $w$. Then $\text{len}(\text{rev}(w)) = \text{len}(w)$.

Inductive Step: Goal: Show that $P(w \cdot a)$ holds for any character $a$.

Let $a$ be an arbitrary character.
\[
\begin{align*}
\text{len}(\text{rev}(w \cdot a)) &= \text{len}(a \cdot \text{rev}(w)) \quad \text{[By Definition of reverse]} \\
&= 1 + \text{len}(\text{rev}(w)) \quad \text{[By Definition of length]} \\
&= 1 + \text{len}(w) \quad \text{[By IH]} \\
&= \text{len}(w \cdot a) \quad \text{[By Definition of length]}
\end{align*}
\]

This proves $P(w \cdot a)$.

Conclusion: $P(x)$ holds for all strings $x$ by structural induction.
2. Structural Induction: Dictionaries

Recursive definition of a Dictionary (i.e. a Map):

- Basis Case: [] is the empty dictionary
- Recursive Case: If D is a dictionary, and a and b are elements of the universe, then \((a \rightarrow b) :: D\) is a dictionary that maps \(a\) to \(b\) (in addition to the content of \(D\)).

Recursive functions on Dictionaries:

\[
\begin{align*}
\text{AllKeys}([]) &= [] & \text{len}([]) &= 0 \\
\text{AllKeys}((a \rightarrow b) :: D) &= a :: \text{AllKeys}(D) & \text{len}((a \rightarrow b) :: D) &= 1 + \text{len}(D)
\end{align*}
\]

Recursive functions on Sets:

\[
\begin{align*}
\text{len}([]) &= 0 \\
\text{len}(a :: C) &= 1 + \text{len}(C)
\end{align*}
\]

Statement to prove:
Prove that \(\text{len}(D) = \text{len}(\text{AllKeys}(D))\).

Solution:
Proof. Define \(P(D)\) to be \(\text{len}(D) = \text{len}(\text{AllKeys}(D))\) for a Dictionary \(D\). We will go by structural induction to show \(P(D)\) for all dictionaries \(D\).

Base Case: \(D = []\): Note that:

\[
\begin{align*}
\text{len}(D) &= \text{len}([]) \\
&= \text{len}(\text{AllKeys}([])) & \text{[Definition of AllKeys]} \\
&= \text{len}(\text{AllKeys}(D))
\end{align*}
\]

Inductive Hypothesis: Suppose \(P(C)\) to be true for an arbitrary dictionary \(C\).

Inductive Step:
Let \(D' = (a \rightarrow b) :: C\). Note that:

\[
\begin{align*}
\text{len}((a \rightarrow b) :: C) &= 1 + \text{len}(C) & \text{[Definition of Len]} \\
&= 1 + \text{len}(\text{AllKeys}(C)) & \text{[IH]} \\
&= \text{len}(a :: \text{AllKeys}(C)) & \text{[Definition of Len]} \\
&= \text{len}(\text{AllKeys}((a \rightarrow b) :: C)) & \text{[Definition of AllKeys]}
\end{align*}
\]

So \(P(D')\) holds.

Conclusion: Thus, the claim holds for all dictionaries \(D\) by structural induction. ∎
3. Structural Induction: CFGs

Consider the following CFG:

\[ S \rightarrow SS \mid 0S1 \mid 1S0 \mid \epsilon \]

Prove that every string generated by this CFG has an equal number of 1's and 0's.

**Hint:** You may wish to define the functions \( \#_0(x), \#_1(x) \) on a string \( x \).

**Solution:**
First we observe that the language defined by this CFG can be represented by a recursively defined set. Define a set \( S \) as follows:

**Basis Rule:** \( \epsilon \in S \)

**Recursive Rule:** If \( x, y \in S \), then \( 0x1, 1x0, xy \in S \).

Now we perform structural induction on the recursively defined set. Define the functions \( \#_0(t), \#_1(t) \) to be the number of 0's and 1's respectively in the string \( t \).

**Proof:** For a string \( t \), let \( P(t) \) be defined as "\( \#_0(t) = \#_1(t) \)". We will prove \( P(t) \) is true for all strings \( t \in S \) by structural induction.

**Base Case** \( (t = \epsilon) \): By definition, the empty string contains no characters, so \( \#_0(t) = 0 = \#_1(t) \)

**Inductive Hypothesis:** Suppose \( P(x), P(y) \) hold for some arbitrary strings \( x, y \).

**Inductive Step:**

**Case 1:** Goal is to show \( P(0x1) \) holds.
By the IH, \( \#_0(x) = \#_1(x) \). Then observe that:

\[ \#_0(0x1) = \#_0(x) + 1 = \#_1(x) + 1 = \#_1(0x1) \]

Therefore \( \#_0(0x1) = \#_1(0x1) \). This proves \( P(0x1) \).

**Case 2:** Goal is to show \( P(1x0) \) holds.
By the IH, \( \#_0(x) = \#_1(x) \). Then observe that:

\[ \#_0(1x0) = \#_0(x) + 1 = \#_1(x) + 1 = \#_1(1x0) \]

Therefore \( \#_0(1x0) = \#_1(1x0) \). This proves \( P(1x0) \).

**Case 3:** Goal is to show \( P(xy) \) holds.
By the IH, \( \#_0(x) = \#_1(x) \) and \( \#_0(y) = \#_1(y) \). Then observe that:

\[ \#_0(xy) = \#_0(x) + \#_0(y) = \#_1(x) + \#_1(y) = \#_1(xy) \]

Therefore \( \#_0(xy) = \#_1(xy) \). This proves \( P(xy) \).

So by structural induction, \( P(t) \) is true for all strings \( t \in S \). \( \square \)
4. Regular Expressions

(a) Consider the following Regular Expression (RegEx):

\[ 1(45 \cup 54)^*1 \]

List 5 strings accepted by the RegEx and 5 strings from \( T := \{1, 4, 5\}^* \) rejected by the RegEx. Then, summarize this RegEx in your own words.

**Solution:**

Accepted:
- 1451
- 1541
- 145541
- 1454545451
- 11

Rejected:
- 1
- 1441
- 45
- 14451
- 111

This RegEx accepts exactly those strings that start and end with a 1, and have one or more pairs of 45 or 54 in the middle.

(b) Consider the following Regular Expression (RegEx):

\[ 0^* (0 \cup 1)^* ((01) \cup (11) \cup (10) \cup (00)) 1^* (0 \cup 1)^* \]

List 3 strings accepted by the RegEx and 3 strings from \( S := \{0, 1\}^* \) rejected by the RegEx. Then, summarize this RegEx in your own words and write a simpler RegEx that accepts exactly the same set of strings.

**Solution:**

Accepted:
- 01
- 10
- 10100100101

Rejected:
- \( \epsilon \)
- 0
- 1

This RegEx accepts all binary strings that are 2 or more characters long. A simpler RegEx for this is \((0 \cup 1)(0 \cup 1)(0 \cup 1)^*\).
5. Constructing Regular Expressions

For each of the following, construct a regular expression for the specified language.

(a) Strings from the language $S := \{a\}^*$ with an even number of $a$'s.

Solution:

$(aa)^*$

(b) Strings from the language $S := \{a, b\}^*$ with an even number of $a$'s.

Solution:

$b^*(b^*ab^*ab^*)^*$

(c) Strings from the language $S := \{a, b\}^*$ with odd length.

Solution:

$(aa \cup ab \cup ba \cup bb)^*(a \cup b)$

(d) (Challenge) Strings from the language $S := \{a, b\}^*$ with an even number of $a$'s or an odd number of $b$'s.

Solution:

$b^*(b^*ab^*ab^*)^* \cup (a^* \cup a^*ba^*ba^*)^*b(a^* \cup a^*ba^*ba^*)^*$