Problem Set 8

Deliverable: Submit your responses as a single, readable PDF file on the collab site before **6:29pm** on **Friday, 4 November**.

Collaboration Policy - Read Carefully

The collaboration policy is identical to that for PS7: you should work in groups of *one* to *four* students of your choice with no restrictions, and follow the rest of the collaboration policy from PS3.

Preparation

This problem set focuses on recursive data types and structural induction — Chapter of the MCS book, and Class 16 and Class 17.

Directions

Problems 1–8 are expected for everyone; solve as many as you can. The Programming with Procedures problems are optional (see the note before them).

Tsilly Lists

Consider an alternate way of defining a list from the one we used in Class 16 and 17, where instead of *prepend*, lists are constructed using *postpend* (to avoid confusion, we call our postpended list a *tsil*, and reserve *list* for the original prepended list):

> A *tsil* is either the empty tsil (λ), or the result of postpend(t, e) for some tsil t and object e.

1. Define the meaning of the observer operations (similarly to the beginning of Class 17) for the tsil: last : $Tsil \to Object$, frest : $Tsil \to Tsil$, and empty : $Tsil \to Boolean$.
2. Provide a definition of *length* for the tsil type.
3. Prove that there is an equivalent tsil for every list. (Your answer should include a clear definition of what *equivalent* means.)

Structural Induction on Trees

In Class 17, we defined a binary tree as either **null** or a node constructed from a binary tree, object, and binary tree. Here, we define a binary tree where the labels are natural numbers.

A tree has these operations:

$$
\begin{aligned}
&\textbf{null} : \mathit{Tree}\\
&\text{node} : \mathit{Tree} \times \mathbb{N} \times \mathit{Tree} \to \mathit{Tree}\\
&\text{label} : \mathit{Tree} \to \mathbb{N}\\
&\text{left} : \mathit{Tree} \to \mathit{Tree}\\
&\text{right} : \mathit{Tree} \to \mathit{Tree}\\
&\text{empty} : \mathit{Tree} \to \{\mathbf{T}, \mathbf{F}\}
\end{aligned}
$$

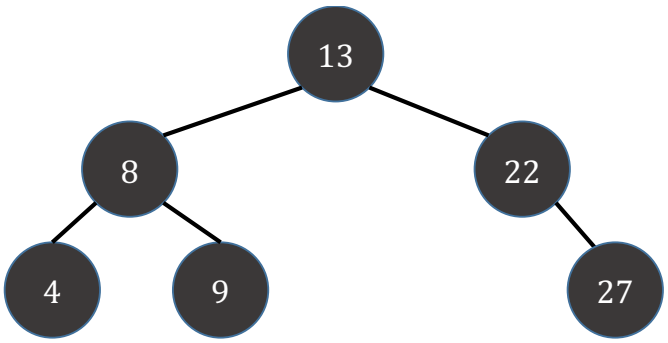
The meaning of the operations is defined for all trees t_1, t_2, and all $n \in \mathbb{N}$, by:

$$
\begin{aligned}
&\text{label}(\text{node}(t_1, n, t_2)) \to n\\
&\text{left}(\text{node}(t_1, n, t_2)) \to t_1\\
&\text{right}(\text{node}(t_1, n, t_2)) \to t_2\\
&\text{empty}(\textbf{null}) \to \mathbf{T}\\
&\text{empty}(\text{node}(t_1, n, t_2) \to \mathbf{F}
\end{aligned}
$$

4. The *height* of a tree is the maximum distance (number of edges) from its root (the one node that has no parent node) to a leaf. Provide a *constructive* definition of *height* for our binary tree type. (Hint: the height of node(**null**, n, **null**) is 0.)

5. Prove that the maximum number of nodes in a binary tree of height h is $2^{h+1} - 1$.

The in-order traversal of a binary tree is a list of the node labels in the order they appear from left-to-right across the tree. For example, the in-order traversal of the tree shown below would be the list $(4, 8, 9, 13, 22, 27)$.



We can define traverse to produce a (prepend) list as: (note the + operation here is list concatenation, as defined in Class 17)

$$\text{traverse}(\textbf{null}) = \textbf{null}$$
$$\text{traverse}(\text{node}(t_1, n, t_2)) = \text{traverse}(t_1) + \text{prepend}(n, \text{traverse}(t_2))$$

6. Prove that for all trees t with n nodes, the result of traverse(t) is a list of length n.

Ordered Binary Trees

We define an *OrderedBinaryTree* as:

– **Base case: null** $\in$ *OrderedBinaryTree*.

– **Constructor case:** if $t_1, t_2 \in$ *OrderedBinaryTree* and $n \in \mathbb{N}$, and maximum$(t_1) < n$ and minimum$(t_2) > n$, then node$(t_1, n, t_2) \in$ *OrderedBinaryTree*.

You may assume all the other tree operations (including traverse from question 5) are defined for *OrderedBinaryTree*s also.

7. Define the minimum : *OrderedBinaryTree* $\to \mathbb{N}$ and maximum : *OrderedBinaryTree* $\to \mathbb{N}$ operations used in the definition of *OrderedBinaryTree* above.

8. ($\star$) Prove that $\forall t \in$ *OrderedBinaryTree*. traverse(t) is an ordered list. (A list, $p = (p_1, p_2, \cdots, p_n)$ is an ordered list if $\forall i \in \{1, \cdots, n-1\}. p_i < p_{i+1}$.)

Programming with Procedures

> These problems are *optional*, and provided to give students who are interested some experience with functional programming which will make you a more powerful, fashionable, and prolific programmer. You do not need to do them to earn "gold star" level credit on this assignment, and nothing on the exams will depend on them. You will receive "bonus" credit on this assignment for turning in good answers to these questions.

These questions assume you have some experience programming in Python, but are sadly lacking in previous experience using procedures as parameters and results and realize that you cannot be a true kunoichi programmer without becoming adept with programming with procedures.

Download: pairs.py

In pairs.py, we defined `make_pair` and various procedures for building and using lists. You should download this code, run it in your favorite Python3 environment, and make sure you understand it.

9. Define a function `list_tostring(lst)` that takes a list (constructed using the `list_append` function from `pairs.py`) as its input and returns a string representation of that list. For example, `list_tostring(list_prepend(1, list_prepend(2, list_prepend(3, None))))` should print out `[1, 2, 3]`. (You can use `str(x)` to turn any Python object `x` into a string.)

10. Define a function `list_map(fn, lst)` that takes as inputs a function and a list, and returns a list that is the result of applying the input function to each element of `lst`. For example, `list_map(lambda x:  x + 1, list123)` should return the list `[2, 3, 4]`.

11. Define a function `list_accumulate(fn, lst, base)` that takes as inputs a function, a list, and a base value, and returns the result of applying the function through the list. For example, `list_accumulate(lambda a, b:  a + b, lst, 0)` should return the sum of all the elements in the list, and `list_accumulate(lambda a, b:  a * b, lst, 1)` should return their product, and `list_accumulate(lambda a, b:  a + 1, lst, 0)` should return the length of the list.

12. Define `list_map` (as in problem 10) using `list_accumulate`.

