Homework 2
SNU 4910.210 Fall 2012

Chung-Kil Hur

due: 10/02 (Thu) 24:00

The objectives of this homework are :

- to learn how to write recursive functions
- to learn how to make recursively-defined data
- to learn how to write programs with types in mind

Exercise 1 "Tree-type data"

Tree-type data are commonly used in computer science.

Trees are defined as follows:

- Base case: a leaf is a tree.
- Inductive case: A node with one or more branches that contain sub-trees is a tree.

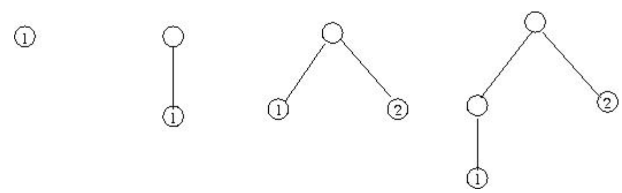
The base case is a way to make basic trees, called leaf trees, and the inductive case is a way to make a new tree using already made trees.

Write the following two functions that make trees:

$$\texttt{leaf} : \tau \to \tau \ \mathit{tree}$$
$$\texttt{node} : (\tau \ \mathit{tree}) \ \mathit{list} \to \tau \ \mathit{tree}$$

`leaf` takes a value of type τ and evaluates to the leaf tree with the value. For example, `(leaf 1)` evaluates to the leaf tree with 1 and `(leaf '(1 2))` to that with (1 2).

As further examples, the following trees are `(leaf 1)`, `(node (list (leaf 1)))`, `(node (list (leaf 1) (leaf 2)))`, and `(node (list (node (list (leaf 1))) (leaf 2)))`, respectively.



Write the following three functions for trees:

$$\texttt{is-leaf?} : \tau\ \mathit{tree} \rightarrow \mathit{bool}$$
$$\texttt{leaf-val} : \tau\ \mathit{tree} \rightarrow \tau$$
$$\texttt{nth-child} : \tau\ \mathit{tree} \times \mathit{nat} \rightarrow \tau\ \mathit{tree}$$

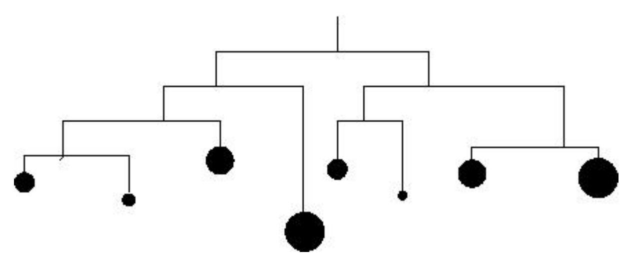
`is-leaf?` determines whether a given tree is a leaf tree or not. `leaf-val`, if given a leaf tree, evaluates to the value contained in it. `nth-child`, if given a non-leaf tree and $n \geq 0$, evaluates to its n-th subtree. Note that the 0-th, the 1-th, ... subtrees denote the first, the second, ... subtrees. □

Exercise 2 "Weighing a mobile"

Imagine a mobile hanging from the ceiling. Usual (binary) mobiles can be defined as follows:

- Base case: An object is a mobile.
- Inductive case: A node with two branches that contain sub-mobiles at certain distances from the node is a mobile.

The following is an example of mobile:



Write the following three functions that make mobiles. You should use the functions that you defined in Exercise 1.

$$\texttt{model} : \mathit{nat} \rightarrow \mathit{mobile}$$
$$\texttt{make-branch} : \mathit{nat} \times \mathit{mobile} \rightarrow \mathit{branch}$$
$$\texttt{make-mobile} : \mathit{branch} \times \mathit{branch} \rightarrow \mathit{mobile}$$

`model` takes a natural number n and makes a single-object mobile with weight n. `make-branch`, given a natural number d and a mobile m, makes a branch that contains the mobile m at distance d from the center. `make-mobile` takes two branches and makes a mobile with the branches.

Write the following two functions for mobiles:

`is-balanced?` : *mobile* $\rightarrow$ *bool*
`weight` : *mobile* $\rightarrow$ *nat*

`is-balanced?` checks whether a given mobile is balenced. A mobile is balanced if either it is a single object, or all sub-mobiles of it have balenced branches. Two branches are balenced if they have the same torque (*i.e.*, the distance $\times$ the weight). `weight`, given a mobile, calculates the sum of weights of all objects in the mobile. $\Box$

Exercise 3 "Boolean Circuit"

Boolean circuits are inducitvely defined as follows. Every circuit has a single ouput.

- Base case: A wire with output 0 is a boolean circuit.
- Base case: A wire with output 1 is a boolean circuit.
- Inductive case: Appending `not` to a boolean circuit makes a boolean circut.
- Inductive case: Connecting two boolean circuits with `and` makes a boolean circuit.
- Inductive case: Connecting two boolean circuits with `or` makes a boolean circuit.

Define the above five ways to make boolean circuits:

`zero` : *circuit*
`one` : *circuit*
`not-circuit` : *circuit* $\rightarrow$ *circuit*
`and-circuit` : *circuit* $\times$ *circuit* $\rightarrow$ *circuit*
`or-circuit` : *circuit* $\times$ *circuit* $\rightarrow$ *circuit*

You should use the functions that you defined in Exercise 1.

And, write the following six functions for boolean circuits:

$$\begin{aligned}
&\texttt{is-zero?} : \textit{circuit} \to \textit{bool} \\
&\texttt{is-one?} : \textit{circuit} \to \textit{bool} \\
&\texttt{is-not?} : \textit{circuit} \to \textit{bool} \\
&\texttt{is-and?} : \textit{circuit} \to \textit{bool} \\
&\texttt{is-or?} : \textit{circuit} \to \textit{bool} \\
&\texttt{sub-circuit} : \textit{circuit} \times \textit{nat} \to \textit{circuit}
\end{aligned}$$

`sub-circuit` takes a circuit and a natural number $0 \le n \le 1$ and returns the n-th sub-circuit. □

Exercise 4 "Computatino of a boolean circuit"

Write a function that computes the result of a boolean circuit.

$$\texttt{output} : \textit{circuit} \to \{0, 1\}$$

The result of a boolean circuit is recursively defined as follows. The result of `zero` is 0. The result of `one` is 1. The result of `(not` B`)` is 0 if that of B is 1; 1, otherwise. The result of `(and` B_1 B_2`)` is 1 if both B_1 and B_2 result in 1; 0, otherwise. The result of `(or` B_1 B_2`)` is 0 if both B_1 and B_2 result in 0; 1, otherwise. □

Exercise 5 "Manual Type Checking"

Type check the following three functions that are given in the lecture slides.

map-reduce : $(\tau \to \gamma)$ * τ list * $(\gamma$ * $\sigma \to \sigma)$ * $\sigma \to \sigma$
map : $(\tau \to \gamma)$ * τ list $\to \gamma$ list
reduce : γ list * $(\gamma$ * $\sigma \to \sigma)$ * $\sigma \to \sigma$