CS 496: Homework Assignment 2
Due: 3 February, 11:55pm

1 Assignment Policies

Collaboration Policy. Homework will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions.

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be re-used. Violations will be penalized appropriately.

2 Assignment

This assignment has two sections. The first one addresses the definition of a type `dTree` encoding *binary decision trees* in Scheme using variant records and then defining some simple operations on them. Two examples of such trees are given in Fig. 1. The second section presents a small application using `dTree`s which you are requested to implement.

3 dTree in Scheme

1. Define `dTree`, a user-defined type in Scheme which encodes *binary decision trees* as depicted in Fig 1 via variant records (i.e. using the `define-datatype`). The names of the two constructors should be `leaf-t` and `node-t`.

2. Define two expressions of type `dTree` that represent each of the two trees in Fig. 1.

3. Implement the following functions indicating, for each one, its type. In the examples below, we use `tLeft` to denote the example tree (Fig. 1–left) encoded as a value of type `dTree` and similarly for `tRight`.

   (a) `dTree-height`: that given a `dTree` returns its height. Eg.

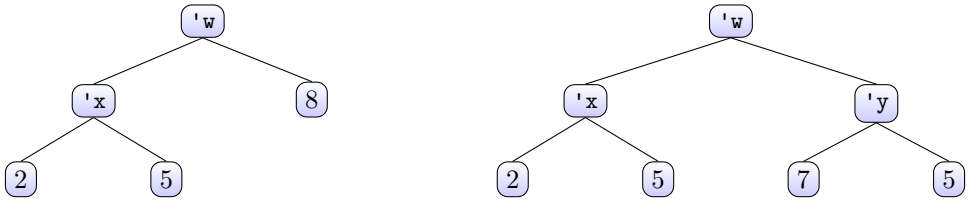


Figure 1: Sample *DTrees*

```
> (dTree-height tLeft)
2
```

(b) `dTree-size`: that given a `dTree` returns its size. The size of a `dTree` consists of the number of nodes and leaves.

```
> (dTree-size tLeft)
5
```

(c) `dTree-paths`: that given a `dTree` returns a list with all the paths to its leaves. A path is a list of digits in the set $\{0, 1\}$ such that if we follow it on the tree, it leads to a leaf. The order in which the paths are listed is irrelevant. Eg.

```
> (dTree-paths tLeft)
((0 0) (0 1) (1))
```

(d) `dTree-perfect?`: that determines whether a `dTree` is *perfect*. A `dTree` is said to be *perfect* if all leaves have the same depth. Eg.

```
> (dTree-perfect? tLeft)
#f
> (dTree-perfect? tRight)
#t
```

(e) `dTree-map`: that given the following arguments

a function `f: symbol? -> symbol?`
a function `g: number? -> number?`
a `dTree t`

returns a new `dTree` resulting from `t` by applying `f` to the symbols in each node and `g` to the numbers in each leaf. In order to give an example, suppose we have the following function which converts symbols to uppercase:

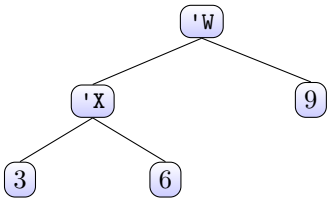
```
;; symbol? -> symbol?
(define symbol-upcase
     (compose string->symbol (compose string-upcase
   symbol->string)))
```

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

```
'( (x y z) .
 (((0  0  0) . 0)
  ((0  0  1) . 1)
  ((0  1  0) . 1)
  ((0  1  1) . 0)
  ((1  0  0) . 1)
  ((1  0  1) . 0)
  ((1  1  0) . 0)
  ((1  1  1) . 1)
))
```

Figure 2: Binary function and its encoding in Scheme

Then `(dTree-map symbol-upcase succ tLeft)` should return an expression of type `dTree` representing the tree

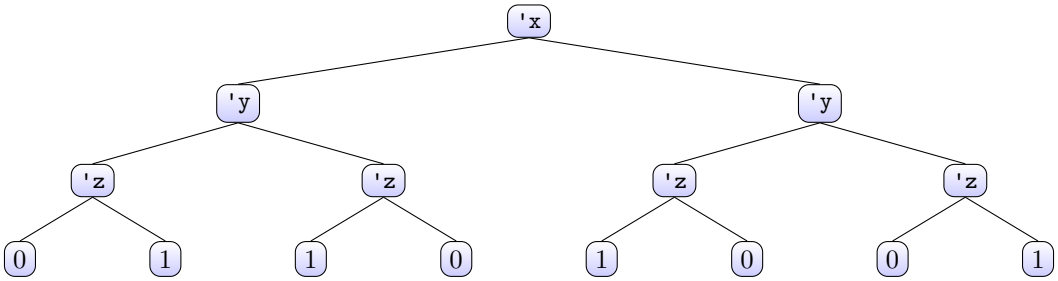


4 An Application of dTrees: Encoding Boolean Functions

A *boolean function* is a function from booleans to booleans. An example is given on the left in Fig. 2. There are many ways to encode boolean functions in Scheme; in this exercise we consider two of them:

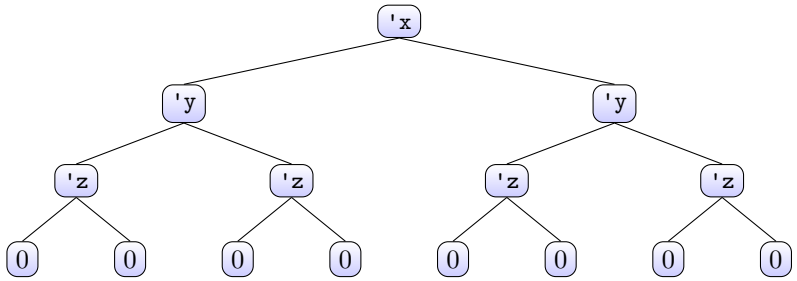
- *pair-encoding*
- *tree-encoding*

Let us first consider the *pair-encoding*. The *pair-encoding* consists of a pair where the first component is the list of its parameters and the second is its graph. For example, the *pair-encoding* of f is indicated in Fig. 2 on the right. The *tree-encoding* of a boolean function consists of a *DTree* in which each value of the function is given by a path in the tree (the leaf at the end of the path). For example, the tree-encoding of f is:



The aim of this exercise is to a function `bf->dTree` that takes a pair-encoding of a boolean function and returns its tree-encoding.

4. Define `list->tree`, a function that given a list of symbols `l`, creates a tree in which the symbols of an inner node at level n corresponds to the n-th element in `l`. The value of all of its leaves may be set to 0. E.g. `(list-> tree '(x y z))` produces the `dTree` representing:



5. Define `replaceLeafAt`, a function that given a tree `t` and a graph for a function `f`, replaces all the leaves in `t` by the value indicated in the graph of the function.

6. Define a function `bf->dTree` that takes a pair-encoding of a boolean function and returns its tree-encoding. Note that, depending on the order in which the formal parameters are processed, there may be more than one possible tree-encoding for a given pair-encoding. You may return any of them.

5 Submission instructions

Submit a file `hw2_SURNAME.rkt` through Canvas, where `SURNAME` is replaced by your surname. Your grade will be determined as follows:

Exercise	Grade
1	1
2	1
3(a)-(e)	5 (1 each)
4	1
5	1
6	1