Mutually Recursive Data

Programming Language ISL+

Purpose To process mutually recursive data.

Setup If two data definitions refer to each other, so should their templates.

Quote can be used a shorthand to create lists of atomic data very easily. For example, `'((+ 1 2 #f) meow ("cat" 3 (4 5)))` is the same as `(list (list '+ 1 2 #false) 'meow (list "cat" 3 (list 4 5)))`.

Exercises

Exercise 1

```
; An AExp (Arithmetic Expression) is one of:
; - Number
; - (cons Operation LoA)

; An LoA (List of Arithmetic Expression) is one of:
; - '()
; - (cons AExp LoA)

; An Operation is one of:
; - '+
; - '*
```

- Define the template for an arithmetic expressions and sufficient examples to test on.
- Design a function that consumes an AExp and evaluates it to a single number.

Exercise 2

```
; An AlExp (Algebraic Expression) is one of:
; - Number
; - Symbol (indicating a variable)
; - (cons Operation LoAl)

; An LoAl (List of Algebraic Expression) is one of:
; - '()
; - (cons AlExp LoAl)

; An Assignment is a (list Symbol Number)
```

- Define the template for an algebraic expression and sufficient examples to test on.
- Design a function that determines if an algebraic expression is an instance of an arithmetic expression.
- Design a function that consumes two algebraic expressions and determines if they are exactly equal. Do not use `equal?`.
- Design a function that given an algebraic expression and a list of assignments, produces that algebraic expression where all of the variables that have been assigned a number have been replaced with the appropriate number. You can assume the list of assignments don't contain duplicate symbols.

  Hint: Design a funtion which substites all instances of a single variable for a single number in a AlExp, and then use `foldr` to do this with a list of assignments.

The following exercise is challenging and optional.

Exercise 3 Design a function that determines if two **arithmetic** expressions are equal up to reordering. Two arithmetic expressions are equal up to reordering if they are the same number or if they are applying the same operation to equal arithmetic expressions up to reordering in any order.

Here are some instances of arithmetic expressions that are equal up to ordering:

- `5` and `5`
- `'(+ 1 2)` and `'(+ 2 1)`
- `'(+ 2 (* 3 4) 2)` and `'(+ 2 2 (* 4 3))`

Here are some instances of arithmetic expressions that are not equal up to reordering:

- `5` and `3`
- `'(+ 1 2)` and `3`
- `'(+ 1 2)` and `'(* 2 1)`
- `'(+ 1 2)` and `'(+ 2 2)`
- `'(+ 2 (* 3 4) 2)` and `'(+ 2 (* 3 4) (* 4 3))`
- `'(+ 1 (+ 2 3))` and `'(+ 1 2 3)`