Assignment 1

Exercise 2

Note: Throughout this exercise we have assumed that a list l_1 can be split only if $|l_1| \geq 2$, i.e. cases like $g(src(l_1), src([]))$ shall never occur and only calls to g like $g(src(l_1), src(l_2))$ with non-empty lists l_1 and l_2 are valid calls.

(i)

From the definition of $src[e, f, g]$, we see that for a non empty list $l_1 = \{x_1, \dots, x_n\}$,

e has type Q

f takes input x with type T, which is the type of elements in $reprarray$ And f is applied to all elements in the list l_1, and returns a type Q variable.

g takes input (λ_1, λ_2) with type $Q \times Q$ and returns type Q.

To make sure no ambiguity exists, the any list l that can be written in two sublist and the two sublist are defined by

- $l_1 = l[1 : l.getlength()/2 - 1]$
- $l_2 = l[l.getlength()/2 : l.getlength()]$

To ensure there's no ambiguity in the operation, function src must be $surjective$ (i.e. f and g must be $surjective$) , which requires the g to be associative with e, i.e. $g(src(e, f, g)(l_1), e) = g(e, src(e, f, g)(l_1)$(though g is assumed to never be called with parameter e)

(ii)

- *Determining the length (if not stored)*

$$\begin{cases} e := 0 \\ f(x) := 1 \\ g(\lambda_1, \lambda_2) := \lambda_1 + \lambda_2 \end{cases}$$

- *Applying a function to all elements of a list*

  Let h be the specified operation on elements of the given list.

$$\begin{cases} e := [] \\ f(x) := [h(x)] \\ g(l_1, l_2) := concat(l_1, l_2) \end{cases}$$

- *Creating a sublist of list elements satisfying a condition φ*

  Let φ be the condition that a element needs to satisfy.

$$\begin{cases} e := [] \\ f(x) := I[x].\, \varphi(x) \\ g(l_1, l_2) := l_1 \cup l_2 \end{cases}$$

  Note: f(x) returns an empty list if x does not satisfy φ, and returns a list containing x if x satisfies φ

(iii)

Let the length of the given list be n. In the process, f is applied n times and g is applied $n-1$ times.

For the complexity of f, assume the operation of f take t_f steps, then the complexity of f amounts to $O(t_f n)$.

As for g, there are two cases:

- If we consider data types where when splitting a list into sublist, no allocation and copying is necessary, then assume the cost of splitting to be some constant c, then the complexity of g amounts to $O(c(n-1)) = O(n)$.

  And the total complexity of src amounts to $O(mn + n) = O(mn)$.

- If we consider data types where when splitting a list, extra actions are needed (see my implementation of src on AList), then the complexity of g becomes different: In case of AList, each time a list is split, src have to copy all elements of the original list into its sublist. Let the cost of copying be some constant c, then the complexity of g amounts to $O(c * n * \lfloor \log_2(n) \rfloor) = O(n * \log_2(n))$

  And the total complexity of src amounts to $O(mn + n \log_2(n))$

(iv)

C++ code implementation are in the folder. After complication, the executable you get provides 2 test cases:

1. Generate the length of an auto-generated AList. SIZE of the list is defined using macro definition and the list is filled to natural numbers. The default setting has

```
#define SIZE 20
```

2. Using the same testing list, the test cases defines e, f, g such that the src function returns a sublist with all elements ≥ 14