TDT4165 Programming Languages Assignment 3

Martin Hegnum Johannessen

October 1, 2024

Task: 1 Quadratic equation solver

Task: 1.1 Implementation of `proc {QuadraticEquation A B C ?RealSol ?X1 ?X2}`

The procedure solves the quadratic equation $Ax^2 + Bx + C = 0$ using the quadratic formula:

$$x = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

It first computes the discriminant $\Delta = B^2 - 4AC$. If Δ is negative, it sets `RealSol` to `false`, indicating no real solutions. Otherwise, it calculates the two real roots $X1$ and $X2$. The implementation is presented in Figure 1.

Task: 1.2 Test Cases

- For $A = 2, B = 1, C = -1$:
  Discriminant: $\Delta = 9$
  Roots: $X1 = -1.0, X2 = 0.5$
  `RealSol` is `true`.
- For $A = 2, B = 1, C = 2$:
  Discriminant: $\Delta = -15$
  No real solutions. `RealSol` is `false`.

Task: 1.3 Why are procedural abstractions useful?

Procedural abstractions allow encapsulation of functionality into reusable code blocks, making programs modular, easier to maintain, and more readable.

Task: 1.4 Difference between a procedure and a function

- A **procedure** performs actions and may have side effects but doesn't return a value.
- A **function** computes and returns a value as its main goal.

Task: 2 Sum Function

The `Sum` function recursively calculates the sum of a list by adding the head element to the result of summing the tail. If the list is empty, it returns 0. Figure 2 presents an example call of the function.

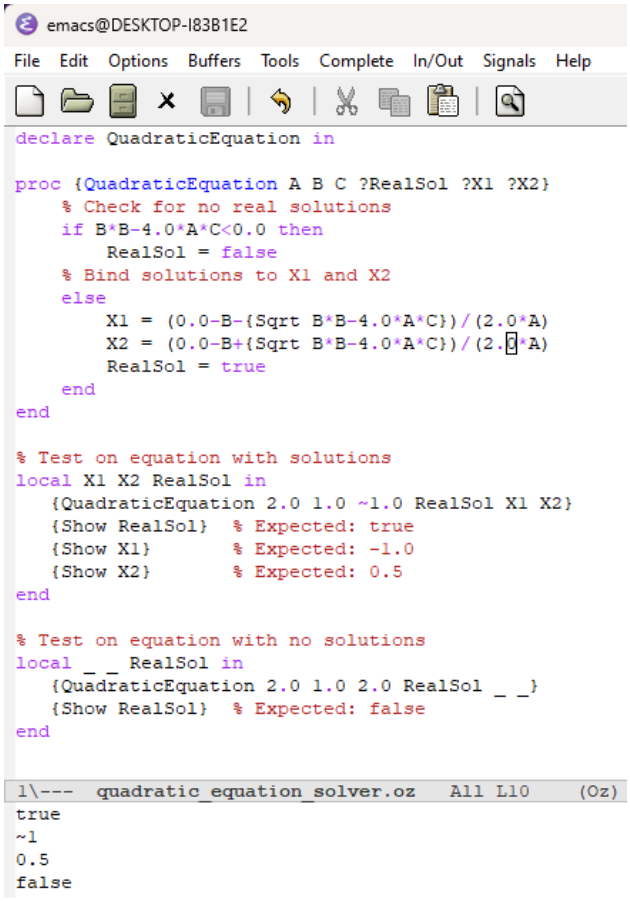

Figure 1: quadratic_equation_solver.oz

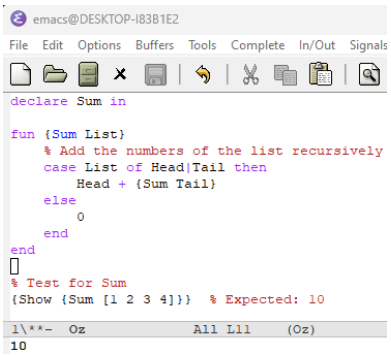

Figure 2: sum_list.oz

Task: 3 RightFold, Sum, and Length using RightFold

Task: 3.1 Implementation of `fun {RightFold List Op U}`

`RightFold` recursively applies the operation `Op` to each element of the list, starting with the neutral element `U`. Plese see Figure 3 for the code implementation.

Task: 3.2 Explanation of RightFold Code

- Base case: If the list is empty, return the neutral element `U`.
- Recursive case: Apply the operation `Op` to the head of the list and the result of folding the tail.

Task: 3.3 Re-implementation of `Sum` and `Length`

Using `RightFold`:

- `Sum2` calculates the sum of a list by folding it with `Op = +` and neutral element 0.
- `Length2` calculates the length of a list by folding it with `Op = 1+Y` and neutral element 0.

Task: 3.4 Difference between LeftFold and RightFold

For commutative operations like addition, LeftFold and RightFold yield the same result. For non-commutative operations like subtraction, they produce different results.

Task: 3.5 Appropriate value for U in product operation

The neutral element for the product operation is 1.

Task: 4 Quadratic Function

The `Quadratic` function returns another function that calculates the value of the quadratic expression $f(x) = Ax^2 + Bx + C$ for a given x. The code implementation and given example *{System.show {{Quadratic 3 2 1} 2}}* is displayed in Figure 4.

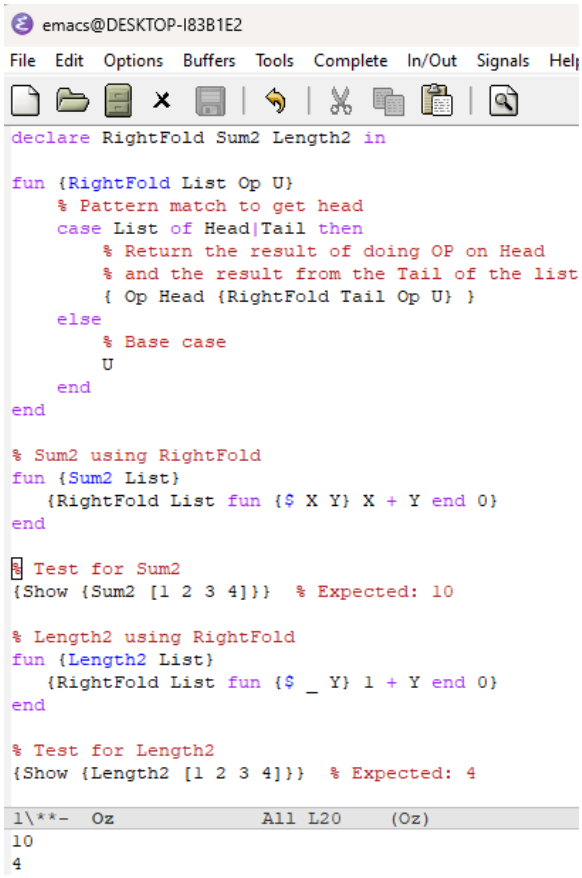

Figure 3: rightfold.oz

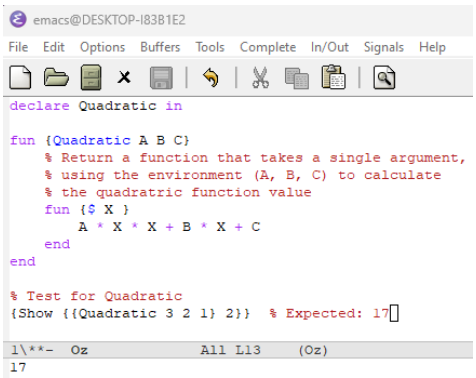

Figure 4: quadratic.oz

Task: 5 Lazy Number Generator

Task: 5.1 Implementation of `fun {LazyNumberGenerator StartValue}`

`LazyNumberGenerator` returns a pair where the first element is the current number and the second element is a function that produces the next number in the sequence. The code implementation and given examples are presented in Figure 5.

Task: 5.2 High-level description and limitations

This function generates a sequence of numbers lazily, meaning that the next number is computed only when requested. However, without native support for lazy evaluation, deeply nested recursive calls can lead to stack overflow.

Task: 6 Tail Recursion

Task: 6.1 Is the Sum function tail recursive?

No, the `Sum` function is not tail-recursive because the addition happens after the recursive call.

Task: 6.2 Benefit of tail recursion

Tail recursion is optimized by reusing the current function frame, thus improving memory usage and preventing stack overflow.

Task: 6.3 Do all languages benefit from tail recursion?

No, only languages that support tail call optimization benefit from tail recursion. Some languages do not optimize tail calls and still accumulate stack frames.

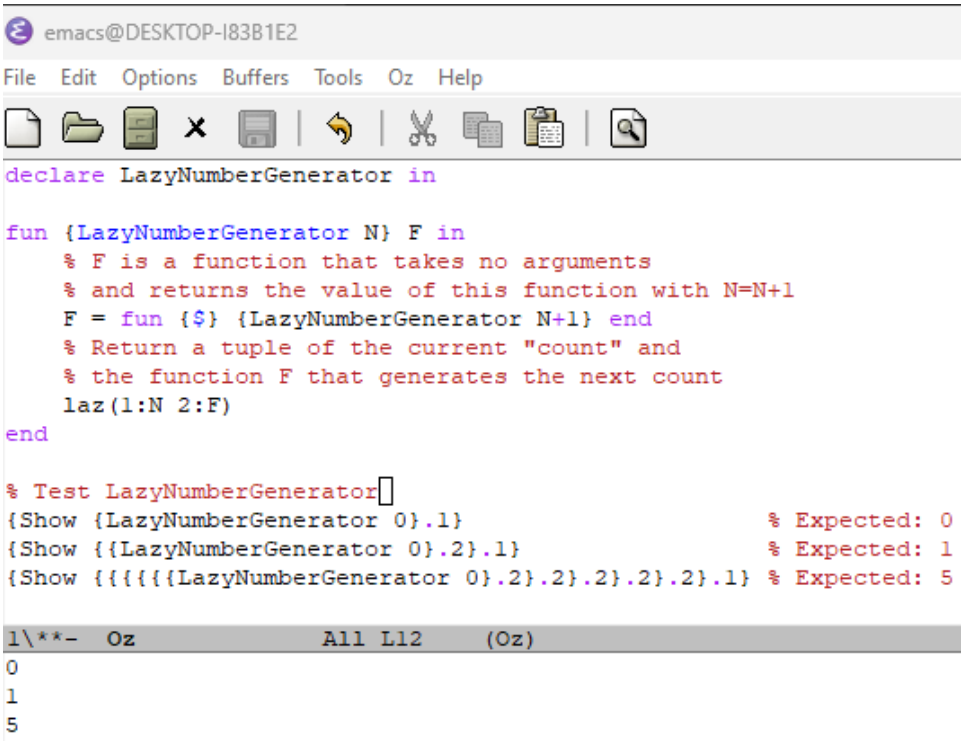

Figure 5: lazy_number_generator.oz