Names:

Project 0: Lisp Introduction and Set Operations

CSCI 561

September 13, 2024

1. Describe the Lisp development environment that you used for this project. (*Hint: The correct answer is ALIVE, SLIME, or similar.*)
2. What are the types of the following Lisp expressions?
   (a) `1` : *An integer Literal.*
   (b) `(+ 1 2)` : *An s-expression that evaluates to an integer (3) because it adds two integers.*
   (c) `'(+ 1 2)` : *A quoted list with three elements that doesn't evaluate.*
   (d) `(eval '(+ 1 2))` : *An s-expression that evaluates to integer (3) because it's evaluating a quoted list with three elements.*
   (e) `(lambda () (+ 1 2))` : *An anonymous function that adds two integers.*
   (f) `"foo"` : *A string literal.*
   (g) `'bar` : *A quoted symbol bar.*
3. Tail Calls:
   (a) What is tail recursion?
   *A*nswer:
   (b) In the recursive implementation, will `fold-left` or `fold-right` be more memory-efficient? Why?
   *A*nswer:
4. Lisp and Python represent code differently.
   (a) Contrast the representations of Lisp code and Python code.
   *A*nswer: Lisp represents code as lists, allowing it to treat code as data, while Python represents code as text that needs to be parsed before execution.
   (b) How does Python's `eval()` differ from the approach of Lisp?
   *A*nswer: Python's `eval()` evaluates a string of code and it requires parsing, whereas Lisp can directly evaluate lists, since its code is inherently structured as data.
5. GCC supports an extension to the C language that allows local/nested functions (functions contained in other functions). A GCC local function can access local variables from its parent function.
   (a) What problems would arise if you return a function pointer to a GCC local function? *(Hint: "Funarg problem")*
   *A*nswer: When returning and exiting that scope local variables from the GCC local function's parent function would be deallocated so the returned local function would no longer be able to access them and would produce undefined behaviour.
   (b) How does Lisp handle this problem?
   *A*nswer: Lisp solves this by packing the function with all the local variables it requires so that no matter where it is used the function will always have access to the environment in which it was created.

Names:

6. Test the performance of your implementation of `merge-sort`.

   (a) Plot the running time of both your `merge-sort` implementation and the builtin Lisp `sort` function for increasing input sizes. Include enough data points to demonstrate the empirical asymptotic running time.

   `graphics/mergesort1.png`

   `graphics/mergesort2.png`

   (b) What asymptotic running time did you expect for `merge-sort` and the builtin Lisp `sort` function, and what running time did you observe? Explain any differences.

   *A*nswer: I expected O(nlogn) time for merge-sort because merge sort takes O(nlogn) time and at no point in my implementation did I use methods with a slower asymptotic time complexity than assumed (such as by splitting a list in O(n^2) time rather than O(n)). I expected O(nlogn) time for sort because it is a generic sorting algorithm and provided that nothing is known about the list being sorted, O(nlogn) is the best asymptotic running time any existing sorting algorithm can achieve. Both of these hypotheses correlated with the results as shown in the second graph in the previous answer by how dividing the number of cycles by the list size times the log of the list size and plotting that vs the list size creates a roughly horizontal line.

7. Newton's method is a powerful technique for finding roots of real-valued functions. Given a function $p(x)$, Newton's method iteratively computes the next approximation x_{n+1} of the root using its derivative, $p'(x)$, as follows:

$$x_{n+1} = x_n - \frac{p(x_n)}{p'(x_n)}$$

   For this question, you will use your implementation of `find-fixpoint` to find roots of polynomials via Newton's method.

   (a) Determine $f(x)$:

   Given a polynomial $p(x)$ and its derivative $p'(x)$, determine the function $f(x)$ that should be used in the `find-fixpoint` function. Write down the expression for $f(x)$.

   *A*nswer:

   Our `find-fixpoint` function does not have any specific functionality that would prevent it from finding roots if the correct input function was given, thus we need only replace $f(x)$ with the iterative Newton's expression to find the roots using our function. This means that our function is:

$$f(x) = x - \frac{p(x_n)}{p'(x_n)}$$

   (b) Implement and Test:

   Implement the function $f(x)$ you derived, and use `find-fixpoint` to find the roots of the following polynomials using Newton's method:

   i. $p(x) = x^2 - 2$

   ii. $p(x) = x^3 - x - 2$

   iii. $p(x) = x^3 - 6x^2 + 11x - 6$ (roots are 1, 2, 3)

   For each polynomial, choose an initial guess and use a precision of 0.001. Give the parameters to the call to `find-fixpoint` and report the root found by your implementation.

*A*nswer:
For $p(x) = x^2 - 2$ our function must be:

$$f(x) = x - \frac{x^2 - 2}{2x}.$$

We set our initial value as 1.0 to find our positive root and -1.0 to find our negative root. After implementing `find-fixpoint`, we found that the roots were 1.414 and -1.414, respectively.
For $p(x) = x^3 - x - 2$, ur function must be:

$$f(x) = x - \frac{x^3 - x - 2}{3x^2 - 1}.$$

Since there is only one root, we set our initial value as 1.0 to find the root. After implementing `find-fixpoint`, we found the root to be 1.521.
For $p(x) = x^3 - 6x^2 + 11x - 6$, our function must be:

$$f(x) = x - \frac{x^3 - 6x^2 + 11x - 6}{3x^2 - 12x + 11}.$$

Since there were three roots, we set our initial values to 0.6, 1.6, and 2.6 to make sure we captured all three roots. After implementing `find-fixpoint` we found the roots to be 1, 2, and 3 respectively.