CSCE625: Artificial Intelligence
Programming Assignment 1 : Search, BFS, and Lazy Evaluation

Dylan A. Shell

Jan 21, 2019

This is posted on the 22 Jan. 2019. Please have your implementation working by the 31st so that we can discuss it in class on that day. (Of course, discuss this on piazza or with your peers!)

Question

Consider the following puzzle. It is in the vein of Russell and Norvig's toy problem on page 73, that they attribute to Donald Knuth.

You are given a pair of numbers: the first is the seed, x_0, and the second the target x_f. You are provided, additionally, with two functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$.

Puzzle: Write a program to give either the minimal number of function applications (of f and g) needed to reach x_f from x_0, or state that one cannot reach x_f from x_0.

Approach

This problem is from `https://nmattia.com/posts/2016-07-31-bfs-tree.html` In that blog post Nicolas Mattia considers the two functions f and g defined as follows:

$$f(x) = 2x + 1 \qquad \text{and} \qquad g(x) = 3x + 1.$$

The author provides code to solve exactly this problem on that page. Actually, that page *is* code, written in Haskell. (Look up literate programming if you've never heard of it.)

Task 1

Convert his solution, for those two particular f and g functions, to Scheme. His approach is not quite the same as the straightforward stack-based method given in Russell and Norvig. Instead, it uses Haskell's lazy evaluation to provide a sort of infinite data structure. You are to do the same thing: implement the search using lazy evaluation.

We have Gambit Scheme installed on the `compute.cs.tamu.edu` server. The page at `http://www.iro.umontreal.ca/~gambit/doc/gambit.html` provides some information that you might find useful in getting up and running with the interpreter and compiler.

By default Scheme uses eager evaluation. You have to make use of the `delay` and `force` keywords. Details on this can be found here: `http://www.shido.info/lisp/scheme_lazy_e.html`

(Note that the text in that guide uses Guile, not Gambit Scheme, so the `promise?` type checking function isn't available for us. The other functionality is supported, however.)

Task 2

Reflect on these things:

- Consider Knuth's original problem. That requires more than just an f and g. Can you pose his problem with a modification of your code?
- What properties are needed to ensure that your program is guaranteed to give a solution to the puzzle?