An *Efficient* Interpreter

November 14, 2016

Our step-by-step interpreter works, but is very slow. There are two kinds of slowness:

1. Intrinsic slowness: put an $O(n)$ program in, get an $O(n^2)$ execution time.
2. Incidental slowness: interpreters are just slow.

This document concerns the first.

1 Sources of intrinsic slowness in our existing interpreter

At least:

- Finding the redex each time
- Looking up things in the environment
- Removing redundant let-bindings
- Printing out the current step

2 Possible solutions

- Finding the redex: we need to adapt the techniques used for abstract machines where the next redex is known (use of evaluation holes etc). We must retain the ability to print out the each step, and evaluate step-by-step, but can we gain the ability to find the next redex in $O(1)$
- Looking things up in the environment: standard techniques, presumably. Hash table or something. But we must keep the environment along with each node, so we don't need to keep making it up on every step. Can this be done easily?
- Removing redundant let-bindings: We have introduced a new type, which keeps implicit let-bindings around each expression. This is useful to simplify pattern matching when finding the redex, but it may also allow us to automatically remove let-bindings when they are no longer required. For example, when `let x = 1 in let y = 2 in x + y` is transformed to `let x = 1 in let y = 2 in 1 + y` we can automatically go to `let y = 2 in 1 + y`. Is this possible in general? To keep track of the number of uses of a variable, and know when it goes to zero? This is a bit like reference counting...
- Printing out the current step: If we don't print out each step of the evaluation, the three techniques above will suffice. However, if we print out some or all steps of the evaluation, it's clear that the complexity increases (not least because the amount of characters output to the screen is probably $O(n^2)$ for an $O(n)$ program. However, adjacent steps of evaluation have similar printings: only the redex (plus maybe some dropped lets) differ. Can we exploit this?