6.945 Final Project Proposal: Debugging, Better

Jared Pochtar, Blake Elias, and Kenny Friedman

April 7, 2017

"Debugging is twice as hard as writing the code in the first place."
-Brian W. Kernighan

Kernighan wrote those words in 1978, and they are still true today. Our goal in this project is to make debugging slightly easier, by introducing a suite of tools to understand the execution of Scheme code.

We plan to implement a programming aid / debugging / sketching environment which allows programmers to better visualize and comprehend the execution of programs as the programmers build them. Below, we describe the three major components to our system.

1 Sliders

One simple feature we want in Scheme, is to turn numerical variables into graphical sliders which manipulate their respective variables.

We can extend this to have sliders that animate the evolution of a program, as well. In Chapter 1 of SICP, figures 1.3 and 1.4 show the evolution of the `fact` and `fact-iter` procedures as they get evaluated. We plan to have function which automatically generates these kinds of diagrams. However, rather than having to see the entire diagram, we plan to implement a graphical slider that lets you flip back and forth through the different levels of calls. Starting with `(fact 6)`, dragging the slider a bit to the right would change the expression to `(* 6 (fact 5))`. Dragging right a bit more would change it to `(* 6 (* 5 (fact 4)))`, and so forth. This would be a more intuitive way to understand the behavior of a program.

The most similar functionality we have been able to find is on Python-Tutor, although it is a bit uglier than we'd like because it has to show all the state that Python stores, whereas we plan to just show a single s-expression that expands/contracts/evolves as the programmer drags the slider back

and forth. (Although, we may run into some complication when handling stateful operations like `set!`, `display`, etc., and if the tool detects such a case, may have to revert back to showing the whole environment).

2 Highlight to Evaluate

In Emacs and Edwin, it is possible to see the evaluation of a subset of a function by executing part of the function, within the function. There are two major problems with this: first, you have to carefully navigate your cursor to the proper location, and 2) the result of the execution is placed as a comment, within your code. If you only wanted to execute the function to debug, you are either left with an ugly comment in your code, or you have to manually delete the comment.

What if, anytime you highlight a piece of code, an attempt at the execution is performed. If it succeeds quickly, then you see the result of the execution directly below your cursor: not as an inline comment, but as a temporary subscript that dissapears when you deselect the code. We believe this feature will make it extremely fast to quickly explore the subcomponents of a function you are writing. It will also serve a valuable purpose as a tool for quickly reading and comprehending subcomponents of other people's code.

3 Autocomplete Arguments

When typing a function to call, you currently have to remember how many arguments the function takes, as well as what those arguments are. One way to assist in remembering the parameters is to have an editor smart enough to autofill the names of the parameters of the function you are typing. Some languages and IDEs already have this feature, such as Objective-C and Swift in XCode. We plan to implement this for Scheme. This would not be in-line text, but a token. Therefore, if the autocomplete gets the code wrong, a user simply continues to type, and it overrides the suggestion.

Implementation

Each of us will take the lead on one of these major components. Jared will lead on Autocomplete Arguments, Kenny will lead Highlight to Evaluate, and Blake will lead Sliders. However, it is likely that we will all work

on each of them, to collectively create a unified programming environment experience.

We plan to have documentation showing representative use cases of each feature. We also plan to have well-named functions and commented code.