Project Part 2

Jason Hemann

January 27, 2018

1 Preamble

- Do consult class notes, online lecture notes, and test cases when completing this assignment.
- Name your file `main.ss`, and use a command like the following: (`zip <username>-p2.zip main.ss pmatch.ss`)

2 To begin:

Double-check that you have completed the first part before you begin here. You will not want to go back and fix it.

Save versions of your files. You will not want to have to go back and debug this. **Don't debug!** If it goes wrong, just throw version `n` away and start back at `n-1`.

You should be able to test your program between each step. Sometimes you can even test your program as you work through the pieces of a step.

2.1 Subsequent Steps

1. Define `apply-env`, `apply-closure`, and `apply-k`, and add all the calls to those functions. Notice that after CPSing, your `apply-closure` and `apply-env` now take three arguments.
2. Define `extend-env` and replace all 3 of your explicitly higher-order representations of environments with calls to `extend-env`.
3. Add a `^` to each of the formal parameters of `extend-env` (not the inner function, mind you). Ensure that the inner functions in both `extend-env` and `empty-env` use the same formal parameters as the second argument to `apply-env` (here, typically `y` and `k^`).
4. Replace your higher-order function representation of environments with a tagged-list data structure representation. Consider first ensuring that the last two formal parameters to `apply-env` (`y` and `k`) are the same as the formal parameters to the inner functions in `extend-env` and `apply-env`. Remember, if you add `(else (env-cps y k))` as the last line of your `pmatch` expression, you can test each transformation one at a time.
5. If you used it, remove the `(else (env-cps y k))` as the last line of your `pmatch` expression in `apply-env`.

6. Define `make-closure` and replace your explicitly higher-order representation of a closure with a call to `make-closure`.

7. Replace your higher-order function representation of closures with a tagged-list data structure representation. Consider first ensuring that the last two formal parameters to `apply-closure` (`a` and `k`) are the same as the formal parameters to the inner function in `make-closure`.

8. Define constructors for your continuations, and replace your explicitly higher-order function continuations with calls to these constructors. For nested continuations, you should consider working from the inside out. Remember, if you transform your constructors one at a time, you can test your program after each replacement.

9. Add a `^` to each of the formal parameters of your continuation helpers, and change the body to correspond with these changed variable names (that is, do an alpha substitution). Then, ensure that the inner function in each of your continuation helpers uses the same formal parameter as the second argument to `apply-k` (typically, `v`).

10. Replace your higher-order function representations of continuations with a tagged-list data structure representation. Remember, if you add `(else (k v))` as the last line of your `pmatch`, you can test each transformation one at a time. If you're good, you can do almost all of this step with a keyboard macro ;-).

11. Remove the `(else (k v))` line to ensure that you've properly removed all higher-order function representations of continuations.