CS 450 Homework 9

The Explicit-Control Evaluator

Carl Offner

Spring 2022

Due Monday, May 2, 5:00 PM

In this assignment you will make some small modifications to the explicit-control evaluator presented in Chapter 5 of the text, and experiment with enabling and disabling tail-recursion.

The assignment is not easy and will take you a fair amount of time. But please note that of the 5 problems, the last 3 are very straightforward and probably won't take very much time at all. You can do these problems in any order.

Start by copying your `hw8/regsim.scm` file into a new `hw9` directory. Then copy all the files in `~offner/cs450/hw9` into that directory as well. You may end up making edits to any of these files (with the exception of `load-eceval.scm`; you won't need to make any changes to that file). I will collect them all. I will also collect a file `notes.txt`, as usual.

Remember that you invoke the explicit-control evaluator by loading `load-eceval.scm`. For instance, from the Unix prompt you can just type

```
scheme load-eceval.scm
```

If `eceval` exits to the underlying Scheme because of an error, you can restart `eceval` from within Scheme by simply typing

```
(start eceval)
```

You should do either problem 1 or problem 2. You don't have to do both. Pick either one. If you do them both, I will count the second one as extra credit.

If you are unsure which one you want to do, I suggest problem 2, because the experience of doing that problem may help you in the next assignment. But either one is acceptable.

You must do problems 3, 4, and 5.

---

1. The footnote on page 549 points out that the dispatch could be written in data-directed style, as we did in `s450.scm`. Implement this.

   Some hints: conceptually this is really very similar to what we did in `s450.scm`. You create a table to manage the special forms, just as you did for `s450.scm`. The difference is this: In `s450.scm` the `cdr` of each pair was the `lambda` expression that implemented the special form. But now the `cdr` of each pair is the label in `eceval.scm` at which the code implementing that special form is found. So `install-special-form`

wants to be called like this, in `eceval-support.scm`:

```
(install-special-form 'quote 'ev-quoted)
(install-special-form 'set! 'ev-assignment)
```

and so on.

`eceval.scm` will need to use a built-in operation (i.e., a machine "operation", or what I have been calling a "machine primitive") `type-of`, which is just as trivially simple as it was in `s450.scm`. You will also need to check to see if a "bare symbol" is a special form name, so you will probably also need a built-in operation in `eceval.scm` for this purpose. Don't be surprised if you end up having two jumps to the same location; at least, that's what happened when I implemented this.

When I implemented this, I found it necessary to allow instructions in `eceval.scm` of the form

```
(assign val (label (reg val)))
```

That is, in the expression `(label xxx)`, `xxx` must be able to be a register reference rather than a literal. To do this you have to add some code to `regsim.scm`. This happens in `make-elementary-exp`. (Note: if you don't need to do this, that's perfectly fine. I'm curious to see how different people do this. I just wanted to warn you of something that might come up and how to deal with it.)

2. Exercise 5.24 (page 560). This exercise asks you to add support for the special form `cond`. In the previous Exercise 5.23 (which I am not asking you to do), the book suggests you "cheat" and use the syntax transformer `cond->if` in `syntax.scm`. **I don't want you to do this. And I don't want you to use `expand-clauses`, either.** Make `cond` a special form with its own machine code in `eceval.scm` and its own label `ev-cond`. You will of course need some of the selectors such as `cond-clauses` from `syntax.scm` as built-in operators; just remember to add them to the list of built-in operators in the machine. You may want to add some more selectors of your own; feel free to do this.

   You will want to be careful about considering which registers need to be saved and restored when implementing this. Present your reasoning in `notes.txt`.

3. Exercise 5.26 (page 564). For clarity, call this function (which the book calls `factorial`) `fact-i` (the "i" for "iterative"). Be sure to answer the questions in parts `a` and `b`.

   Note that this exercise does not involve writing any code. You just have to experiment with the machine.

4. Exercise 5.27 (page 564). For clarity, call this function (which the book also calls `factorial`) `fact-r` (the "r" for "recursive").

   Like the last exercise, this exercise does not involve writing any code. In both exercises, you should collect data at least for values of `n` from 1 through 5.

5. Exercise 5.28 (page 565). This requires changing some of the code in `eceval.scm`, but the book shows you exactly what to do. Add the code anywhere you want, just so long as it won't be "fallen into", and just leave the label `ev-sequence` for the new code commented out. Then to try the new code for this exercise, comment out the old label `ev-sequence` and uncomment the new one.

Please note that this problem is similar to the two previous ones combined, so you will need to report just as much data and just as many formulas as in those two problems combined.

Some more information about problems 3, 4 and 5:

- You are asked to write some simple formulas. You have all taken algebra (and calculus, for that matter) and you should all know how to write a simple formula cleanly. For instance, suppose you come up with some expression such as

  ```
  2(n-3) + 11
  ```

  This is **not** what you want to write. You want to simplify this, so you write

  ```
  2n + 5
  ```

  And you **do not** write

  ```
  2*n + 5    !!! WRONG !!!
  ```

  That's not mathematical notation. That is what you might write if you were writing code in some computer language. But you are not writing code in some computer language. You are writing an algebraic expression.

  Please be careful about this.

- The point of producing all these tables (there should be four of them) and formulas (there should actually be eight of them) is to learn something, not just produce some meaningless expressions. What do you observe from these results? Put your answers in `notes.txt`.