Homework 6: Save the Environment, Part I

Part I Due Thursday, March 19 at 9:55 am

This homework refers to the heavily commented implementation of the Eph language at http://cs.williams.edu/~morgan/www/cs334/code/eph/eph.zip, which is required reading.

The best way for some people to internalize the material that we've covered on interpreters: parsers, grammars, eval, apply, value and expression domains, and environments is to actually implement it. Unfortunately, debugging the implementation can take a long time and sometimes you have to refactor a design several times until reaching a good one. I encourage you to write an interpreter or modify an existing one if that is how you learn best. However, to save you time, for this assignment I'm giving you a complete interpreter and asking you to study the source code for insight rather than starting your own from scratch. Homeworks 6 and 7 refer to the same source code. So that you do not have to work over spring break, homework 7 is relatively short and can be completed without the benefit of a weekend.

It will probably take you several hours to read the code. Don't just start at the beginning and read the whole thing. Instead, use these questions as a guide and ask yourself other questions about how pieces of the interpreter operate. Trace lines of execution and follow a single feature as it winds through the code. It is much easier to download the code and play with in DrScheme than to read it from a printout. This also lets you search the code. Again, I encourage you to modify the interpreter, either functionally or with print statements, to better understand how it works.

Scheme code has varying significance. Some large procedures are just boilerplate and the key features may be condensed into a few lines. Some important parts to understand about the interpreter are:

- eph-apply-PROC in interpreter.ss
- evt-set! and evt-get in evt.ss
- wrap in library.ss
- How the Eph library's `apply` procedure (not eph-apply) is defined in library.ss
- How the value and expression domains are distinguished throughout (and why that is important)

1. **QUOTE** (4 points)
   a. Write the equivalent Scheme forms of the following four expressions using the QUOTE form instead of the "tick-mark" syntactic sugar:

   ```
   'x
   '(x)
   ''x
   '('x)
   '(quote x)
   ```

   b. The Eph parser treats Scheme numbers, booleans, null, and void as Eph literals (LIT). So, why does the Eph parser convert 'x into a variable (VAR) instead of a symbol literal?

2. **AND** (6 points)
    a. Why don't cases for AND and OR expressions appear in eph-eval (interpreter.ss)?
    b. Why is AND implemented in parser.ss as:

    ```
    [(and) (IF (parse (second e)) (parse (third e)) (LIT #f))]
    ```

    instead of:

    ```
    [(and) (IF (parse (second e)) (parse (third e)) (BOOL #f))]
    ```

3. **Laziness**
    a. Eph is implemented as an eager language. How would you change it to be a lazy language? Your answer can be either the relevant pieces of an actual implementation, or a detailed description of the changes required at the design level. In either case, there are several details to which you must attend. Note that when a procedure argument is eventually evaluated, it should only ever be evaluated *once*. In a language with mutation (or other side-effects), multiple evaluation can produce incorrect results.
    b. Once Eph is lazy, how can you simplify the rest of the interpreter (perhaps by moving some work to the parser...)?

4. **Environments** (6 points)
    a. What is the difference between evt-set! and evt-define! (evt.ss)?
    b. Look at how the provided "library" of routines is created. There are two pieces—the manually created procedures and the ones that are evaluated, all within library.ss. What is the difference between the two techniques, and why are both used?

5. **Apply** (6 points)
    This question refers to material that was not explicitly covered in class. Use the PLT documentation for APPLY and examination of library.ss to guide you to a solution.

    a. What value does the Scheme expression:

    ```
    (apply + (list 7 10))
    ```

    evaluate to?
    b. How does Scheme's built-in APPLY differ from a direct application of the first argument (which must be a procedure)?
    c. How is Scheme's APPLY useful in the context of writing your own interpreter?

Challenge: (glory and wisdom, but no points)

Add a new feature to the implementation. Some ideas are:

- LETREC
- WHILE loops (invent your own syntax!)
- Quoted list literals

Homework 7: Save the Environment, Part II

Part II Due Thursday, April 2 at 9:55 am

This homework refers to the heavily commented implementation of the Eph language at http://cs.williams.edu/~morgan/www/cs334/code/eph/eph.zip, which is required reading.

6. **Elegance** (9 points)
   I used a larger set of Scheme functions and forms than you've seen before, and generally wrote in a more aggressively functional style than you are accustomed to. You will need to look some of them up in the Scheme manual (push F1 in DrScheme). Identify two design patterns that you find particularly "good" (elegant, clever, efficient, or clear) and for each:

   a. note one location where each is used,
   b. describe why it is good, and
   c. describe how the same task is generally accomplished in Java.

7. **Inelegance** (9 points)
   In some places I intentionally wrote in a very verbose or imperative style to make the code clearer to you on reading. There are likely even a few bugs still left in the code. Identify one area that could be made significantly more elegant, efficient, or correct. Give that code, your rewritten version, and an explanation of why your version is an improvement.

8. **LET** (10 points)
   Describe how to implement Scheme's LET for Eph. This should be the full-blown multi-variable version of LET, and it should use environments, not substitution. Your answer can be either the relevant pieces of an actual implementation, or a detailed description of the changes required at the design level. In either case, there are several details to which you must attend. There are many ways of implementing LET, and some are much better (i.e., *worth more points*) than others. Explain why you chose your method. Be careful not to implement LET* or LETREC by accident.

9. **Continuations** (10 points)
   Briefly describe the changes to the following areas to add continuations. Unlike previous questions, I am only looking for a high-level answer and not all of the details:

   a. Grammar
   b. eph-eval/eph-apply
   c. Library

Challenge: (glory and wisdom, but no points)

Add a new feature to the implementation. Some ideas are:

- One or more new library routines, like: foldl, reverse, eval, set-car!, set-cdr!, length
- String data type (and literals and utility functions)
- Return