MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science

6.5150/6.5151 Spring 2024
Problem Set 7

Issued: Wed. 3 April 2024 **Due: Fri. 12 April 2024**

Reading:
SDF Chapter 5 -- Evaluation: Sections 5.3 and 5.4

Some, perhaps useful background material
SICP, From Chapter 4: section 4.3; (pp. 412--437)

Heavy Evaluator Hacking

In this problem set we build interpreters in a different direction. We start with the essential EVAL/APPLY interpreter, written as an analyzer of the syntax into a compiler of compositions of execution procedures -- a small combinator language. We will warm up by making modifications to this evaluator.

Next, we will change the evaluator to include AMB expressions. To add AMB, the execution procedures will all have a different shape: in addition to the environment, each will take two "continuation procedures" SUCCEED and FAIL. In general, when a computation comes up with a value it will invoke SUCCEED with the proposed value and a complaint department which, if invoked, will try to produce an alternate value. If a computation cannot come up with a value, it will invoke the complaint department passed to it in the FAIL continuation.

An important lesson to be learned here is how to use continuation procedures to partially escape the expression structure of the language. By construction, a functional expression has a unique value. However, in the AMB system an expression may be ambiguous as to its value... Think about how we arrange that to make sense!

Separating Syntactic Analysis from Execution
(Compiling to Execution Procedures)

You should read Section 5.3 of SDF (and perhaps SICP section 4.1.7 for more background) carefully here.

To Do

Load the material for the following problems, with the incantation

```
(manage 'new 'compiling-to-execution-procedures)
```

This will get the evaluator described in SDF (similar to the one described in SICP, but generalized for extensibility).

Note: A really good job on the following problems may require adding a compile-time environment to each ANALYZE handler.

Exercise 5.14: Constant folding p. 268
Remember, you already have some experience with "algebraic simplification" from chapter 4.

Exercise 5.15: Other optimizations p. 269
One way to implement optimizations is to integrate a code simplifier into each of the handlers for ANALYZE. For example, in the analysis of an IF, can we determine the value of a predicate expression, perhaps by constant folding, at compile time? If so, the consequent or the alternative of the IF can be elided (dead-code elimination).

Exercise 5b: OPTIONAL (Not in book)
We have had experience making a primitive type-inference engine. Indeed, that type inference system maintained an inference-time environment that might involve the compile-time environment. Explain, in a SHORT essay, how would you integrate that kind of type inference into the ANALYZE phase of this system? How could it help to make better execution procedures?

AMB and Nondeterministic Programming

Now comes the real fun part of this problem set! Please read section 5.4 of SDF (and perhaps section 4.3 of SICP) carefully before starting this part. This interpreter requires a change in the interface structure of the execution procedures that code compiles into, so it is quite different. Of course, our system differs from the one in SICP in that it is implemented with generic extension capability. You can load the interpreter extended for AMB with:

```
(manage 'new 'exploratory-behavior)
```

To Do

Exercise 5.17: A puzzle	**pp. 277–278**
Exercise 5.18: Failure Detection	**p. 278**
Exercise 5.19: Assignment	**pp. 278–279**