MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Compvter Science

Problem Set 5

Problem 1: State

Do exercise 8.9 (a) and (b) on page 296 of the course notes.

Problem 2: More State

a. Give a translation of call-by-value FLAVARK! into call-by-value FLK!. You do not need to translate `rec`.

b. Give a translation of call-by-reference FLAVARK! into call-by-value FLK!. You do not need to translate `rec`.

Problem 3: Control

Sam Antix decides to add a new exception handling primitive to FL! + {`raise`, `trap`}. He adds the following expression to the grammar of FL! + {`raise`, `trap`}:

$$\texttt{(handle } I \ E_h \ E_b\texttt{)}$$

Informally, Sam's new expression is similar to

$$\texttt{(trap } I \ E_h \ E_b\texttt{)}.$$

Both expressions evaluate E_h to a handler procedure and dynamically install the procedure as a handler for exception I. Then the body expression E_b is evaluated. If E_b returns normally, then the installed handler is removed, and the value returned is the value of E_b.

However, if the evaluation of E_b reaches an expression

$$\texttt{(raise } I \ E\texttt{)},$$

then E is evaluated and the handler procedure is applied to the resulting value. With `trap`, this application is evaluated at the point of the `raise` expression. But with `handle`, the application is evaluated at the point of the `handle` expression. In particular, both the dynamic environment and continuation are inherited from the `handle` expression, *not* the `raise` expression.

Here are a few example evaluations involving `handle`:

```
(handle a (lambda (x) (+ 4000 x))
  (handle b (lambda (x) (+ 300 (raise a (+ x 4))))
    (handle a (lambda (x) (+ 20 x))
      (+ 1 (raise b 2)))))
⇒ 4006

(handle a (lambda (x) (* x 10))
  (+ 1 (raise a (+ 2 (raise a 4)))))
⇒ 40
```

a. Extend the denotational semantics of call-by-value FLK! + {`raise`, `trap`} with a valuation clause for `handle`.

b. Give a desugaring of `handle` into FL! + {`raise`, `trap`, `label`, `jump`}.