

Principles of Programming Languages Handout

Local binding (like `let` in Scheme) binds some variables to values, and then evaluates an expression (the body of the `let`) in the scope of these bindings. It is straightforward to add this construct to the interpreter: one needs to extend the environment. The additional clause in the `cases-expression` is this:

```
(let-exp (ids rands body)
  (let ((args (eval-rands rands env)))
    (eval-expression body (extend-env ids args env))))
```

Static binding is implemented using closures: a procedure evaluates to a closure, which carries its environment with it. Closures are defined as a datatype as follows:

```
(define-datatype procval procval?
  (closure
    (ids (list-of symbol?))
    (body expression?)
    (env environment?)))
```

The environment field of a closure holds the environment at the point of definition of the procedure.

A closure can be applied to some arguments. The environment in the closure is extended with bindings of the actual arguments for the formal parameters; and the body of the closure is then evaluated in this extended environment.

```
(define apply-procval
  (lambda (proc args)
    (cases procval proc
      (closure (ids body env)
        (eval-expression body (extend-env ids args env))))))
```

The interpreter with first-class procedures and local binding looks like this:

```
(define eval-expression
  (lambda (exp env)
    (cases expression exp
      various cases as before
      (let-exp (ids rands body)
        (let ((args (eval-rands rands env)))
          (eval-expression body (extend-env ids args env))))
      (proc-exp (ids body) (closure ids body env))
      (app-exp (rator rands)
        (let ((proc (eval-expression rator env))
              (args (eval-rands rands env)))
          (if (procval? proc)
              (apply-procval proc args)
              (eopl:error 'eval-expression
                "Attempt to apply non-procedure ~s" proc))))))
```