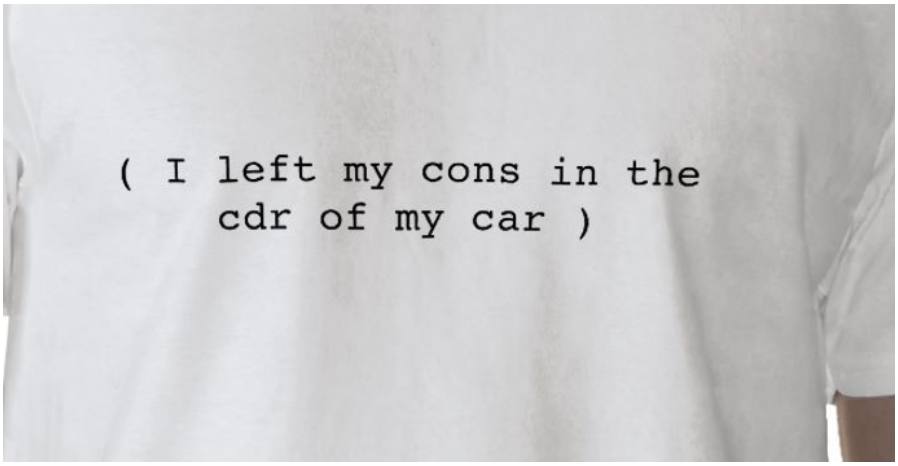
PRIMITIVES

Primitives



The Buck Stops Here!

- (Harry Truman)
- Primitives are evaluated as system calls.
- In our case, by calling Racket procedures
- Had we used C to implement our interpreter, we would have invoked C functions.

Storing Primitives

- Parse tree:

  (app-exp ((var-exp +) (lit-exp 3) (lit-exp 4)))
- Map (var-exp +) to Racket procedure +
- Do this in the environment:

  (+ <procedure +>)

Storing Arguments

- How do we best store the arguments to a procedure, wether primitive or not?
- Some procedures take one arguments, some may take an arbitrary number of arguments.
- Consider + and car:

```
(car '(a b c))          ;;; We are good to go.
(+ '(3 4 5))            ;;; This does not work.
```

Apply

- (apply <procedure> <list>) will apply the <procedure> to the <list> returning a single value.
- Different to “map”
- (apply + '(3 4 5)) returns 12
- (apply car '((a b c))) returns a

Improved Way of Storing Primitives

- Sometimes need to call apply, sometimes not
  (car '(a b c))
  (apply + '(3 4 5))
- We cannot simply apply primitives to arguments:

```
(define apply-proc
  (lambda (proc arg env)
    (if (closure? proc)
        (cases closure proc
         [closure-record (id body env)
           (eval-tree …)])
        (proc arg))))
```

Improved Way of Storing Primitives

- Instead, we need a separate function that handles all primitives:

```
(define apply-primitive-proc
  (lambda (id args)
    (case id
      [(+) (apply + args)]
      [(-) (apply - args)]
      [(car) (caar args)]
      [(cdr) (cdar args)]
      [(add1) (+ (car args) 1)]
      [else (error …)])))
```

Improved Way of Storing Primitives

- `apply-primitive-proc` gets called from `apply-proc`, which now looks like this:

```
(define apply-proc
  (lambda (proc args)
    (cases procedure proc
           [closure (parameters body env)
            (eval-expression body …)]
           [primitive (id)
            (apply-primitive-proc id args)]
```

Improved Way of Storing Primitives

- This means that we need to extend the data type for procedures.

```
(define-datatype procedure procedure?
  [primitive
     (id symbol?)]
  [closure
     (parameters (list-of symbol?))
     (body expression?)
     (env environment?)])
```

Improved Way of Storing Primitives

- This means a change to the entries in the environment.
- We need to tag all primitive procedure symbols with the added data type.
- `(apply-env <env> '+)` returns:

  `(primitive +)`

Adding primitives to the Environment

```
(define *prim-proc-names* '(+ - car cdr add1))

(define global-env
  (extend-env *prim-proc-names*
              (map primitive *prim-proc-names*)
              (empty-env)))
```