CS403: Programming Languages

Evaluation Order

Key ideas

- how Scheme evaluates expressions
- applicative order evaluation (immediate evaluation of function arguments)
- normal order evaluation (delayed evaluation of function arguments)

Applicative order

One way to evaluate a non-primitive expression in Scheme uses the following algorithm:

```
for each term of the expression (in any order)
    evaluate that term
apply the first term to the remaining terms
```

Consider the following definitions ...

```
(define (sqr a) (* a a))
(define (double a) (+ a a))
```

and the following expression...

```
(square (+ (double 2) (double 3)))
```

Under applicative order, the expression first evaluates to

```
(square (+ 4 9))
```

and then to:

```
(square 13)
```

and finally to:

```
169
```

Normal order

Contrast the above with normal order, in which all complex terms are substituted with equivalent primitive expressions before evaluation proceeds. Using the example above,

```
(square (+ (double 2) (double 3)))
```

becomes:

```
(square (+ (+ 2 2) (+ 3 3)))
```

which in turn becomes

```
(* (+ (+ 2 2) (+ 3 3)) (+ (+ 2 2) (+ 3 3)))
```

At this point simplification can begin, since all that is left are primitives, yielding

```
(* (+ 4 9) (+ 4 9))
```

With more simplification, the expression becomes

```
(* 13 13)
```

Finally, the end result is generated...

```
169
```

Note that applicative order can be much more efficient. Scheme uses applicative order, but later we will investigate a way to fake normal order evaluation for some interesting uses in Scheme.
enddocument