

## OPL, November 3, 2006

### I. Lisp

- Pure lisp: no destructors, like (setq x '(d e f)) (rplaca x 'b)
- Lambda expressions
- Functions as first-class entities: not in Common Lisp, but in Scheme

#### Common Lisp

```
(defun square (x)
  (* x x)
)
(defun mystery(x y)
  (funcall x y)
)
(mystery #'square 3)
```

#### Scheme

```
(define (square x)
  (* x x)
)
(define (mystery x y)
  (x y)
)
(mystery square 3)
```

- read, and (print(eval(read)))
- cond and recursion (see samples.lsp for fact, expt)
- (list 'z 'z 'z) for list creation [could be defined recursively from cons]
- prog, and lisp\_interpreter (also from samples.lsp)
- deriv.lsp (simplify (derivative a 'x)); Derive originally written in MuLisp
- unify.lsp to unify two patterns with variables and functions, written  
defparamter merely means dynamic global variable

### II. Prolog

- Pure prolog: declarative, no procedural part like ! or is
- Read pp. 07.10–07.17, to prepare for final lab, Lab 3, except for final project. It is due in two weeks; i.e. Friday, November 17. These people should make an appointment with me, together, so pls submit schedules: Josh, William, Chuck

### III. If time permits, more data type stuff from Chapter 6