CS 520: Introduction to Artificial Intelligence

Prof. Louis Steinberg

Lecture 7:

Lisp:

functional programming

closures

Review - LISP

- **Function as a data type**
- **Variable as pointer to typed data**
  - **Binding: a variable is a table key, not a place**
- **Garbage collection**
- **Symbol data type**
  - **Uniquified strings**
- **Code = tree of symbols**
- **Current dialects**
  - **Scheme, Common Lisp, Scripting languages**

Review - Lists

- Written with parens, no commas
- As data: Car = first, cdr = rest, cons = new
- As program: normally
  - Car specifies function
  - Cdr is a list of arguments
- Special forms:
  - `(quote a)` or `'a` prevents evaluation
  - `(if test alt1 alt2)` evaluates `test` and one `alt`

Evaluation of Non-Lists

- A number evals to itself
  - Value of 4 is 4
- A symbol is a variable
  - let => local binding

```
(let ((x 3)
      (y (+ 2 4)))
   (+ x y))
```

  - setf => assignment

```
(setf z (car x))
```

Functions

- **Meaning as a function ≠ value**
  - **unlike scheme**
- **Defun gives name a globally-visible function meaning**

```
(defun up-2 (number) (+ 2 number))
```

- **Flet gives name a locally-visible function meaning**

```
(flet ((up-2 (number)(+ 2 number)))
    (up-2 a))
```


Functions as arguments

```
(mapcar #'up-2 '(3 4 5)) => (5 6 7)
(some #'oddp '(3 4 5)) => t
(every #'oddp '(3 4 5)) => nil
(mapcar #'(lambda (number)
                  (+ 2 number))
          '(3 4 5))
    => (5 6 7)
```

- **`#'foo` means `(function foo)`**
  means "get the meaning of foo as a function"

Using functional values

- **Funcall evaluates all args normally, but then treats value of first arg as function.**

```
(let ((fns (list #'max #'min))
      (a 3)
      (b 4))
   (funcall (car fns) (+ b a) (- b a)))
 => 7
```


Lexical Scope

```
(let ((list '(5 4 3)))
  (mapcar #'(lambda (num)(+ num (car list)))
          '(10 15 20)))

(defun mapcar (func list)
   (if (null list)
       nil
     (cons (funcall func
                    (car list))
           (mapcar func
                   (cdr list)))))
```

Closures

```
(let* ((list '(5 4 3))
       (fn #'(lambda (num)
               (+ num (car list)))))
  (mapcar fn '(10 15 20)))
```

- **Fn is a *closure* - contains both**
  - **Code**
  - **Bindings**

Closures

```
(let* ((list '(5 4 3))
       (fn #'(lambda (num)
               (+ num (car list)))))
  (mapcar fn '(10 15 20)))
```

- **Fn is a *closure* - contains both**
  - **Code**
  - **Bindings**

CS520: Steinberg Lecture 07 9

Closures

```
(let* ((list '(5 4 3))
       (fn #'(lambda (num)
               (+ num (car list)))))
   (mapcar fn '(10 15 20)))
```

- **Fn is a *closure* - contains both**
  - **Code**
  - **Bindings**

Closures

```
(setq fn
  (let ((num 2))
     #'(lambda (x)
         (setq num (+ x num)))))
(print (funcall fn 3))
  5
(print (funcall fn 3))
  8
```

Programming Techniques

- **Data-driven programming**
  - **Many different ways to do a task, e.g. differentiate**
  - **Pieces of code in a data structure**
  - **Use data as a key to look up code in data structure**
- **To write a program for X, extend Lisp to be a good language for X**
  - **See code for derivative program**