

CS 4100 LISP

April 15, 2011

Based on slides by Istvan Jonyer
Book by MacLennan
Chapters 9, 10, 11

1

Fifth Generation

- Skip 4th generation: ADA
 - Data abstraction
 - Concurrent programming
- Paradigms
 - Functional: ML, Lisp
 - Logic: Prolog
 - Object Oriented: C++, Java

2

Chapter 9: List Processing: LISP

- History of LISP
 - McCarthy at MIT was looking to adapt high-level languages (Fortran) to AI - 1956
 - AI needs to represent relationships among data entities
 - Linked lists and other linked structures are common
 - Solution: Develop list processing library for Fortran
 - Other advances were also made
 - IF function: $X = \text{IF}(N, \text{EQ}, 0, \text{ICAR}(Y), \text{ICDR}(Y))$
 - List processing and conditional statement combined

3

What do we need?

- Recursive list processing functions
- Conditional expression

- First implementation
 - IBM 704
 - Demo in 1960
- Common Lisp standardized

4

Example LISP Program

```
(defun make-table (text table)
  (if (null text)
      table
      (make-table (cdr text)
                  (update-entry table (car
text)))
  )
)
```

- Called S-expressions (Symbolic)

5

Central Idea: Function Application

- There are 2 types of languages
 - Imperative
 - Like Fortran, Algol, Pascal, C, etc.
 - Routing execution from one assignment statement to another
 - Applicative
 - LISP
 - Applying a function to arguments
 - $(f a_1 a_2 \dots a_n)$
 - No need for control structures

6

Prefix Notation

- Prefix notation is used in LISP
 - Sometimes called Polish notation (Jan Lukasiewicz)
 - Operator comes before arguments
 - `(plus 1 2)` same as `1 + 2` in infix
 - `(plus 5 4 7 6 8 9)`
- Functions cannot be mixed because of the list structure
 - (As in Algol: `1 + 2 - 3`)
 - LISP is fully parenthesized
 - No need for precedence rules

7

cond Function

```
(cond
  ((null x) 0)
  ((eq x y) (f x))
  (t (g y)) )
```

- Equivalent to

```
if null(x) then 0
elseif x = y then f(x)
else g(y)
```

8

Function Definition

```
(defun make-table (text table)
  (if (null text)
      table
      (make-table (cdr text)
                  (update-entry table (car text)))
  )
)
```

- Function definition is achieved by calling a function(!) called defun, with arguments
 - Name *(make-table)*
 - Parameters *(text table)*
 - Body *(if ...)*

9

Everything Is a List

- Why is everything a list in LISP?
 - *Simplicity Principle*
 - A language should be as simple as possible. There should be a minimum number of concepts, with simple rules for their combination.
 - If there is only one basic mechanism in the language, the language is easier to learn, understand, and implement.

10

The List is the Data Structure

- Lists contain symbolic data
 - (set 'text '(to be or not to be))
 - Lists like (to be or not to be) can be manipulated like numbers in other languages (compared, concatenated, split, passed to functions,...)
- Atoms
 - The list (to be or not to be) has 4 atoms
 - to, be, or, not
 - Functions are provided for manipulation of atoms
- Lists of lists
 - ((to be or not to be) (that is the question))

11

Programs Are Lists

- Programs are also represented as lists
 - (make-table text nil)
 - Can be a list
 - with atoms make-table, text, and nil
 - Can be a function
 - 'make-table' with 2 arguments
- How do we tell apart the program from a data list?
 - Quoted lists are not interpreted:
 - (set 'text '(to be or not to be))
 - Unquoted ones are interpreted
 - (set 'text (to be or not to be))
(function: fo)

12

Implications?

- If programs are lists
 - and data is also list
 - then we can generate a list that can be interpreted as a program
- In other words
 - We can write a program to write and execute another program
 - Useful in artificial intelligence
- Reductive aspects?

13

LISP Is Interpreted

- Most LISP systems provide interactive interpreters
 - One can enter commands into the interpreter, and the system will respond
- ```
> (plus 2 3)
5
> (eq (plus 2 3) (difference 9 4))
t
 (means 'true')
```

14

## Pure vs Pseudo-Functions

- Pure functions
  - plus, eq, ...
  - Only effect is the computation of a value
- Pseudo-functions
  - Has *side-effect*; more like a procedure
  - set
    - (set 'text '(to be or not to be))
    - Side effect:
      - Sets the value of *text* to (to be or not to be)
    - Return value:
      - (to be or not to be)

15

## Data Structures

- Primitives
  - Numbers
    - Operations: plus, minus, times, eq, etc.
  - Non-numeric atoms
    - Strings of characters used as symbols
      - Much like enumerated types in Pascal
      - Not used as strings
    - Operations: eq
    - Special atoms
      - t: true
      - nil: false; non-existent atom; empty list

16