V22.0490.001
Special Topics: Programming Languages

B. Mishra
New York University.

Lecture # 14

—Slide 1—

How LISP Works

- Lisp can be used as an *interpretive* or a *compiled* language
- **Interpreter**

  When a lisp program is interpreted, the lists that represent the functions are examined by `APPLY` and `EVAL`

  `APPLY` and `EVAL`

  1. Cooperate to walk recursively down the list structure
  2. Associate symbols and values to handle argument lists when functions are entered and to handle `LAMBDA` argument lists in internal `LAMBDA` application
  3. (Other variable introduction mechanisms, e.g., `LET` and `DO`, are implemented via `LAMBDA`

—Slide 2—

LISP Compiler

- Similar to the Interpreter
- **Compiler**

1. An assembly language program is produced
2. *Semantically equivalent* to what would be achieved by interpreting the functions
3. Resulting code is highly efficient
   Speed up of factor of 20 or higher

—Slide 3—

LISP Storage Layout

Address space is divided into <u>three</u> parts:

- **Stack**

  Temporary variables and return addresses from function calls

- **Binary Program Space**

  Actual machine instructions

- **Heap**

  Lisp data objects: Cons cells, symbols, numbers, strings, etc.

- Initially, a Lisp starts off with pieces of itself in all three parts

  User program provides additional objects into all three parts

—Slide 4—

LISP Internals

Internally, user objects are indistinguishable from objects in the internal Lisp system itself

- The stack and heap are made of **pointers**. A pointer typically consists of an address and few more bits of additional informations.
- Cons cell ≡ Pair of pointers
- A symbol ≡ Collection of Pointers pointing to
  a) Its value
  b) Its print name (a string)
  c) Its property list
  d) Its functional definition

—Slide 5—

Type of a LISP Object

A LISP object can be stored as:

- **BIBOP (BIg Bag Of Pages)**
  Location of the objects within the address space provides sufficient information. Each page of memory is dedicated to holding objects of a certain type.
- **Encoding in the Pointer (Tag bits)**
  Each pointer contains the address of a Lisp object and few tag bits specifying the type of the object pointed to
- **Encoding in the Object**
  The type of the object is encoded in the storage representation of the object itself. Thus type check has to access the object.

  (Usually this scheme is used in combination with the preceding tag bits encoding.)

—Slide 6—

`OBARRAY` *or* `OBLIST`

A vector of list of symbols
(Think of a Hash Table)

- Given the name of a symbol, first the string is "hashed" to compute its "hash number."
- The hash number is used to index into the `OBARRAY` to retrieve one of the lists of symbols.
- The list is searched to find the symbol and construct its cons cell representation.

—Slide 7—

Garbage Collection

During Lisp computation, periodically the heap becomes full and it is necessary to *free space* by identifying those cons cells that are *no longer accessible* (thus, usable).

A cons cell not accessible,

1. **if either there is no pointer to it**
2. **or all objects that point to it are themselves inaccessible.**

- Each cons cell has an extra bit ("**mark bit**") associated with it.
- At the start of the garbage collection, all mark bits are assumed to be "**off**."

—Slide 8—

Garbage Collection (contd)

- All the symbols and the stack are checked for pointers to cons cell. starting with these pointers, during a **mark phase**, every cons cell that is still accessible by the user's program has its mark bit set.

```
(DEFUN MARK-CELL (CONS-CELL)
   (COND ((NOT (MARKEDP CONS-CELL))
          (SETMARK CONS-CELL)
          (IF (CONSP (CAR CONS-CELL))
              (MARK-CELL (CAR CONS-CELL)))
          (IF (CONSP (CDR CONS-CELL))
              (MARK-CELL (CDR CONS-CELL))))))
```

—Slide 9—

Garbage Collection (contd)

- The heap is "**swept**." Each cons cell is examined with a linear sweep through the heap. If it is marked, the mark bit is turned **off** for the next round of garbage collection; otherwise, it is added to a free-list, `*FREE-LIST*`.

```
(DEFUN CHECK-CELL (CONS-CELL)
   (COND ((MARKEDP CONS-CELL)
          (UNSETMARK CONS-CELL))
         (T (RPLACD CONS-CELL *FREE-LIST*)
            (SETQ *FREE-LIST* CONS-CELL))))
```

—Last Slide—

Lisp Function with Garbage Collection

Now, we can write the LISP **CONS** function as follows:

```
(DEFUN CONS (X Y)
   (IF (NULL *FREE-LIST*)
       (GARBAGE-COLLECT))
   (LET ((CONS-CELL *FREE-LIST*))
        (SETQ *FREE-LIST* (CDR CONS-CELL))
        (RPLACA CONS-CELL X)
        (RPLACD CONS-CELL Y)
        CONS-CELL))
```

[End of Lecture #14]