CS 4100
LISP

April 20, 2012

Based on slides by Istvan Jonyer

Book by MacLennan

Chapters 9, 10, 11

Atoms

- LISP was written for AI
  - to represent complex relationships among objects
  - Objects can have many properties in real life; Atoms allow for modeling this
- Each atom comes with its own property list, and some built-in properties
  - pname (print name); mandatory
  - apval (applied value); to store data
    - If atom is bound to a value
  - expr (expression); to store program
    - If atom is bound to a program

Adding Properties to Atoms

- Other, arbitrary properties may also be added to an atom using *putprop* *(not in our clisp: setf)*

```
(putprop atom propValue propName)
(putprop ‘France ‘Paris ‘capital)
```

  - Paris, in this case, is also an atom
- Find out the value of a property using get

```
>(get ‘France ‘capital)
Paris
>(get ‘France ‘pname)
“France”
```

Special Property: apval

- Assigning a value to an atom

```
(set ‘Europe ‘(England France …))
```

  – is the same as

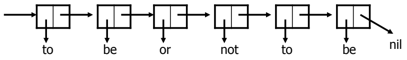
```
(putprop ‘Europe ‘(England France …)
  ‘apval)
```

‘Applied value’ points to the list the atom is bound to

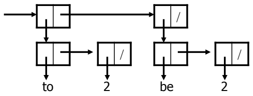
List Representation

- Lists are represented as linked lists

(to be or not to be)

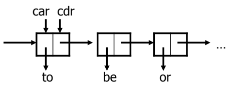


((to 2) (be 2))



Origins of car and cdr

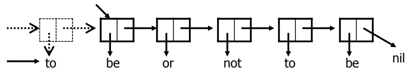
- First LISP was designed for the IBM 704
  - 1 word had 2 fields
    - Address field
    - Decrement field
  - car: “Content of Address part of Register”
  - cdr: “Content of Decrement part of Register”



Implementation of cons

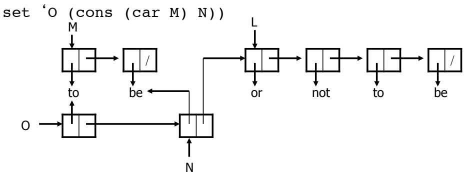
- car and cdr simply return the respective parts of the register
- cons has the job of constructing a new register using two pointers
  - Allocate new memory location
  - Fill in left and right parts of new location

    (cons ‘to ‘(be or not to be))



Sublists Can Be Shared

```
(set ‘L ‘(or not to be))
(set ‘M ‘(to be))
(set ‘N (cons (cadr M) L))
(set ‘O (cons (car M) N))
```



```
[10]> (set 'L '(or not to be))
(OR NOT TO BE)
[11]> (set 'M '(to be))
(TO BE)
[12]> (set 'N (cons (cadr M) L))
(BE OR NOT TO BE)
[13]> (set 'O (cons (car M) N))
(TO BE OR NOT TO BE)
```

List Structures Can Be Modified

- Functions discussed so far do not modify lists
- Modifying lists is possible via
  - replaca (replace address part)
  - replacd (replace decrement part)
- It is possible that more than one symbol points to a list
  - which can be modified using replaca and replacd
  - This can cause unexpected problems (like equivalence in Fortran)

Iteration by Recursion

- Iteration is done by recursion
- Iteration is mostly needed to perform an operation on every element of a list
  - This can be done using combination of
    - testing for end of list,
    - operating on first element, and
    - recursing on rest of the list

```
(defun plus-red (a)
  (if (null a)  nil
       (plus (car a) (plus-red (cdr a))) ))
```

  - Notice: No array bounds are needed! Function is very general

Iteration = Recursion

- Theoretically, recursion and iteration have the same power, and are equivalent
- One can be translated to the other (although may not be practical)
  - Recursion → iteration
    - Use iteration and keep track of auxiliary information in an explicit stack
  - Iteration → recursion
    - Need to pass control information (variables)

Storage Reclamation

- What happens to *cons*'d pointers that are no longer in use?
- Explicit reclamation is the obvious / traditional way
  - C: malloc, calloc, realloc, free
  - C++: new, delete
  - Pascal: new, dispose
- Issues
  - Complicates programming
    - Requires the programmer to keep track of pointers
  - Violates security of the environment
    - Memory freed, but still referenced (dangling pointers)

Automatic Storage Reclamation

- It would be nice for the system to automatically 'reclaim' storage no longer used
- System can keep track of number of references to storage
  - When references decrease to 0, storage is returned to 'free-list'
- Advantage:
  - Storage reclaimed immediately as last reference is destroyed
- Disadvantage:
  - Cyclic structures (points to itself) cannot be reclaimed

Garbage Collection

- A different approach is garbage collection
  - Do not keep track of references to location
  - When last reference is destroyed, we still do not do anything, and leave the memory as garbage (unused, non-reusable storage, littering the memory)
  - Collect garbage if system runs out of storage
    - Mark all areas unused
    - Then examine all visible pointers and mark storage they point to as 'used'
    - Leftover is garbage, and can be put on free-list
  - This is called the *mark-and-sweep* method

Garbage Collection

- Advantages
  - Fast until runs out of memory
  - No additional memory is needed for tracking references
- Disadvantages
  - Garbage collection itself can be slow
    - If memory is large, and have many references
    - Must halt entire system, since all dynamic memory must be marked as unused first
- Java uses this approach