1

Side Effects

Input and Output - Examples

```
--> (car (setq x (read)))
--> (a b c)          ; user types in
   a                 ; value of form
--> x
   (a b c)
--> (print 's)
   s              ; prints "s", then
   s              ; returns "s"
-->(print (+ 3 4 ))
   7
   7
-->(null (print (+ 3 4)))
   7
   nil
--> (defun test() (print (sqrt (read)))
                     (terpri))
   test
--> (test)
   16                ; user types in

   4.0               ; new line
   nil               ; returned by
                     ; terpri
```


©1993 John Mylopoulos

2

Input and Output

`(read)`

Read an s-expr from the terminal.
Returns the s-expr as (read)'s value.
(no arguments)

`(print` ***`s-expr`*** **`)`**

Prints *s-expr* on the terminal.
Returns the value of the s-expr.

`(terpri)`

Flushes the print buffer, and prints carriage return. Returns nil.

`(load` ***`file`*** **`)`**

Reads in the complete *file* , by reading and evaluating every expression in *`file`*, as if it had been entered at the terminal.
(Also will read in compiled Lisp).
Returns `nil` if the load failed.

©1993 John Mylopoulos

3

Lisp Functions are Non-Destructive

```
-> (setq   x  '(a b c))
(a b c)

-> (reverse  x)
(c b a)

-> x
(a b c)                          ;; Why???
```

`rplaca`, `rplacd` (REPLAce CAr, CDr)

These functions allow the assignment of a new value to the car/cdr field of a cons cell:

```
(setq x '(a b c))
(a b c)

(rplaca    x  '(v x y))
((v x y) b c)

-> x
((v x y) b c);; The value of x has been
              ;; changed by rplaca
```


4

`setf` (SET Field)

Can be used to assign a new value to a cons field, very much like an assignment statement of the form

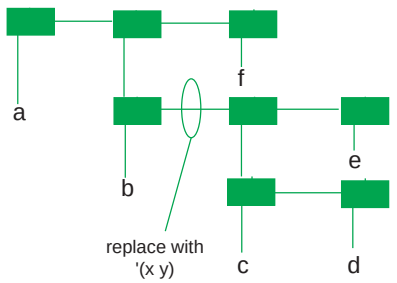
```
field(rec) <- newValue
```

in a conventional programming language.

```
(setf (car x) y) = (rplaca x y)

-> (setq   x  '(a (b (c d) e) f)
(a (b (c d ) e) f)

-> (setf      (cdadr x)  '(x y))
                                 ???
```



5

Garbage Collection

If Lisp functions work normally non-destructively, they must generate a lot of lists! Consider

```
->(reverse '(1 2 3 4 ..100))
                ;;a 100-element list
   (revloop '(1 2 3 ... 100) nil)
      (revloop '(2 3 4 ...) (1))
         (revloop '(3 4 ...) (2 1))
            (revloop '(4 5 ...) (3 2 1))
               ...
```

This sequence of calls will create lists of length 99 + 1, 98 + 2, 97 + 3, ..., 0 and 100 (200 lists in all!)

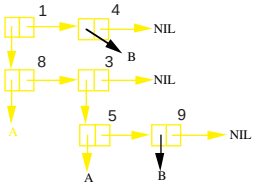
Space may be used very quickly! Most of these lists are discarded after their function call is completed. What happens to the space they occupy?

All Lisp implementations offer garbage collection as a means for reclaiming unused space.

6

How does garbage collection work?

Consider the list



A Lisp system partitions (part of) the space it has available into cons cells, to be used for lists such as the above. Assuming that a particular program has 10 available cells, this list might be represented as follows

ConsCell Space

1	8	4
2		
3	5	A3
4	A2	A3
5	A1	9
6		
7		
8	A1	3
9	A2	A3
10		

AtomSpace

1	A
2	B
3	NIL
4	
5	

7

How does garbage collection work?

Run down the atom space looking for atoms that have a value; for each such atom x do:

• If its value is an atom, mark that on the atom space as 'used'

• If the value is a list, mark every cons cell used to represent that list as 'used'

Iterate over the two stacks, and add every unused cell in an 'unused cons cell' and an 'unused atom cell' list respectively

ConsCell Stack

1	8	4
2		
3	5	A3
4	A2	A3
5	A1	9
6		
7		
8	A1	3
9	A2	A3
10		

AtomCell Stack

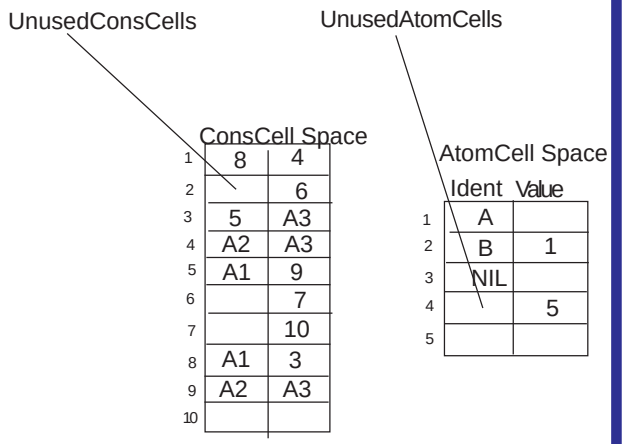
	Ident	Value
1	A	
2	B	1
3	NIL	
4		
5		

©1993 John Mylopoulos

8

How does garbage collection work?

Here is the result of garbage collection for the previous example:



©1993 John Mylopoulos

9

Programs as Sequences of Forms: `Do`

```
(DO (
  (<param1> <initialVal1> <updateForm1>)
  (<param2> <initialVal2> <updateForm2>)
   ...
  (<paramn> <initialValn> <updateFormn>))
 (<termination test>
    <intermedForms,if any> <resultForm>)
   <body>)
```

- `Do` expressions have three parts:
  - part 1 consists of a list of parameters that are all bound to initial values on entering the `Do`;
  - part 2 determines when the iteration will be terminated and what is returned;
  - part 3, the body, consists of forms that are evaluated sequentially.

- The use of `return` in `Do`: whenever an s-expression starting with `return` is encountered in the body of a `Do`, evaluation of the `Do` is terminated immediately; the value returned is the value of the form inside the `return` expression that stopped the evaluation.

©1993 John Mylopoulos

10

An Example

The following function does depth-first search of a tree structure:

```
;;Funtion search-df performs depth-first
;;search in a graph-structured state
;;space. The function maintains a list,
;;called OPENL, in which it places the
;;nodes to be examined, initially OPENL
;;contains the start node.

(defun search-df (start)
  (do((OPENL (list start)
       (append (successors (car OPENL)))
               (cdr OPENL))))
     ((null OPENL))  ;;test and return
     (cond ((goalp (car OPENL))
        (return (car OPENL)))) ;; 3: body
                                ))
```

- `GOALP` checkes whether a particular node is a goal node
- `successors` returns the list of successors of a tree node

©1993 John Mylopoulos

The `Prog` Special Form

The following functions evaluate all of their forms, in the order *form1 , form2,* up to and including *formi.*

```
(prog   form1 form2 ...  formi )
(prog1  form1 form2  ... formi )
(prog2  form1 form2  ... formi )
(progn  form1 form2  ... formi )
```

`prog1` returns the value of *form1.*
`prog2` returns the value of *form2.*
`progn` returns the value of the last form, *formi.*

The last three forms are more disciplined than the special form *prog* which relies on the functions `return` and `go,` and also treats atomic expressions as labels.

`def`

`def` does the same thing as `defun`!! But it is somewhat more "pure", and verbose. (not available in Common Lisp) For example the following two expressions are equivalent:

```
(defun  squared (x) (* x x))

(def squared  (lambda (x) (* x x)))
```


©1993 John Mylopoulos