1

Functional Programming with Lisp

©1993 John Mylopoulos

2

Functional Programming with Lisp

Table of Contents

1	Introduction	(12 slides)
2.	Forms	(12 slides)
3.	Functions	(11 slides)
4.	Variables	(7 slides)
5.	Efficiency Considerations	(9 slides)
6.	Scoping	(10 slides)
7.	Data Types	(8 slides)
8.	Side Effects	(9 slides)
9.	Functional Arguments	(13 slides)
10.	Programs as Data	(11 slides)

©1993 John Mylopoulos

3

Functions

A function `f` is a mapping
from one set, `D` (the domain)
to another set, `R`. (the range)
such that each element of the domain is mapped onto at most one element of the range.

More formally, let `f` be a set of ordered pairs in `D×R`; `f` is a function when for all $(x_1, y_1), (x_2, y_2) \in f$ if $y_1 \neq y_2$, then $x_1 \neq x_2$.

We say `f` is applied to an argument in `D` to return a value in `R`.

If `D` is a set of n-tuples, then `f` is a function of n arguments (or n-ary function)

For example, + is a function of 2 arguments returning their sum

Functional Composition

Given functions `f` and `g`, the functional composition of the two functions, denoted `f • g` , is a new function whose value for a given argument `x` is specified by

```
f • g(x) = f(g(x))
```


©1993 John Mylopoulos

4

A Function Evaluator

Lisp is an interactive system that takes as input a function and its arguments and returns the value of the function applied to those arguments.

```
% cl
Allegro CL 4.3 [SPARC; R1] (5/10/96 17:51)
Copyright (C) 1985-1996, Franz Inc.,
Berkeley, CA, USA. All Rights Reserved.
;; Optimization settings: safety 1, space
1, speed 1, debug 2.
...
USER(1):(+ 2  3)
5
USER(1):(exit)
; Exiting Lisp
```

Lisp Background

- Lisp is a list processing language
- Used extensively in Artificial Intelligence
- Invented by John McCarthy around 1960 at the MIT AI Lab.
- Permits functional and procedural programming.
- Many dialects.
- **Allegro Common Lisp** will be used mostly in the course.notes

©1993 John Mylopoulos

5

Symbolic Expressions

Everything in Lisp is represented by symbolic expressions called s-expressions or s-exprs.

An s-expr can be either an atom or a list of s-exprs.

Atoms

An atom is any word (sequence of characters) that doesn't look like a list.

```
x
+
university
Toronto
23-and-a-half
2+2=4
```

Numbers are also atoms.

```
255      22/7     23.45
-35      44/14    2.345E1
2.345E+1
```

Non-number atoms are also called symbols

©1993 John Mylopoulos

6

Lists

A list is represented in terms of zero or more s-exprs enclosed in parentheses and separated by one or more blanks.

```
(a b c)
(a (b c))
((a b) c)
(((((a)))))
()
```

The empty list is written as:

```
()
```

and is special, in the sense that it is treated as both a list and an atom, and can also be written as **`nil`**.

Parentheses are significant!!

`a` is different from `(a)`

`((b))` is different from `(b)`

©1993 John Mylopoulos

7

Operations on Lists

`first` (`car`) -- this is a function that takes a list and returns its first element. (In old Lisps known as `car`; that name can still be used and stands for "Contents Address Register").

```
first   of (a b c)     is a
first   of (a (b c))   is a
car     of ((a b) c)   is (a b)
first   of ((((a))))   is (((a)))
first   of ()          is nil      !!
first   of a           is undefined
```

`rest` (`cdr`) -- this is a function that takes a list and returns a list consisting of all but the first element. (In old Lisps known as `cdr`, that name can still be used. Short for Contents Data Register).

```
rest  of (a b c)     is (b c)
rest  of (a (b c))   is ((b c))
cdr   of ((a b) c)   is (c)
rest  of ((((a))))   is nil
rest  of ()          is nil        !!
rest  of a           is undefined
```


©1993 John Mylopoulos

8

Operations on Lists:

We often need to access the nth element of a list. Instead of writing

`(car (cdr (...(cdr X))...)` (n - 1 `cdr` calls)

we can write

`(cad...dr X)` (n - 1 d's)

More generally, a sequence of `car` and `cdr` calls can be written in shorthand form as a sequence of `a`'s and `d`'s

For example,

```
(car (cdr (cdr X))) = (caddr X)
```

In Common Lisp the function `nthcdr` applies `cdr` n times.

`(nthcdr 4 '(1 2 3 4 5))` returns `(5)`

©1993 John Mylopoulos

9

Operations on Lists

`cons` -- this is a function that takes two s-expr arguments, the second of which must be a list, and returns a **cons**tructed list whose *first* is the first argument, and whose *rest* is the second argument.

`cons` of `a` and `(b c)` is `(a b c)`
`cons` of `a` and `((b c))` is `(a (b c))`
`cons` of `(a b)` and `(c)` is _____
`cons` of `(((a)))` and `nil` is `((((a))))`
`cons` of `nil` and `nil` is _____
`cons` of `a` and `b` is *undefined*

©1993 John Mylopoulos

10

Predicates

Predicates are Boolean functions which return `t` (true) ot `nil` (false) depending on their arguments. Here are some basic ones:

`atom` -- takes an s-expr and returns `t` if it is an atom.

`atom` of `a` is `t`
`atom` of `nil` is _____
`atom` of `(a b)` is _____

`null` and **`not`** -- these predicates take an s-expr and return `t` if it is `nil`. Use whichever function name is more meaningful in a particular context.

`null` of `a` is `nil`
`null` of `nil` is _____
`null` of `()` is `t`
`null` of `(a b)` is _____
`not` of `a` is `nil`
`not` of `nil` is _____
`not` of `()` is `t`
`not` of `(a b)` is _____

In Common Lisp the predicate `endp` is like `null` but is only defined for list arguments

©1993 John Mylopoulos

11

Equality Predicates

There is a range of equality predicates for testing the equality of two s-exprs: `eq` (the simplest), `eql`, `equal`, and `equalp` (the most general). As well, there is an equality predicate, =, which compares numbers for equality.

`eq` takes two s-exprs and returns `t` if their values are stored in the same address. Generally this is only the case for atoms:

`eq`	of	`a`	and	`a`	is	`t`
`eq`	of	`a`	and	`b`	is	`nil`
`eq`	of	`45`	and	`45`	could be	`nil` or `t`
`eq`	of	`45`	and	`45.0`	is	`nil`
`eq`	of	`(a b)`	and	`(a b)`	is	`nil`

`eql` takes two s-exprs and returns `t` if they are numbers of the same type with the same value, or if they are `eq`. This is the usual equality test for atoms:

`eql`	of	`a`	and	`a`	is	`t`
`eql`	of	`a`	and	`b`	is	`nil`
`eql`	of	`45`	and	`45`	is	`t`
`eql`	of	`45`	and	`45.0`	is	`nil`
`eql`	of	`45.0`	and	`45.0`	is	`t`
`eql`	of	`(a b)`	and	`(a b)`	is	`nil`

©1993 John Mylopoulos

12

Equality Predicates

`equal` takes two s-exprs and returns `t` if they have the same atom or list value. Two lists are `equal` if their firsts are `equal` and their rests are also `equal`:

`equal`	of	`a`	and	`a`	is	`t`
`equal`	of	`a`	and	`b`	is	`nil`
`equal`	of	`45`	and	`45`	is	`t`
`equal`	of	`45`	and	`45.0`	is	`nil`
`equal`	of	`45.0`	and	`45.0`	is	`t`
`equal`	of	`(a b)`	and	`(a b)`	is	`t`

`=` takes two numbers and returns `t` if they are numerically equal. `=` may not be used on non-numbers.

`=`	of	`a`	and	`a`	is	*undefined*]
`=`	of	`45`	and	`45`	is	`t`
`=`	of	`45`	and	`45.0`	is	`t`
`=`	of	`45.0`	and	`45.0`	is	`t`
`=`	of	`(a b)`	and	`(a b)`	is	*undefined*

`equalp` is the most general equality predicate and works properly for symbols, numbers and lists

©1993 John Mylopoulos