

Last updated: 2022-07-28 23:58:24-04:00

Contents

<b>1</b>	<b>Building Abstractions with Procedures</b>	<b>1</b>
<b>A</b>	<b>Notes on LISP</b>	<b>2</b>
A.1	McCarthy 1960 . . . . .	2
A.2	LISP 1.5 Programmer’s Manual . . . . .	6
	<b>Index</b>	<b>7</b>

# 1 Building Abstractions with Procedures

**Definition (computational process).**

Abstract beings that inhabit computers.

**Definition (data).**

Computational processes manipulate other abstract things called ***data*** as they evolve.

**Definition (program).**

A pattern of rules by which the evolution of a computational process is directed.

**Definition (programming language).**

That in which programs are carefully composed from symbolic expressions that prescribe the tasks we want our computational processes to perform.

**Definition (bug, glitch).**

Small errors.

**Definition (debug).**

Remove bugs.

**Programming in Lisp**

See the [appendix](#).

# Appendix

## A Notes on LISP

### A.1 McCarthy 1960

*Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I*

LISP:

- **LIS**t **P**rocessor
- Developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T.
- Designed to facilitate experiments with a proposed system called the *Advice Tracker*:
  - a machine that could be instructed to handle declarative as well as imperative sentences and could exhibit “common sense” in carrying out its instructions.
  - originally proposed in November 1958.
- main requirement was a programming system for manipulating expressions representing formalized declarative and imperative sentences so that the Advice Taker system could make deductions.

**Definition (conditional statement).**

A ***conditional expression*** has the form

$$(p_1 \rightarrow e_1, p_2 \rightarrow e_2, \dots, p_n \rightarrow e_n)$$

where each  $p_i$  is a propositional expression and each  $e_i$  is an expression of any kind.

It is read “if  $p_1$  then  $e_1$ , else if  $p_2$  then  $e_2$ , else ... else if  $p_n$  then  $e_n$ .”

Determining the value, starting from  $i = 1$ :

- If  $p_i$  is undefined or if  $i = n$  and  $p_n$  is false, then the value is *undefined*.
- If  $p_i$  is true, then the value is  $e_i$ .
- If  $p_i$  is false, check  $p_{i+1}$ .

*Example.*

- $(1 < 2 \rightarrow 4, 1 > 2 \rightarrow 3) = 4$
- $(2 < 1 \rightarrow 4, 2 > 1 \rightarrow 3, 2 > 1 \rightarrow 2) = 3$
- $(2 < 1 \rightarrow 4, T \rightarrow 3) = 3$
- $(2 < 1 \rightarrow \frac{0}{0}, T \rightarrow 3) = 3$
- $(2 < 1 \rightarrow 3, 4 < 1 \rightarrow 4)$  is undefined.
- $(2 < 1 \rightarrow 3, 4 < 1 \rightarrow 4)$  is undefined.

*Example.*

- $|x| = (x < 0 \rightarrow -x, T \rightarrow x)$
- $\delta_{ij} = (i = j \rightarrow 1, T \rightarrow 0)$
- $\text{sgn}(x) = (x < 0 \rightarrow -1, x = 0 \rightarrow 0, T \rightarrow 1)$

*Example (recursive functions).*

- $n! = (n = 0 \rightarrow 1, T \rightarrow n \cdot (n + 1)!)$
- $\text{gcd}(m, n) = (m > n \rightarrow \text{gcd}(n, m), \text{rem}(n, m) = 0 \rightarrow m, T \rightarrow \text{gcd}(\text{rem}(n, m), m))$
- $\text{sqrt}(a, x, \epsilon) = (|x^2 - a| < \epsilon \rightarrow x, T \rightarrow \text{sqrt}(a, \frac{1}{2}(x + \frac{a}{x}), \epsilon))$

*Example (propositions).*

- $p \wedge q = (p \rightarrow q, T \rightarrow F)$
- $p \vee q = (p \rightarrow T, T \rightarrow q)$
- $\neg p = (p \rightarrow F, T \rightarrow T)$
- $p \supset q = (p \rightarrow q, T \rightarrow T)$

**Definition (function).**

A **function** has the form

$$\lambda((x_1, \dots, x_n), \mathcal{E})$$

where  $(x_1, \dots, x_n)$  is a list of  $n$  variables and  $\mathcal{E}$  is a form of those variables.

*Example.*

$$\lambda((x, y), y^2 + x)(3, 4) = 19$$

*Remark.*

Variables occurring in the list of variables are dummy or bound variables. We may change the names of the bound variables in a function expression without changing the value of the expression, provided that we make the same change for each occurrence of the variable and do not make two variables the same that previously were different.

*Example.*

$$\lambda((x, y), y^2 + x) = \lambda((u, v), v^2 + u)$$

**Definition ( $\text{label}(a, \mathcal{E})$ ).**

$\text{label}(a, \mathcal{E})$  denotes the expression  $\mathcal{E}$  provided that occurrences of  $a$  within  $\mathcal{E}$  are to be interpreted as referring to the expression as a whole.

*Example.*

$\lambda$ -notation is inadequate for naming functions defined recursively. For example, we can convert

the definition

$$\text{sqrt}(a, x, \epsilon) = (|x^2 - a| < \epsilon \rightarrow x, T \rightarrow \text{sqrt}(a, \frac{1}{2}(x + \frac{a}{x}), \epsilon))$$

into

$$\text{sqrt} = \lambda((a, x, \epsilon), (|x^2 - a| < \epsilon \rightarrow x, T \rightarrow \text{sqrt}(a, \frac{1}{2}(x + \frac{a}{x}), \epsilon)))$$

but the right-hand side cannot serve as an expression for the function because there would be nothing to indicate that the reference to `sqrt` within the expression stood for the expression as a whole.

The *label* notation makes explicit what symbol in  $\mathcal{E}$  should refer to the statement itself. Thus we can write

$$\text{label}(\text{sqrt}, \lambda((a, x, \epsilon), (|x^2 - a| < \epsilon \rightarrow x, T \rightarrow \text{sqrt}(a, \frac{1}{2}(x + \frac{a}{x}), \epsilon))))$$

as a name for our `sqrt` function.

### Definition (atomic symbol).

A string starting with one of  $\{A, B, \dots, Z\}$  and continuing with zero or more of  $\{A, B, \dots, Z, 0, 1, \dots, 9\}$ .

### Definition (S-expression).

1. Atomic symbols are S-expressions.
2. If  $e_1$  and  $e_2$  are S-expressions, so is  $(e_1 \cdot e_2)$ .

*Note.*

‘S’ stands for ‘symbolic’.

*Notation.*

The list

$$(x_1, x_2, \dots, x_n)$$

is represented by the S-expression

$$(x_1 \cdot (x_2 \cdot (\dots \cdot (x_n \cdot \text{NIL}) \dots)))$$

*Example.*

$$((A, B), C, D \cdot E) = ((A \cdot (B \cdot \text{NIL})) \cdot (C \cdot ((D \cdot E) \cdot \text{NIL})))$$

### Definition (M-expression).

An expression representing a function of S-expressions.

*Notation.*

In order to clearly distinguish the expressions representing functions from S-expressions, we shall use sequences of lower-case letters for function names and variables ranging over the set of S-expressions. We also use brackets and semicolons, instead of parentheses and commas, for denoting the application of functions to their arguments.

*Example.*

- $car[x]$
- $car[cons[(A \cdot B); x]]$

*Note.*

‘M’ stands for ‘meta’.

**Definition ( $atom[x]$ ).**

$atom[x]$  has the value of True or False according to whether  $x$  is an atomic symbol.

*Example.*

- $atom[X] = T$
- $atom[(X \cdot A)] = F$

**Definition ( $eq[x; y]$ ).**

$eq[x; y]$  is defined if and only if both  $x$  and  $y$  are atomic.  $eq[x; y] = T$  if  $x$  and  $y$  are the same symbol, and  $eq[x; y] = F$  otherwise.

*Example.*

- $eq[X; X] = T$
- $eq[X; A] = F$
- $eq[X; (X \cdot A)]$  is undefined.

**Definition ( $car[x]$ ).**

$car[x]$  is defined if and only if  $x$  is not atomic.  $car[(e_1 \cdot e_2)] = e_1$ .

*Example.*

- $car[(X \cdot A)] = X$
- $car[((X \cdot A) \cdot Y)] = (X \cdot A)$
- $car[X]$  is undefined.

**Definition ( $cdr[x]$ ).**

$cdr[x]$  is also defined when  $x$  is not atomic.  $cdr[(e_1 \cdot e_2)] = e_2$ .

*Example.*

- $cdr[(X \cdot A)] = A$
- $cdr[((X \cdot A) \cdot Y)] = Y$
- $cdr[X]$  is undefined.

**Definition (*cons*[*x*; *y*]).**

*cons*[*x*; *y*] is defined for any *x* and *y*.  $\text{cons}[e_1; e_2] = (e_1 \cdot e_2)$ .

*Example.*

- $\text{cons}[X; A] = (X \cdot A)$
- $\text{cons}[(X \cdot A); Y] = ((X \cdot A) \cdot Y)$

*Example.*

- $\text{car}[\text{cons}[x; y]] = x$
- $\text{cdr}[\text{cons}[x; y]] = y$
- $\text{cons}[\text{car}[x]; \text{cdr}[x]] = x$  for non-atomic *x*

## A.2 LISP 1.5 Programmer's Manual

*LISP 1.5 Programmer's Manual*

## Index

*atom*[*x*], [5](#)

*car*[*x*], [5](#)

*cdr*[*x*], [5](#)

*cons*[*x*; *y*], [6](#)

*eq*[*x*; *y*], [5](#)

*label*(*a*,  $\mathcal{E}$ ), [3](#)

atomic symbol, [4](#)

bug, [1](#)

computational process, [1](#)

conditional statement, [2](#)

data, [1](#)

debug, [1](#)

function, [3](#)

glitch, [1](#)

M-expression, [4](#)

program, [1](#)

programming language, [1](#)

S-expression, [4](#)