Last updated: 2022-07-29 15:49:23-04:00

Contents

1 Building Abstractions with Procedures **1**

A Notes on LISP **2**
A.1 McCarthy 1960 2
A.2 LISP 1.5 Programmer's Manual 9

Index **10**

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

$$(p_1 \to e_1, p_2 \to e_2, \cdots, p_n \to e_n)$$

where each p_i is a propositional expression and each e_1 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.
- $(1 < 2 \to 4,\ 1 > 2 \to 3) = 4$
- $(2 < 1 \to 4,\ 2 > 1 \to 3,\ 2 > 1 \to 2) = 3$
- $(2 < 1 \to 4,\ T \to 3) = 3$
- $(2 < 1 \to \frac{0}{0},\ T \to 3) = 3$
- $(2 < 1 \to 3,\ 4 < 1 \to 4)$ is undefined.
- $(2 < 1 \to 3,\ 4 < 1 \to 4)$ is undefined.

Example.

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

Example (recursive functions).

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

Example (propositions).

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

Definition (function).

A ***function*** has the form

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

where $(x_1, \ldots, x_n)$ 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 ($label(a, \mathcal{E})$).

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

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

the definition

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

into

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

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

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

as a name for our sqrt function.

Definition (atomic symbol).
A string starting with one of $\{A, B, \ldots, Z\}$ and continuing with zero or more of $\{A, B, \ldots, Z, 0, 1, \ldots, 9\}$.

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

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

is represented by the S-expression

$$(x_1 \cdot (x_2 \cdot (\ldots \cdot (x_n \cdot NIL) \ldots)))$$

Example.
$((A, B), C, D \cdot E) = ((A \cdot (B \cdot NIL)) \cdot (C \cdot ((D \cdot E) \cdot NIL)))$

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. *cons*[$e_1; e_2$] = $(e_1 \cdot e_2)$.

Example.

- *cons*[$X; A$] = $(X \cdot A)$
- *cons*[$(X \cdot A); Y$] = $((X \cdot A) \cdot Y)$
- *car*[*cons*[$x; y$]] = x
- *cdr*[*cons*[$x; y$]] = y
- *cons*[*car*[x]; *cdr*[x]] = x for non-atomic x

Definition (`ff[x]`).

```
ff[x] = [
    atom[x] -> x;
    T -> ff[car[x]]
]
```

Returns the first atomic symbol of x.

Example.

```
ff[((A.B).C)] ->
    ff[(A.B)] ->
    ff[A] ->
    atom[A] ->
    A
```

Definition (`subst[x;y;z]`).

```
subst[x;y;z] = [
    atom[z] -> [
        eq[z;y] -> x;
        T -> z
    ];
    T -> cons[
        subst[x;y;car[z]];
        subst[x;y;cdr[z]]
    ]
]
```

Returns a copy of the S-expression z such that every occurrence of y in z has been replaced with x.

Example.
subst[$(X \cdot A);\ B;\ ((A \cdot B) \cdot C)$] = $((A \cdot (X \cdot A)) \cdot C)$

Definition (`equal[x;y]`).

```
equal[x;y] = [
    [atom[x] AND atom[y] AND eq[x;y]] OR
    [-atom[x] AND -atom[y] AND equal[car[x];car[y]] AND equal[cdr[x];cdr[y]]]
]
```

Returns True or False according to whether x and y are equivalent.

Definition (`null[x]`).

```
null[x] = [
    atom[x] AND eq[x;NIL]
]
```

Returns True or False according to whether x is the atomic symbol NIL.

Notation.

When navigating through nested S-expressions, an expression such as

$$car[cdr[cdr[car[cdr[x]]]]]$$

could occur. As a shorthand, this example expression can be written as

$$caddadr[x]$$

Notation.

Another useful abbreviation is to write

$$list[e_1; e_2; \ldots; e_n]$$

for

$$cons[e_1; cons[e_2; cons[\ldots; cons[e_n; NIL]]]]$$

Definition (`append[x;y]`).

```
append[x;y] = [
    null[x] -> y;
    T -> cons[car[x];append[cdr[x];y]]
]
```

Given lists x and y, return a copy of x such that the NIL that marks the end of the list x is substituted with the list y, which effectively appends y onto x.

Definition (`among[x;y]`).

```
among[x;y] = [
    -null[y] AND [
        equal[x;car[y]] OR among[x;cdr[y]]
```

```
    ]
]
```

Returns True or False according to whether x is among the members of the list y.

Definition (`pair[x;y]`).

```
pair[x;y] = [
    null[x] AND null[y] -> NIL;
    -atom[x] AND -atom[y] -> cons[
        list[car[x];car[y]];
        pair[cdr[x];cdr[y]]
    ]
]
```

Returns a list whose ith member is a list (X_i, Y_i) where X_i is the ith member of x and Y_i is the ith member of y.

Example.

- $pair[(A, B, C); (X, (Y, Z), U)] = ((A, X), (B, (Y, Z)), (C, U))$
- $pair[(A, B); (C, D, E)]$ is undefined. Eventually $x = NIL$ and $y = (E \cdot NIL)$. Since $y \neq NIL$ and x is atomic, the conditions of $pair[(A, B); (C, D, E)]$ are all false.

Definition (`assoc[x;y]`).

```
assoc[x;y] = [
    eq[caar[y];x] -> cadar[y];
    T -> assoc[x;cdr[y]]
]
```

Given that y is a list of key-value pairs (k_i, v_i), if x matches one of the keys k_i then return the corresponding value v_i.

Example.

$assoc[X; ((W, (A, B)), (X, (C, D)), (Y, (E, F)))] = (C, D)$

Definition (`sublis[x;y]`, `sub2[x;z]`).

```
sublis[x;y] = [
    atom[y] -> sub2[x;y];
    T -> cons[
        sublis[x;car[y]];
        sublis[x;cdr[y]]
    ]
]

sub2[x;z] = [
    null[x] -> z;
```

```
    eq[caar[x];z] -> cadar[x];
    T -> sub2[cdr[x];z]
]
```

Given x is a list of key-value pairs (k_i, v_i) where each k_i is atomic, $sublis[x; y]$ returns a copy of y such that for every atomic symbol z in y, if z is a key k_i of x then substitute z for the corresponding value v_i.

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
`among[x;y]`, 7
`append[x;y]`, 7
`assoc[x;y]`, 8
`equal[x;y]`, 7
`ff[x]`, 6
`null[x]`, 7
`pair[x;y]`, 8
`sub2[x;z]`, 8
`sublis[x;y]`, 8
`subst[x;y;z]`, 6

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