Simple Lisp

LISP is over half a century old and it still has this perfect, timeless aura about it.

I wonder if the cycles will continue forever.

A few coders from each new generation rediscovering the LISP arts.

These are your father's parentheses.

Elegant weapons for a more... civilized age.

Alonzo Church

John McCarthy
Simple Lisp

See the class website for a pdf version.

This is a very interesting book by Gregory Chaitin! It has to do with “Algorithmic Information Theory” (Information Compression and Randomness) (also known as “Minimum Description Length”) which I think is a very interesting topic. There is a small section on lisp that I’d like you to read (i.e., pages 38 – 44 of the pdf version). DrScheme code that goes along with the reading starts on the next slide. And, if you like, you can read the entire book to feed your intellectual curiosity :-) .
Simple Lisp in Scheme

**Code for Chaitin page 40**

(if true (+ 1 2) (+ 3 4))
→ 3

(if false (+ 1 2) (+ 3 4))
→ 7

**Code for Chaitin page 41**

Instead of (' (a b c)) → (a b c)
'(a b c)
→ (list 'a 'b 'c)

(if (= 23 32) true false)
→ False

(if (= (list 1 2 3) (list 1 2 3)) true false)
→ . . =: expects type <number> as 1st argument, given: (list 1 2 3); other arguments were: (list 1 2 3)

Instead of (if (atom ...)
  (if (list? (list 1 2 3)) true false)
→ true
  (if (list? 21) true false)
→ false
  (if (list? 'a) true false)
→ false
Simple Lisp in Scheme

**Code for Chaitin page 41 continued**

Instead of (let n (+ 1 2) (* n 3))

(let ((n (+ 1 2))) (* n 3))
→ 9

Instead of (let (f n) (* n n) (f 10)) – see Scheme’s definition of “let” in the Scheme Tutorial at [http://www.ccs.neu.edu/home/dorai/t-y-scheme/t-y-scheme-Z-H-7.html#node_idx_274](http://www.ccs.neu.edu/home/dorai/t-y-scheme/t-y-scheme-Z-H-7.html#node_idx_274)

(let ((f (lambda (n) (* n n)))) (f 10))
→ 100

**Code for Chaitin page 42**

Instead of (car (‘ (a b c )))

(car ’(a b c))
→’a

Instead of (cdr (‘ (a b c )))

(cdr ’(a b c))
→(list ’b ’c)

Instead of (cons (‘ a) (‘ (b c )))

(cons ’a ’(b d))
→ (list ’a ’b ’d)
Simple Lisp in Scheme

**Code for Chaitin page 43**

Instead of (let (factorial N) (if (= N 0) 1 (* N (factorial (- N 1)))) (factorial 5)) – see Scheme’s definition of “letrec” in the Scheme Tutorial at [http://www.ccs.neu.edu/home/dorai/t-y-scheme/t-y-scheme-Z-H-8.html#node_idx_288](http://www.ccs.neu.edu/home/dorai/t-y-scheme/t-y-scheme-Z-H-8.html#node_idx_288)

(letrec ((factorial (lambda (N) (if (= N 0) 1 (* N (factorial (- N 1))))))) (factorial 5))

→ 120

(letrec ((factorial (lambda (N) (if (= N 0) 1 (* N (factorial (- N 1))))))) (factorial 100))

→ 9332621544394415268169923885626670049071596826438162146859296389521759999322991560894146397615651828625369792082722375825118521091686400000000000000000000000

-------------------

**More interesting code:**

(letrec ((first (lambda (List) (if (null? List) (list) (car List))))) (first (list 1 2 3)))

(letrec ((rest (lambda (List) (if (null? List) (list) (cdr List))))) (rest (list 1 2 3)))

(letrec ((sum-list (lambda (List) (if (null? List) 0 (+ (car List) (sum-list (cdr List))))))) (sum-list (list 1 2 3)))

(letrec ((nth (lambda (N List) (if (not (= N 0))(nth (- N 1) (cdr List))(car List))))) (nth 2 (list 1 2 3)))

(letrec ((head (lambda (N List) (if (= N 0) (list) (cons (car List) (head (- N 1) (cdr List))))))) (head 3 (list 1 2 3 4 5)))
Simple Lisp in Scheme

(letrec ( (first (lambda (List) (if (null? List) (list) (car List)))))
  (sum-list (lambda (List) (if (null? List) 0 (+ (car List) (sum-list (cdr List))))))
  (nth (lambda (N List) (if (not (= N 0))(nth (- N 1) (cdr List))(car List))))
  (head (lambda (N List) (if (= N 0) (list) (cons (car List) (head (- N 1) (cdr List)))))))
  (nth 1 (list 1 2 3)))
→ 2

(letrec ( (List (list 1 2 3 4 5 6))
  (first (lambda (List) (if (null? List) (list) (car List)))))
  (sum-list (lambda (List) (if (null? List) 0 (+ (car List) (sum-list (cdr List))))))
  (nth (lambda (N List) (if (not (= N 0))(nth (- N 1) (cdr List))(car List))))
  (head (lambda (N List) (if (= N 0) (list) (cons (car List) (head (- N 1) (cdr List)))))))
  (head (nth 1 List) List) )
→ (list 1 2)

**Code for Chaitin page 43 - 44**

(letrec ( (map (lambda (Function List) (if (null? List) List (cons (Function (car List)) (map Function (cdr List)))))) )
  (factorial (lambda (N) (if (= N 0) 1 (* N (factorial (- N 1)))))) )
  (map factorial (list 4 1 2 3 5))
→(list 24 1 2 6 120)

**Define statement:**

(define nth (lambda (N List) (if (not (= N 0))(nth (- N 1) (cdr List))(car List))))
(nth 2 (list 1 2 3 4 5))
→ 3
A little Bit of Lambda Calculus – Y Combinator in Scheme

Are these really the same?

(letrec ((factorial (lambda (N) (if (= N 0) 1 (* N (factorial (- N 1))))))) (factorial 50))

⇒ 30414093201713378043612608166064768844377641568960512000000000000

-----------------------------------------------

(((lambda (X)
  ((lambda (procedure)
    (X (lambda (arg) ((procedure procedure) arg))))
  (lambda (procedure)
    (X (lambda (arg) ((procedure procedure) arg))))))
)(lambda (func-arg)
  (lambda (n)
    (if (zero? n)
      1
      (* n (func-arg (- n 1)))))) 50)

⇒ 30414093201713378043612608166064768844377641568960512000000000000

For more details see Section 22.4 of the textbook.

(define make-recursive-procedure
  (lambda (p)
    ((lambda (f)
      (f f))
     (lambda (f)
       (p (f f))))))
The function *Square* has \( \mathbb{R} \) (the reals) as domain and range.

\[
\text{Square} : \mathbb{R} \rightarrow \mathbb{R}
\]

\[
\text{Square}(n) = n^2
\]

A lambda expression is a particular way to define a function:

\[
\text{LambdaExpression} \rightarrow \text{variable} \mid (M N) \mid (\lambda \text{variable} . M)
\]

\[
M \rightarrow \text{LambdaExpression}
\]

\[
N \rightarrow \text{LambdaExpression}
\]

E.g., \((\lambda x . x^2)\) represents the *Square* function.
In $(\lambda x . M)$, $x$ is bound. Other variables in $M$ are free. A substitution of $N$ for all occurrences of a variable $x$ in $M$ is written $M[x \leftarrow N]$. Examples:

\[
\begin{align*}
x[x \leftarrow y] &= y \\
(xx)[x \leftarrow y] &= (yy) \\
(zw)[x \leftarrow y] &= (zw) \\
(zx)[x \leftarrow y] &= (zy) \\
(\lambda x \cdot (zx))[x \leftarrow y] &= (\lambda u \cdot (zu))[x \leftarrow y] = (\lambda u \cdot (zu)) \\
(\lambda x \cdot (zx))[y \leftarrow x] &= (\lambda u \cdot (zu))[y \leftarrow x] = (\lambda u \cdot (zu))
\end{align*}
\]

• An alpha-conversion allows bound variable names to be changed. For example, alpha-conversion of $\lambda x . x$ might yield $\lambda y . y$.

• A beta reduction $((\lambda x . M)N)$ of the lambda expression $(\lambda x . M)$ is a substitution of all bound occurrences of $x$ in $M$ by $N$. E.g., $((\lambda x . x^2) \ 5) = 5^2$
Composition of Relations

**Composing Relations.** Suppose that $R$ and $S$ are relations on $A$. The composition $R \circ S$ of $R$ and $S$ is the relation on $A$ that is defined by

$$x(R \circ S)z \equiv \exists y \in A(xRy \land ySz).$$

Furthermore, for $n \in \mathbb{N}, n \geq 1$ we define $R^n$ by means of $R^1 := R$, $R^{n+1} := R^n \circ R$.

*REL> r
\{(1,2),(2,3),(2,4),(2,5),(2,6),(6,7),(6,8)\}

*REL> repeatR r 2
\{(1,3),(1,4),(1,5),(1,6),(2,7),(2,8)\}

*REL> repeatR r 3
\{(1,7),(1,8)\}

*REL> repeatR r 4
\{\}

composed with $\{(1,2),(2,3),(2,4),(2,5),(2,6),(6,7),(6,8)\}$

composed with $\{(1,3),(1,4),(1,5),(1,6),(2,7),(2,8)\}$

composed with $\{(1,7),(1,8)\}$

composed with $\{(1,2),(2,3),(2,4),(2,5),(2,6),(6,7),(6,8)\}$
Composition of Relations

**Composing Relations.** Suppose that $R$ and $S$ are relations on $A$. The
*composition* $R \circ S$ of $R$ and $S$ is the relation on $A$ that is defined by

$$x(R \circ S)z \equiv \exists y \in A(xRy \land ySz).$$

Furthermore, for $n \in \mathbb{N}, n \geq 1$ we define $R^n$ by means of $R^1 := R,
R^{n+1} := R^n \circ R$.

*REL> r
{(1,2),(2,3),(2,4),(2,5),(2,6),(6,7),(6,8)}

*REL> repeatR r 2
{(1,3),(1,4),(1,5),(1,6),(2,7),(2,8)}

*REL> repeatR r 3
{(1,7),(1,8)}

*REL> repeatR r 4
{}

{(rose,phil),(phil,nicolette),(phil,antoinette),(phil,jeanette),
(phil,philJ),(philJ,philJJ),(philJ,Patrick)}

{(rose,nicolette),(rose,antoinette),(rose,jeanette),(rose,philJ),
(phil,philJJ),(phil,patrik)}  **Grandparent Relation**

{(rose,philJJ),(rose,patrick)}
A little Bit of Lambda Calculus – Lambda Calculus Arithmetic

zero  (lambda (f ) (lambda (x) x))
one  (lambda (f ) (lambda (x) (f x)))
two  (lambda (f ) (lambda (x) (f (f x))))
i.e., in Scheme - ((lambda (f ) ((lambda (x) (f (f x))) 4)) (lambda (z) (+ z z)))
three  (lambda (f ) (lambda (x) (f (f (f x)))))
i.e., in Scheme - ((lambda (f ) ((lambda (x) (f (f (f x)))) 4)) (lambda (z) (+ z z)))
succ
  (lambda (n)
   (lambda (f )
    (lambda (x)
     (f ((n f ) x)))))
i.e., in Scheme -
   ((lambda (n) (n ((lambda (f ) ((lambda (x) (f (f x))) 4)) (lambda (z) (+ z z))) )) (lambda (z) (+ z z)))
or ((lambda (n) ((lambda (z) (+ z z)) n)) ((lambda (f ) ((lambda (x) (f (f x))) 4)) (lambda (z) (+ z z))))
or (define succ(lambda (n) ((lambda (z) (+ z z)) n)))
   (succ ((lambda (f ) ((lambda (x) (f (f x))) 4)) (lambda (z) (+ z z)) )
sum
  (lambda (m)
   (lambda (n)
    ((n succ) m)))
prod
  (lambda (m)
   (lambda (n)
    ((n (sum m)) zero)))

For more details see Section 22.3 of the textbook.
Notion of Truth - Gödel's Incompleteness Theorem

There are Predicate Logic Statements that are True that can’t be proved True (Incompleteness) or there are Predicate Logic Statements that can be proved True that are actually False (Unsoundness).