



LINGUAGEM SCHEME

Alunos:

Gabriel Luciano
Geovane Fonseca
Isabelle Langkammer
Luigi Domenico



SUMÁRIO

1. Histórico
2. Paradigmas
3. Características marcantes
4. Linguagens semelhantes
5. Considerações finais
6. Bibliografia



HISTÓRICO

Introdução

- Linguagem desenvolvida entre os anos de 1975 e 1980
- Laboratório de Inteligência Artificial e Ciência da Computação do MIT
- Desenvolvida por Guy Steele e Gerald Sussman

Autores

- Gerald Jay Sussman
- Professor de Engenharia Elétrica no MIT
- Bacharelado: Matemática (MIT, 1968)
- Ph.D: Matemática (MIT, 1973)
- Coautor do livro de introdução a Ciência da Computação SICP (*Structure and Interpretation of Computer Programs*)
- Inventor da linguagem de programação Scheme





Autores

- Guy L. Steele Jr.
- Bacharelado: Matemática Aplicada (Harvard, 1975)
- Mestrado: Ciência da Computação (MIT, 1977)
- Ph.D: Ciência da Computação (MIT, 1980)
- Inventor da linguagem de programação Scheme





Influências

- LISP
- ALGOL
- Cálculo Lambda
- Teoria dos Atores - Carl Hewitt



Versões

- Scheme 78
- *Revised n Report on the Algorithmic Language Scheme (RnRS)*
- *Institute of Electrical and Electronics Engineers (IEEE)*



PARADIGMAS

Imperativo

Definição: Descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa

Principais comandos que definem o paradigma na linguagem:

- Construtores Imperativos
- Procedimentos mutadores de lista
- Procedimentos mutadores de String



Imperativo

Exemplo: Construtor Imperativo

a) Atribuição *set!*:

```
(define x 10)  
(set! x (+ x 1))
```

b) Construtor *begin*:

```
(define x 0)  
(if (= x 0)  
    (begin (set! x (+ x 1))  
           x))
```



Imperativo

Exemplo: Mutadores de Lista

- a) Mutador de Lista *set-car!*:
(**define** lst '(0 1 2 3 4 5))
(set-car! lst 7)

- b) Mutador de Lista *set-cdr!*:
(**define** lst '(0 1 2 3 4 5))
(set-cdr! lst 7)



Funcional

Definição: Trata a computação como avaliação de funções matemáticas e evita estados ou dados mutáveis

Principal característica em Scheme: **Técnica Currying**

Onde:

Dada uma função $f: (X \times Y) \rightarrow Z$, utilizando a técnica de *Currying*, teremos:

$$\text{curry}(f): X \rightarrow (Y \rightarrow Z)$$



Funcional

Exemplo:

```
(define (map function lst)
  (cond ((null? lst) '())
        (else (cons (function (car lst))
                      (map function (cdr lst))))))
(define (double x) (* 2 x))
```

```
(define lst (list 1 2 3 4 5))

(display lst)
(display (map double lst))
```



Orientado a Objetos

Definição: Modelo de programação de software baseado na composição e interação entre diversas unidades chamadas de objetos

Principais característica em Scheme:

- a) Possibilidade de interpretar uma *closure* como um objeto
- b) Considerar funções como variáveis de 1ª classe
- c) Amarração estática das variáveis livres



Orientado a Objetos

Template de uma classe:

```
(define (class-name construction-parameters)
  (let ((instance-var init-value)
        ...)
    (define (method parameter-list)
      method-body)
    ...
    (define (self message)
      (cond ((eqv? message selector) method)
            ...
            (else (error "Undefined message " message))))
    self))
```



Orientado a Objetos

Funções auxiliares:

```
(define (new-instance class . parameters)  
  (apply class parameters))
```

```
(define (send message object . args)  
  (let ((method (object message)))  
    (cond ((procedure? method) (apply method args))  
          (else (error "Error in method lookup " method))))))
```



CARACTERÍSTICAS

Família LISP

- Homoicônicos
- Macros
- Condição de reinicialização
- *Call with current continuation*



Macro - exemplo em Scheme

```
(define-macro (while condition . body)
  `(let loop ()
     (cond (condition
            (begin . body)
            (loop)))))
```

```
(let ((i 0))
  (while (< i 10)
    (display i)
    (newline)
    (set! i (+ i 1))))
```



CARACTERÍSTICAS

Scheme

- *Quote, quasiquote, unquote e unquote-splicing*
- Tipagem forte
- Tipagem dinâmica
- *Closure*
- *Tail Recursion*
- *High-order function*
- *Hygienic macro*



Quote - exemplo

```
(display (+ 1 2))  
(newline)  
(display (quote (+ 1 2)))  
(newline)  
(display '(+ 1 2))
```

Saída: 3

```
(+ 1 2)  
(+ 1 2)
```



Quasiquote e unquote - exemplo

```
(define str '(Hello world))  
(display `(,str))
```

Saída: ((Hello world))



Quasiquote e unquote-splicing - exemplo

```
(define str '(Hello world))  
(display `(@str))
```

Saída: (Hello world)



Tipagem forte - exemplo

Scheme:

```
(define (addOne x)
  (+ x 1))
```

```
(display (addOne 10))
(display (addOne "10"))
```

Error

+: contract violation

expected: number?

given: "10"

argument position: 1st

C:

```
#include "stdio.h"
```

...

```
int main(void) {
  printf("%d\n", addOne(10));
  printf("%d\n", addOne("10"));
  return 0;
}
```

Saída:

11

4195785



Tipagem dinâmica - exemplo

Scheme:

```
(define value 10)
(display value)
(newline)
```

```
(set! value "Hello world")
(display value)
(newline)
```

Saída:

```
10
Hello world
```

Go:

```
package main
import "fmt"
func main() {
    i := 10
    fmt.Println(i)
    i = "Hello world"
}
```

Error: cannot use "Hello world" (type string) as type int in assignment



Closure - exemplo

Scheme:

```
(define fibonacci
  (let ((memo (make-eq-hashtable)))
    (lambda (n)
      (cond ((<= n 0) 0)
            ((= n 1) 1)
            ((hashtable-contains? memo n) (hashtable-ref memo n -1))
            (else (let ((value (+ (fibonacci (- n 1)) (fibonacci (- n 2)))))
                      (hashtable-set! memo n value)
                      value))))))
  (display (fibonacci 100))
  (newline))
Saída: 354224848179261915075
```




Closure - exemplo

Python:

```
def fibonacci():  
    memo = { }  
    def calcFib(n):  
        ...  
    return calcFib
```

```
fib = fibonacci()  
print(fib.__closure__)
```

Saída: (<cell at 0x7f29a9850ad0: function object at 0x7f29a9857500>,
<cell at 0x7f29a9850b40: dict object at 0x7f29a984ad70>)



Tail recursion - exemplo

Recursividade em cauda:

```
(define (factorial n)
  (define (factorial-helper n acc)
    (if (<= n 0)
        acc
        (factorial-helper (- n 1)
                           (* n acc))))
  (factorial-helper n 1))
```

...

Saída: 120

Recursividade comum:

```
(define (factorial n)
  (if (<= n 1)
      1
      (* n (factorial (- n 1)))))

(display (factorial 5))
(newline)
```

Saída: 120



High-order function - exemplo

```
(define (merge list1 list2 func)
  (cond
    ((null? list1) list2)
    ((null? list2) list1)
    ((func (car list1) (car list2)) (cons (car list1)
                                           (merge (cdr list1) list2 func)))
    (else (cons (car list2) (merge (cdr list2) list1 func)))))
```



Hygienic macro - exemplo

Macro “primitiva”:

```
(define-macro
  (while condition . body)
  `(let loop ()
     (cond (condition
            (begin . ,body)
            (loop)))))
```

Hygienic macro:

```
(define-syntax while
  (syntax-rules ()
    ((_ condition body ...)
     (let loop ()
       (cond (condition
              (begin body ...))
              (loop))))))
```



Hygienic macro - exemplo

Macro “primitiva”:

```
(let ((i 0) (loop "0"))  
  (while (< i 10)  
    (set! loop  
      (number->string i))  
    (display loop)  
    (newline)  
    (set! i (+ i 1))))
```

Saída:

0

Error: 'oops is not a function...

Hygienic macro:

```
(let ((i 0) (loop "0"))  
  (while (< i 10)  
    (set! loop (number->string i))  
    (display loop)  
    (newline)  
    (set! i (+ i 1))))
```

Saída:

0

...

9



LINGUAGENS SEMELHANTES

As linguagens que possuem semelhanças com scheme são:

- LISP
- Common LISP
- Lua
- Ruby
- Racket



LISP e Common LISP

Possuem as seguintes semelhanças com Scheme:

- Símbolos MACRO
- Escopo de variáveis dinâmico
- Sistema padrão de tratamento de exceções
- Sistema padrão de objetos
- Sistema padrão de pacotes



Lua

Possui as seguintes semelhanças com Scheme:

- Linguagem multiparadigma: Imperativa, funcional e OO
- Trata as funções como variáveis de 1ª classe
- Suporta *closures*
- Escopo de variáveis dinâmico



Ruby

Possui as seguintes semelhanças com Scheme:

- Fortemente tipada
- Escopo de variáveis dinâmico



Racket

É uma variante de Scheme e por isso possui diversas semelhanças:

- Linguagem multiparadigma: Imperativa, funcional e OO
- Trata as funções como variáveis de 1ª classe
- Suporta *closures*
- Escopo de variáveis dinâmico
- Fortemente tipada
- Símbolos MACRO




CONSIDERAÇÕES FINAIS


- Possibilita o programador a desenvolver novas implementações relacionadas a sintaxe da linguagem e por meio disso prototipar uma gramática.
- Sintaxe extensa e minimalista provocou o surgimento de novas linguagens
- Scheme é uma linguagem que possibilita a compreensão e didática de expressões.



BIBLIOGRAFIA

- ABELSON, Harold; SUSSMAN, Jay Gerald; SUSSMAN, Julie. Struct and Interpretation of Computer Programs. 2nd. ed. [S.l.]: MIT Press, 1979. ISBN 0262510871.
- DWARAMPUDI, Venkatreddy et al. Comparative study of the pros and cons of programming languages java, scala, c++, haskell, VB .net, aspectj, perl, ruby, PHP & scheme - a team 11 COMP6411-S10 term report. CoRR, abs/1008.3431, 2010. Disponível em: <<http://arxiv.org/abs/1008.3431>>.
- DYBVIG, R. Kent. The Scheme Programming Language: ANSI Scheme. 2nd. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1996. ISBN 0134546466.

- 
- NAIM, Rana et al. Comparative studies of 10 programming languages within 10 diverse criteria. 08 2010.
 - NøRMARK, Kurt. Functional Programming in Scheme. 2013. Disponível em:
<<http://people.cs.aau.dk/~normark/prog3-03/html/notes/theme-index.html>>.
 - REVOLVY. History of the Scheme programming language. 2017. Disponível em:
<<https://www.revolvy.com/main/index.php?s=History\%20of\%20the\%20Scheme\%20programming\%20language>>.

- 
- SEBESTA, R.W. Conceitos de Linguagens de Programação - 9.ed.: Grupo A - Bookman, 2009. ISBN 9788577808625. Disponível em: <<https://books.google.com.br/books?id=vPldwBmt-9wC>>.
 - WIKI, C2. Functional Programming in Scheme. 2014. Disponível em: <<http://wiki.c2.com/?TailRecursion>>.