Revised Report on the Algorithmic Language SCHEME

Entwickler

- Guy Lewis Steele Jr.
- Gerald J. Sussman
- 1975
- MIT

Spezifikationen

- IEEE Standard
- RnRS (R5RS)

LISP

- Scheme
- Common Lisp
- Emacs Lisp
- Mac Lisp
- InterLisp
- AutoLisp
- muLisp
- XLisp
- ISLisp
- Eulisp
- SML
- TcL Logo
- NetLisp

Implementations

- Gambit-C (Unix)
- The DrScheme (Pc)
- The Bigloo Scheme
- MIT Scheme
- Guile
- The Scheme48 System
- Pixie Scheme (Mac)

Scheme

- leicht erlernbar
- wenige Regeln
- kann zu imperativen,Logik basierten, OO Programmiersprache umgewandelt werden.

Scheme - C

- Präfix
- (+ 2 3 4)
- (< low x high)
- (+ (* 2 3) (* 4 5))
- (f x y)
- (define (sq x) (* x x))

- Infix
- (2+3+4)
- ((low <x) && (x<High))
- ((2*3)+(4*5))
- f(x,y)
- Int sq(int x){return (x*x)}

Lambda Kalkül

- $f(x)= x + 2$
- $\lambda x.\ x+2$
- $x=3 \rightarrow (\lambda x.\ x+2).3$

Interpreter

- Read (einlesen der Eingabe)
- Eval (Auswerten der Eingabe)
- Print (Ausgeben einer Antwort)
- Loop (setze fort bei 1.)

Kurzblick über Scheme

- Kommentare

  durch ein Semikolon (;) eingeleitet und reichen bis Zeilenende

Variablen

- …dynamisch geschrieben
- … entweder *define* oder *let* ausgedefiniert
- (define var1 value)
- (let ((var2 value))
  …..)

Funktionen

- eine Funktion mit zwei Argumenten arg1 und arg2

  (define abc

    (lambda (arg1 arg2)

          …..))

  definiert werden

Listen

- (quote()) eine leere Liste
- (cons 1(cons 2(cons 3)))
- (list 1 2 3 4)
- (car (list(1 2 3 4) => 1
- (cdr (list(1 2 3 4) => (2 3 4)

Vektoren

- Vektoren sind Reihenfolgen wie Zeichenketten aber ihre Elemente können alles und nicht nur Buchstaben sein

- (vektor 0 1 2 3 4) oder
- #(0 1 2 3 4)

Weitere Datentypen

- integer (ganze Zahlen, beliebige Stellenzahl)
- rational (Brüche, beliebige Genauigkeit)
- real (Dezimalzahlen)
- complex (komplexe Zahlen)
- symbol
- string (Zeichenkette)
- port
- boolean
- *Wahr* und *falsch* werden durch #t und #f dargestellt

Fallunterscheidung

- **Cond**

  mit *Cond* ist es möglich mehrere Fälle abzufangen.

  (cond ((= wert 1) (display "Der Wert ist 1"))

  ((= wert 2) (display "Der Wert ist 2"))

  (else (display "Der Wert ist weder 1 noch 2)))

- IF

```
(if (eq? wert #t)
 (display "Der Wert ist wahr")
 (display "Der Wert ist falsch"))
```

Schleifen

Schleifen werden in Scheme für gewöhnlich durch eine Rekursion erreicht.

Eine Endlosschleife sieht im einfachsten Fall so aus:

```
(define (loop)
        (loop))
```

Beispiele

- This example evaluates the sum of the given list,
- and tested in EdScheme for Windows program.
- (define (list-sum lis)
- (if (null? lis) ; if empty list?
- 0 ; then sum is zero
- (+ (car lis) ; else sum is car plus the
- (list-sum (cdr lis))))) ; sum of rest of list
- #void

- =>(list-sum `(1 3 4 5 6 7 8 9 0 11))
- 54

```
Ex.2:
(define checkbook (lambda ()

                ; This check book balancing program was written to illustrate
                ; i/o in Scheme. It uses the purely functional part of Scheme.

                ; These definitions are local to checkbook
                (letrec

                  ; These strings are used as prompts

                  ((IB "Enter initial balance: ")
                   (AT "Enter transaction (- for withdrawal): ")
                   (FB "Your final balance is: ")

                   ; This function displays a prompt then returns
                   ; a value read.

                   (prompt-read (lambda (Prompt)

                                  (display Prompt)
                                  (read)))

                   ; This function recursively computes the new
                   ; balance given an initial balance init and
                   ; a new value t.  Termination occurs when the
```

```
; new value is 0.

                (newbal (lambda (Init t)
                        (if (= t 0)
                            (list FB Init)
                            (transaction (+ Init t)))))

                ; This function prompts for and reads the next
                ; transaction and passes the information to newbal

                (transaction (lambda (Init)
                             (newbal Init (prompt-read AT)))))

               ; This is the body of checkbook;  it prompts for the
               ; starting balance

               (transaction (prompt-read IB)))))
```

```
>(checkbook)
Enter initial balance: 1000
Enter transaction (- for withdrawal): -55
Enter transaction (- for withdrawal): +32
Enter transaction (- for withdrawal): -68
Enter transaction (- for withdrawal): +5
Enter transaction (- for withdrawal): 0
("Your final balance is: " 914)
```

References

- MIT/GNU Scheme
  http://swiss.csail.mit.edu/projects/scheme/index.html
- Scheme Tutorial
  http://cs.wwc.edu/~cs_dept/KU/PR/Scheme.html
- Dybvig, R. Kent.
  The Scheme Programming Language, Third Edition
  Copyright © 2003 he MIT Press. Electronically reproduced by permission.
  http://www.scheme.com/tspl3/
- Springer, G. and Friedman, D.,
  Scheme and the Art of Programming. The MIT Press, 1989.
-
- EdScheme for Windows : http://www.schemers.com/

- Revised(5) Report on the Algorithmic Language Scheme
  http://download.plt-scheme.org/doc/103p1/html/r5rs/node3.htm

- Sitaram, Dorai.
- Teach Yourself Scheme in Fixnum Days
  http://www.ccs.neu.edu/home/dorai/t-y-scheme/t-y-scheme-Z-H-1.html#node_toc_start
- Sperber, Michael. Programmieren für Alle, 2002.
  http://www.deinprogramm.de/overview/overview.html