Une introduction à Scheme

Eric Marsden

Club des Utilisateurs de Linux de Toulouse et des environs

21 January, 1999

Plan

- Pourquoi apprendre un nouveau langage de programmation?
- Histoire
- Particularités du Scheme
- Syntaxe du langage
- Applications
- Implémentations
- Autres langages comparables
- Inconvénients
- Conclusion

Histoire

- inventé en 1975 par G. Sussman, au MIT
- descendance du Lisp (McCarthy, 1950)
- normes R^5RS et IEEE
- largement utilisé dans l'enseignement de la programmation
- grand nombre d'implémentations, passant de l'interpréteur très léger au compilateur vers du code natif

Particularités du Scheme

Ce qui distingue Scheme des langages procéduraux classiques (C, Pascal, Ada, Java):

- simplicité, minimalité, élégance, flexibilité
- un petit nombre de constructions de base permet d'exprimer un grand nombre de concepts
- tout en Scheme peut-être considéré comme étant une donnée. Les programmes ont la même structure syntaxique que les listes.
- typage dynamique (à opposer au typage explicite)
- les continuations

Syntaxe du langage

```
42 → 42

(+ 1 2) → 3

(substring "Syntaxe du langage" 3 7) → "taxe"

(sqrt (- (gcd 2 5) #b11)) → 0.0+1.4142135623731i

((if (odd? 3) + *) 1 2 3) → 6
```

- syntaxe totalement parenthèsée, avec l'opérateur en préfixe
- puissante gestion des nombres
- environnement interactif
- langage extensible
- polymorphisme

Procédures

```
(lambda (x) (+ x 1)) → #<procedure>

((lambda (x) (* x 2)) 3) → 6

(define a 4)

a → 4

(define add (lambda (x y) (+ x y)))

(add a 5) → 9
```

- `lambda` permet de définir une procédure
- la procédure retournée peut être appliquée à des arguments
- `define` permet de donner une valeur à un nom
- on peut donner un nom à une procédure

factorielle

$$\begin{aligned} \mathit{factorielle}\ 0 &= 1 \\ \mathit{factorielle}\ n &= n * \mathit{factorielle}(n-1)\ \mathit{pour}\ n > 0 \end{aligned}$$

```
(define factorielle
   (lambda (n)
      (if (zero? n) 1
          (* n (factorielle (- n 1))))))

(factorielle 5) → 120
(factorielle 30) → 265252859812191058636308480000000
```

- récursivité
- pas de débordement numérique

Curryfication

```
(define add (lambda (x y) (+ x y)))

(define adder
   (lambda (n)
      (lambda (x) (add x n))))

(define succ (adder 1))

(succ 3) → 4
```

- les procédures sont des objets de première classe
- une procédure peut retourner une procédure comme résultat
- application partielle d'une fonction à ses arguments

Fermetures

```
(define (pair left right)
  (lambda (op)
    (case op
      ((gauche) left)
      ((droite) right)))))

(define (gauche cell) (cell 'gauche))
(define (droite cell) (cell 'droite))

(let ((p (pair 6 9)))
   (display (gauche p))
   (display (droite p))) → 69
```

- raccourcis pour éliminer `lambda`
- une définition de procédure engendre une *fermeture*, qui contient les liaisons variable/valeur actifs à l'instant de sa création

Objets actifs

```
(define (creer-compte solde)
   (define (retirer euros)
      (if (>= solde euros)
          (set! solde (- solde euros))
          (error "Désolé, vous n'êtes pas assez riche")))
   (define (deposer euros)
      (set! solde (+ solde euros))
      solde)
   (define (dispatch m . args)
      (cond ((eq? m 'retirer) (apply retirer args))
            ((eq? m 'deposer) (apply déposer args))
            (else (error "Désolé, je ne sais pas faire" m))))
   dispatch)

(define cb (creer-compte 1000))
(cb 'retirer 799)
(cb 'retirer 2000)
(cb 'deposer 1e6)
```

Les procédures sont des objets actifs, qui ont une durée de vie illimitée. Ils peuvent être mis dans des variables et appelées autant de fois que souhaité.

aps & cps

```
(define (factorielle-aps n)
   (define (aps n a)
      (if (zero? n) a
          (aps (- n 1) (* n a))))
   (aps n 1))

(define (factorielle-cps n)
   (define (cps n k)
      (if (zero? n) (k 1)
          (cps (- n 1) (lambda (v) (k (* v n))))))
   (cps n (lambda (x) x)))
```

- abstraction par encapsulation
- utilisation d'un accumulateur
- l'accumulateur peut être une fonction
- TMTOWTDI

Applications de Scheme

- langage d'extension du GIMP
- Emacs
- AutoCAD, FrameMaker
- DSSSL, langage de stylesheet pour documents SGML
- scripts CGI élégants
- enseignement de la programmation
- prototypage rapide

Implémentations

- DrScheme, un environnement d'apprentissage avec aide en ligne hypertexte
- SIOD est un tout petit interpréteur qui tient en 8kB
- Stalin est un compilateur Scheme qui génère du code natif parfois plus rapide que du C
- Guile, le futur langage d'extension officiel du projet GNU
- Kawa, un interpréteur écrit en Java
- Scsh est un Scheme étendu par des primitives POSIX

```
(for-each delete-file
  (filter (lambda (f) (= (file-owner f) (user-uid)))
          (glob "/tmp/*.tcl")))
```


Autres langages comparables

- Common Lisp: extensions CLOS, modularité
- famille ML de langages fonctionnels fortement typés
- Haskell, un langage fortement typé non strict
- Erlang, langage fonctionnel utilisé industriellement par Ericsson
- Python, un langage de script orienté objet bien conçu

Inconvénients

- moins performant qu'un programme équivalent écrit en langage C
- livraison d'applications nécessite un runtime
- pas d'accès direct aux librairies; il faut passer par une FFI