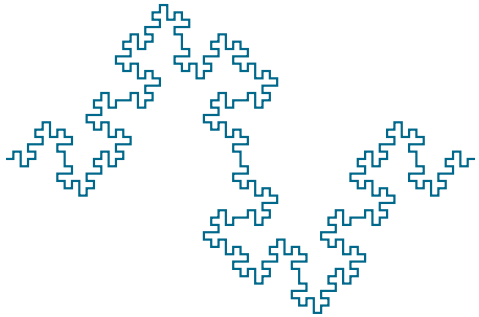
HLIN403 – Programmation Applicative

Blocs, évaluations en liaison lexicale

Christophe Dony – Annie Chateau
Université Montpellier – Faculté des Sciences



Introduction

Blocs, évaluations en liaison lexicale

Identificateurs

Blocs, évaluations en liaison lexicale

Un bloc (concept introduit par algol) est une séquence d'instructions à l'exécution de laquelle est associée un environnement propre.

En *Scheme*, toute fonction définit implicitement un bloc.

```
> (define (f x) (* x x))
> (f 2)
= 3
```

La structure de contrôle **let** définit également un bloc.

Syntaxe :

```
(let ( <liaison>* ) expression*)
  liaison ::= (identificateur s-expression)
```

Evaluation d'un "let"

Evaluation d'un "let".

Soit i une instruction de la forme :

```
(let ((v1 expr1) ... (vn exprN))
  expr)
```

$val(i) = val(expr)$ dans l'environnement du let augmenté des liaisons v1 à val(expr1) ... vn à val(exprn).

```
> (let ((x (+ 2 3)) (y (+ 3 4)))
  (* x y))
= 35
```

Bloc et environnement

L'environnement associé à un bloc de code est initialisé au moment de l'exécution du bloc, (exécution du corps d'une fonction ou du corps d'un *let*.

Pour une fonction, les paramètres formels sont liés aux arguments dans l'environnement associé au corps de la fonction.

define : est une structure de contrôle permettant d'ajouter un identificateur ou d'en modifier un dans l'environnement courant.

Modularité : structuration d'un programme en espaces de nommage.

Espace de nommage : partie d'un programme dotée d'un environnement propre.

Bloc et environnement

Un bloc est un espace de nommage. Il est possible d'utiliser sans ambigüité le même identificateur dans des blocs différents.

```
(define x 12)
(define (carre x) (* x x))
(define (cube x) (* x x x))
```

Chaînage des environnements

Environnement fils : Un environnement E2 peut être chaîné à un environnement E1, on dit que E2 étend E1, (on dit aussi que E2 est fils de E1).

E2 hérite alors de toutes les liaisons de E1 et y ajoute les siennes propres (avec possibilité de masquage).

La façon dont les environnements sont chaînés définit la portée des identificateurs.

Portée lexicale

Le chaînage est dit statique ou lexical quand un environnement d'un bloc est créé comme le fils de l'environnement du bloc englobant lexicalement (inclusion textuelle entre zones de texte).

La portée des identificateurs est alors lexicale, i.e. un identificateur est défini dans toutes les régions de programme englobée par le bloc ou se trouve l'instruction réalisant la liaison.

Scheme obéit à ce modèle (voir exemples) ainsi que la plupart des langages.

Portée dynamique

Le chaînage entre environnements est dynamique, et la portée des identificateurs également, quand l'environnement associé à une fonction devient, à chaque exécution, le fils de celui associé à la fonction appelante.

La portée dynamique a presque disparu ...

Environnement lié à la boucle toplevel

Environnement global de scheme : environnement dans lequel sont définis tous les identificateurs liés aux fonctions prédéfinies. Tout autre environnement est indirectement (fermeture transitive) le fils, de cet environnement global.

Environnement initial : environnement affecté à la boucle toplevel i.e. dans lequel sont interprétées les expressions entrées au toplevel. Cet environnement est un fils de l'environnement global. Tout bloc défini "au toplevel" créé un environnement fils de cet environnement initial.

NB : les fonctions définies dans l'éditeur sont conceptuellement définies au toplevel.

Exemples

```
> (define x 22) ;; liaison de x à 22 dans l'env.
courant
> (+ x 1) ;; une expression E
= 23 ;; ayant une valeur dans l'env courant
> (+ i 1) ;; une autre expression n'ayant pas de
valeur
erreur : reference to undefined identifier: i
```

```
> (define x 10) ;; enrichir l'environnement courant
> (let ((y 30) (z 40)) ;; créer un environnement local
  (+ x y z)) ;; et y faire un calcul
= 80
> x
= 10
> z
erreur : reference to undefined identifier: z
```

Exemples

```
(define x 1)
(define (f x) (g 2))
(define (g y) (+ x y)) ;; Une fonction avec une
variable libre
> (f 5)
= 3 (vaudrait 7 avec portée dynamique)

> (define (nb-racine a b c)
  (define delta (- (carre b) (* 4 a c))
  (if (= delta 0) ...)
)
> delta
erreur : reference to undefined identifier: delta
```

Exemples

Utilisation préférentielle d'une variable locale temporaire.

```
(define (nb-racine a b c)
  (let ((delta (- (carre b) (* 4 a c))))
    (if (= delta 0)
    ...)
)
```

Définition de fonctions, exemples

```
(define carre (lambda (x) (* x x)))
;syntaxe simplifiée
(define (sommeCarres x y)
  (+ (carre x) (carre y)))
> (sommeCarres (sin 60) (cos 60))
= 1
```

Définition de fonctions, exemples

```
(define x 1)
(define f (lambda (y) (g)))
(define g (lambda () y))
> (f 3)
*** reference to undefined identifier: y
```

Définition de fonctions, exemples

```
(define x 1)
(define f (lambda() x))
(define g (let ((x 2)) (lambda() (+ x 2))))
(define h   (lambda (uneFonction)     (let ((x 3))
(apply uneFonction ()))))
> (f)
1
> (g)
4
> (define x 5)
> (f)
5
> (g)
4
> (h g)
4
```