システム理論演習 II

(7) 状態と変数への代入

増原英彦 *(masuhara@graco.c.u-tokyo.ac.jp)*

1997 年 6 月 13 日

1 環境 (environment)

- 「ある変数がどのような値を持っているか」という情報を束縛 (binding) と言う。(例 `(define x 10)`)
- 手続を適用したような場合には、複数の束縛がまとめて発生する。一連の束縛をまとめてフレーム (frame) と呼ぶ。(例 `(define (distance x y) (sqrt (+ (square x) (square y))))`)
- 各フレームには、《外側のフレーム》がある (*enclosing environment*と言う)。

```
(define (sum n)
  (define (iter i)
    (if (< i n) ...))
  ...)
```

- フレームの列を環境 (environment) と言う。

2 評価規則

全体的な評価の方針は、適用的順序 (applicative order) になる。

2.1 Lambda 形式

手続きは、 `lambda` 形式によってのみ作られるものとする。つまり、

`(define (square x) (* x x))`

のような定義があった場合は、次のように読みかえる。

`(define square (lambda (x) (* x x)))`

以下のような `lambda` 形式があった場合:

`(lambda (x) (* x x))`

これを評価した値は、仮引数のリスト・手続本体・<u>現在の環境</u>が組になったものである。ここでは、以下のように書くことにする。

$$\{\text{param} = \texttt{x}, \text{body} = \texttt{(* x x)}, \text{env} = \textit{global-env}\} \tag{1}$$

このようなデータのことを関数クロージャ *(closure)*、あるいは単にクロージャと呼ぶ。

2.2 Define 形式

変数や手続きの定義は、

```
(define square (lambda (x) (* x x)))
```

という形をしている。このような式の評価は、まず`(lambda (x) (* x x))`を評価して (式 (1) のような値になる)、次に現在のフレームに $\texttt{x} = \{\mathrm{param} = \texttt{x}, \mathrm{body} = \texttt{(* x x)}, \mathrm{env} = \textit{global-env}\}$ という束縛を作る。

2.3 適用形式

手続きに値を適用する形式

```
(square 5)
```

を評価するには、まず、 `square` と `5` を評価する。 `square` の値は、式 (1) である。そこで、 param と引数を束縛するような、つまり、 $\texttt{x} = 5$ という束縛を持つフレームを新たに作る。このフレームの enclosing environment は、 `square` の env、つまり *global-env* である。

このようにして新しく作った環境の下で、 `square` の body、つまり`(* x x)`を評価する。

2.4 変数

変数 `x` が参照された場合は、現在のフレーム中で $\texttt{x} = v$ という束縛を探し、見つかった場合は v が `x` の値となる。見つからない場合は、 enclosing environment を順にたどって、一番最初見つかった束縛の値とする。

また、`(set! x 123)`のような代入があった場合は、変数の参照と同様にフレームを探してゆき、最初に見つかった束縛を書き換える。

3 評価の例

3.1 簡単な例:

次の一連の式:

```
(define square (lambda (x) (* x x)))
(square 5)
```

の評価は、以下のようになる。

```
genv で (define square ... の評価
    genv で 副式 (lambda ... を評価
        {param=x,body=(* x x),genv} (= v1) を得る
    define を実行: genv=[square=v1,...] になる
genv で (square 5) の評価
    genv で副式 square を評価: v1 を得る
    genv で副式 5 を評価: 5 を得る
    v1 に 5 を適用
        env1=[x=5]+genv ができる
        env1 で (* x x) を評価
            env1で副式 * を評価: 原始関数を得る
            env1で副式 x を評価: 5 を得る
            env1で副式 x を評価: 5 を得る
        * に 5 5 を適用: 25 を得る
```

3.2 銀行口座の例:

もう少し複雑な例として、銀行口座の例を挙げる。

```
(define make-withdraw
  (lambda (balance)
    (lambda (amount)
      (set! balance (- balance amount))
      balance)))
(define W1 (make-withdraw 10000))
(W1 5000)
```

これの評価は、図 1 のようになる。

```
genv で (define make-withdraw... を評価
    genv で副式 (lambda (balance... を評価
         {param=balance, body=(lambda ...), env=genv} を得る
        これを v1 とする
    define を実行: genv=[make-withdraw=v1, ...] となる
genv で (define W1 ... を評価
    genv で副式 (make-withdraw 10000) を評価
        (中略)
        v1 に 10000 を適用
            env1=[balance=10000]+genvができる
            env1 で (lambda (amount... を評価
             {param=amount, body=(set! ...) balance, env=env1} を得る
            これを v2 とする
    define を実行: genv=[make-withdraw=v1,W1=v2,...] となる
genv で (W1 5000) を評価
    (中略)
    v2 に 5000 を適用
        env2=[amount=5000]+env1 ができる
        env2 で (set! balance ...) balance の評価
            env2 で (set! balance ...) の評価
                (中略)
                set! を実行: env1 は [balance=5000]+genv になる
            env2 で  balance の評価: 50000 を得る
```

図 1: 銀行口座の評価のされ方

課題 7-1: (変数スコープ)

Let 形式は、 lambda 形式の省略記法であった。まず次の式を lambda 形式に書き換えよ。次に上の例に従って評価のされ方を説明せよ。

```
(let ((a 10))
  (+ (let ((a 20)) a)
     a))
```

∎

課題 7-2: (銀行口座)

銀行口座の例のところで、さらに以下の式が評価されたとする。この場合の評価のされ方を説明し、 W1 と W2 の残高がどこにあるか述べよ。

```
(define W2 (make-withdraw 3000))
(W2 1000)
```

∎

課題 7-3: (もっと銀行口座)

銀行口座の例のところでは、内部状態 `balance` を手続きの引数として用いていた。これと、明示的に `let` を使って持つようにした定義:

```
(define (make-withdraw initial-amount)
  (let ((balance initial-amount))
    (lambda (amount)
      (set! balance (- balance amount))
      balance)))
```

を比較せよ。具体的には、

`(define W1 (make-withdraw 10000))`

`(W1 4000)`

とした場合の評価され方を説明し、そこでの違いを示せ。

ヒント:`let` が `lambda` の略記である。

∎