関数型タートルグラフィクスの実験

Why Functional Turtle Matters

01ca0125 鈴木 藍

概要 タートルグラフィクスが有限オートマトンを理解する為にとても有用な題材であることは、1 年後期のアルゴリズムの時間に実感することが出来た。タートルは現在位置、向いている方角、ペンを上げているか、 おろしているかといった「状態」を持ち、「タートルに特定のメッセージを与える」という イベントで、この状態は遷移してゆく。オブジェクト指向という考え方も、やはり「あるオブジェクトの状態 (データの集合)」を「そのオブジェクトが受け取る事が出来るメッセージ」 で遷移させていくことである。オブジェクト指向プログラミングと関数プログラミングは、排他的な見方をされる事が多々ある一方、「状態を遷移させた後のオブジェクトは、その新しい状態を持った、同じ属性のオブジェクトが生成されることである。」という考え方もある。 関数は、あるデータを受け取った後に、ある値を返すものである事も、アルゴリズムの時間に教わった事である。ある値とは、数値はもちろん、関数を返す事も出来る。この論文は、関数そのものが、内部である値を変形させ、新しい、かつ同じような関数を返す事が出来れば、この「状態遷移」を関数プログラミングで実装する事が可能であるかを実験する。今回は、関数型言語である Lisp を用いて、状態を変え、なおかつ常に新しいオブジェクト (タートル) を生成するプログラム F-turtle (Functional Turtle) を実装し、動作を確認する実験を行った。

はじめに

本論文は、関数プログラミングの手法を使い、「あるオブジェクト (タートル) の状態を遷移させる」というプログラム F-turtle を作成、実験し、関数プログラミングとオブジェクト指向プログラミングの理解を深める事を目的としている。学校に入学してから C 言語を覚え、「構造化プログラミング」などの手法を学んだが、プログラムを作成する上で、それぞれ出て来る単語の意味を深く理解する機会は少ない。例えば関数もその一つであるが、そのエッセンスは授業の中である程度理解する事は出来た。プログラミング言語で実装することで、更に理解を深める事が出来ればと思う。

1 作成にあたって

1.1 関数プログラミングとは

関数プログラミングとは、関数だけで全体が構成されているプログラムである為にそう呼ばれている。また、代入という概念を用いないことも特徴である。

1.2 関数プログラミングの利点

関数プログラミングには代入文が含まれない。それゆえに副作用が全くない。関数の呼出しとは、結果を計算する以外の作用は持たないのである。ある変数の値を参照しても変数の値は変わる事がないので、バグの大きな源の一つを断つ事が出来る。また、式の値を変更する副作用が無いので、いつの時点で式を評価してもよい。(実行順序を気にしなくてもよい)

2 プログラムの開発環境

プログラムの作成は自宅で行った。

- OS : VineLinux 2.1.5 (GNU/Linux)
- 解釈系 : cmu Common Lisp

3 プログラムの主な関数

タートルを生成する関数は、常にそのタートルの状態を引数とし、各モジュール間で通信される。

;; タートルを生成する関数

```
(defun turtle (x y)
  (function (lambda (message)
      (eval-message message x y 0 penup)
          )))

;; 状態を変えた時に新しいタートルを生成する関数
(defun next-turtle (x y len pen)
  (function (lambda (message)
      (eval-message message x y len pen)
          )))
```

タートルが受け取る事が出来るメッセージは、eval-message という評価器で評価する。

```
;; メッセージセレクタ
(defun eval-message (message x y len pen)
  (cond
   ((forwardp message)
     (go-ahead x y len pen
               (get-param message)))
   ((backp message)
     (go-back  x y len pen
               (get-param message)))
   ((turn-rightp message)
     (turn-right x y len pen
               (get-param message)))
   ((turn-leftp  message)
     (turn-left  x y len pen
               (get-param message)))
   ((pendownp message)
     (do-pendown x y len))
   ((penupp   message)
     (do-penup   x y len))
   (T (print 'unknown-message))
   )
)
```

評価器でマッチした式はそれぞれの状態の計算を行い、新しい関数 (タートル) を返す。

```
;; forward というメッセージを受け取った時
;; メッセージのパラメータで現在位置を計算し
;; その状態をもつ新しいタートルを生成する。
(defun go-ahead (x y len pen param)
  (print (+ x (* param (sin len))))
  (print (+ y (* param (cos len))))
  (next-turtle (+ x (* param (sin len)))
               (+ y (* param (cos len)))
                                len pen))
```

4 実行結果

まず、定数として penup pendown の値を設定する

```
(setq penup '0)
(setq pendown '1)
```

F-turtle のトップレベルとして、以下の関数を作成した。

```
;; 入力を受け付ける
(defun F-turtle ()
  (loop
   (print '->turtle)
   (eval (read))))
```

意味付の為の二つのマクロを作成した。

```
;; 新規にタートルを生成する
;; 引数 name に、関数を束縛する
(defmacro new (name status)
  (list 'setq name status))

;; タートルにメッセージを送る
;; 束縛した変数を funcall して新しい関数を得た
;; あとに更に 同じ変数名 name に関数を束縛する
(defmacro send (name message)
  (list 'setq name
        (list 'funcall name message)))
```

[Lisp を起動する]

```
* (F-turtle)

->TURTLE

[ 新規タートルを 0 0 の座標に生成する ]
->TURTLE (new hoge (turtle 0 0))
Warning:  Declaring HOGE special.

->TURTLE
```

```
[ ペンをおろす ]
->TURTLE (send hoge '(pendown))

PENDOWN
->TURTLE

[ タートルを前進させる ]
->TURTLE (send hoge '(forward 10))

0.0
10.0
->TURTLE

[ タートルを右に 90 度向かせる ]
->TURTLE (send hoge '(turn-right 90))

90
->TURTLE

[ タートルに前進させる ]
->TURTLE (send hoge '(forward 10))

8.939966
5.5192637
->TURTLE
```

結論　私自身、オブジェクト指向プログラミングと関数プログラミングはかけ離れた物だと考えていたが、今回 この「状態を変化させる」というモデルを実装したことで、「どっちもいっしょ。」という結論に改めて達した。プログラムに関しての不満は多いが、一番の不満は可視化が出来なかったことである。この点に関しては、可視化を初めから容易に出来る関数を用意している解釈系があるとのことなのでそれを使って実装してみたい。また、出力されるタートルの座標にかなりの誤差が出るようである。これに関しても改善してゆきたい。

参考文献

[1] Why Functional Programming Matters
John Hughes

[2] 極論とは
Reishi Morioka
(2002 年 4 月 1 日 初版 ピアソン)

[3] 壁を破れ
Aochi Hara
(2002 年 4 月 1 日 初版 18)