# 関数プログラミング用語集

2006-08-26
LL Ring 2006
LL で関数プログラミング
nobsun at sampou dot org

# 目次

- 関数プログラミング
- 再帰
  - 例題(1)
  - 例題(2)
- 高階関数
- 宣言的プログラム
- 型付け
- 型推論
- 遅延評価
- 参照透明性

# 関数プログラミング(1)

- 関数プログラミングとは
  - Not　関数型言語を使うこと
  - Is　関数を用いて世界を
    スタティック？に記述すること
- cf オブジェクト指向プログラミングとは
  - Not　オブジェクト指向言語を使うこと
  - Is　オブジェクトを生成し世界を
    ダイナミックに模倣すること

- 俺様定義につき注意

# 関数プログラミング(2)

- 関数を中心に・・・
  - 関数が計算の対象
    - 関数が第一級(First-Class)
    - 高階関数が中心になるプログラミング
  - 関数抽象
    - λ(lambda)、無名関数
  - 関数適用
    - β簡約、置き換えモデル

# 再帰(recursion, recursive)

- 再帰的に定義された関数

```
fib(n) = 0                          ,if n = 0
       = 1                          ,if n = 1
       = fib(n-2) + fib(n-1), otherwise
```

- 再帰的定義は理解しにくいか？

- 再帰的に定義されたデータ型

```
List a = Nil
       | Cons a (List a)
```

# 再帰:例題(1)

- 問題
  平方根を計算する
- 構造
  予測値（guess）から x の平方根を求める
  - 予測値 guess の二乗が x に十分近ければ、
    - その予測値を解とする
  - 近くなければ、
    - 予測値を改良し、新たな予測値から x の平方根を求める

# 再帰関数(1)

```
sqrt'(guess,x) = if (enough(guess,x)) then
                    guess
                 else
                    sqrt'(improve(guess,x),x)

improve(guess,x) = average(guess,x/guess)

average(x, y) = (x + y) / 2

enough (guess,x) = abs (guess^2 – x) < 0.001
```

# 再帰:例題(2)

- 両替問題
  50, 25, 10, 5, 1 セント硬貨を使って両替を行う場合の数
- 問題の構造
  「n 種の硬貨で金額 a の両替を行う場合の数」
  ＝「最初の硬貨を使わず a を両替する数」
  ＋「最初の硬貨少くとも 1 枚使って a を両替する数」
- 縮退した場合
  1. 金額が丁度 0 なら両替方法は 1 と数える
  2. 金額が 0 未満なら両替方法は 0 と数える
  3. 両替する硬貨の種類が 0 なら両替方法は 0 と数える

# 再帰関数(2)

```
countChange :: Amount -> Count
countChange a = cc 5 a

cc :: (Coins,Amount) -> Count
cc (c,a) | a == 0    = 1
         | a <  0    = 0
         | c == 0    = 0
         | otherwise = cc (c - 1,a) + cc (c,a – denom(c))

denom :: Coins -> Amount
denom(c) = case c of
             5 -> 50
             4 -> 25
             3 -> 10
             2 ->  5
             1 ->  1
```

# 高階関数

- 入力に関数を含む関数
  - 例：Newton 法
  - 再帰的に定義されたデータ上の高階関数
    - イテレータ？
    - 畳み込み関数
- 出力に関数を含む関数
  - 関数から関数を生成する
- 一般化の道具
  - デザインパターン？

# 高階関数の例

- 例題(1)の拡張、一般化
  - 3乗根とか4乗根も計算したい
  - 基本構造は同じ→再利用パターン？
  - improveを変更する
    - improveをパラメータにする
    - improveを生成する材料になる関数をパラメータにする
  - Newton法

# 宣言的プログラム

- 「命令的(imperative)プログラム」の対照語
- 「宣言的」記述：
  どのように計算するか(How)ではなく
  何を計算するか(What)を記述
- 「順次実行」がない？
- 順序を気にしなくてよい
  - 評価順
  - 定義順

# 型

- 型　＝　値の集合
- 動的型付け：実行時に型付けする
  - 型宣言しなくてよい
  - コンパイル時に型チェックしない
- 静的型付け：コンパイル時に型付けする
  - 型宣言しなくてよい
  - コンパイル時に型チェックする

# 型推論

- 多相型
  - パラメータ多相
    - length
  - アドホック多相
    - (==)
- 関数の型： a -> b
  「a 型の値から b 型の値への関数」という型
- 型推論
  - 型が確定している式を根拠に、他の式の型を推論し型付けする

# 遅延評価

- 正格関数：　f(⊥) ＝ ⊥
- 非正格関数：f(⊥) ≠ ⊥
- 必要になるまで、式を評価しない
- call-by-need( 必要呼び )
- データコンストラクタも遅延評価

- 例：　自然数の集合 ( 無限リスト )
- 例：same-fringe 問題 (call/cc ？ )

# 参照透明性

- 字面と意味との関係
  - 「字面が同じ式は同じ値を示している」
  - 「同じ値の式は交換可能」
- 副作用（変数と値の対応関係の変更）なし
- 引用（Scheme の quote）なし