関数・論理型プログラミング実験
ML演習第7回

松田 一孝
TA: 武田広太郎 寺尾拓

今日の話

- 5/01：簡単な評価器
  - 字句解析・構文解析．簡単な評価器
- 5/13：関数型言語の評価器
  - （高階）関数定義・呼出機構の作成
- 5/20：型システム
  - ML風の型推論の実装
- 5/27：その他拡張
  - 評価規則等

今日の話

- 様々な評価戦略
  (evaluation strategy)
  - 値呼び (call by value)
  - 名前呼び (call by name)
  - 必要呼び (call by need)

評価可能な部分式

- 一般には複数個

```
fst (1+2, 3+5)
```

どの式を先に評価？

評価戦略

- どの式を最初に評価するか

評価戦略の影響

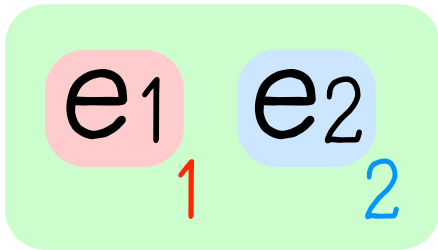
- 基本的には評価順序は評価結果に影響しない
  - 例外：停止しない計算
    - `let rec loop () = loop ()`
      `in fst (3, loop ())`
  - 例外：副作用
    - `(fun (x,y) -> print_int 1; x)`
      `(print_int 2, print_int 3)`

主な3つの評価戦略

- 値呼び (call by value)
- 名前呼び (call by name)
- 必要呼び (call by need)

  - 他の戦略
    * normal orderやapplicative orderなど

値呼び



- 関数の適用前に引数を評価
  - (fun x -> x*x) (2+3)
    → (fun x -> x*x) 5
    → 5*5
    → 25
  - なお，let x = e_1 in e_2の評価は
    (fun x -> e_2) e_1と同じように評価
    * 今回紹介する他の戦略でも同じ
- 今迄実装していたのはこれ

値呼びの利点

- 関数呼出にオーバヘッドがない
  - 特に加算や減算などの組み込み演算
    - 整数値や浮動小数点値を
      そのまま使える

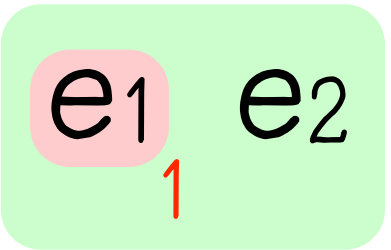
- 評価順がわかりやすい
  - 副作用を扱いやすい
    - 参照，I/O，…

値呼びの欠点

- 他の評価戦略なら止まるのに
  値呼びだと止まらない場合がある
  - fst (5, loop ())
- ifや||, &&を関数で表現できない
- 展開・折り畳みによる最適化
  と相性が悪い
  - fst (x, y) = x なのに
    式中のfst (e_1, e_2) を e_1 にできない

名前呼び

- 関数適用を引数より先に評価
  - (fun x -> x*x) (2+3)
    → (2+3)*(2+3)
    → 5*(2+3)
    → 5*5
    → 25



  - 式は必要になるまで評価しない

名前呼びの利点

- 他の評価戦略で計算が止まるなら名前呼びでも止まる
- ifや&&, ||等を関数として実現できる
  - 「マクロ」的な機能を実現可
- 不必要な計算を避けられる
  - 例：hd (sort xs)がO(xsの長さ)で！
    - * sort の実装にはよるが
- 展開・折り畳みによる最適化と相性が良い

名前呼びの欠点

- 関数呼出のオーバヘッドが大きい
  - 式（+環境）を関数に渡す
    - intやboolなどの基本型の場合もレジスタで渡せなくなる
- 同じ式を何度も評価
  - 例：(fun x -> x*x) (2+3)

- 式の評価回数やタイミングが制御困難
  - 副作用との相性がわるい
    - I/O，参照等が必ずしも副作用で実装されなくてもよいことに注意（cf. Haskell）

必要呼び

- 式は必要になるまで評価しない，が式の評価結果を共有
  - (fun x -> x*x) (2+3)
    → x*x　　{x=2+3}
    → x*x　　{x=5}
    → 5*5
    → 25
    * 2+3の評価はxを通して共有されるので一度のみ
  - HaskellやConcurrent Cleanなどで採用

必要呼びの利点・欠点

- 必要呼びの利点 =
  「名前呼び」の利点
  + 評価回数が高々「値呼び」程度

- 必要呼びの欠点 =
  「名前呼び」の欠点
  − 同じ式を何度も評価
  + オーバヘッドがさらに大きく

必要呼びの注意

- 同じ式を二度評価しないわけではない
  - 例：素朴なfibは指数時間
    - let rec fib n =
      if n < 2 then 1
      else fib (n-1) + fib (n-2)

- 二度評価しないのは
  変数を通して共有される式のみ
  - つまり，let式や引数のとき
    - OK: let x = 2+3 in let y = x in x*y
    - NG: (2+3)*(2+3)

注意

- (e_1, e_2)の評価順序はこれら三つの評価戦略と関係ない
- if e_1 then e_2 else e_3の評価順序は三つの評価順序で同じ
  - e_1を評価しないと分岐できない
  - match式自体も同じ，が match e with…のeをどこまで評価するかが異なる

まとめ

- 三つの評価戦略を紹介した
  - 値呼び（call-by-value）
  - 名前呼び（call-by-name）
  - 必要呼び（call-by-need）

第7回レポート課題
締切 6/10 13:00 (JST)

問1

- 前回のインタプリタを改良し，名前呼びで評価を行うようにせよ
  - 組やリスト，match式は（問1では）対応しないでよい

ヒント

- サンク（thunk）：式と環境の組
  - 適用e_1 e_2の評価
    - 現在の環境でe_1を評価，
      クロージャ〈fun x -> e, env〉を得る
    - envに，xとサンク〈e_2, 現在の環境〉の対応を追加し，eを評価
  - 変数xの評価
    - xが現在の環境で〈e, env〉を指しているならenvの下でeを評価する
      - 環境が，変数からサンクへの写像になっていることに注意

ヒント

- let x = e_1 in e_2の評価は
  (fun x -> e_2) e_1と同じように

- これまで実装した eval は
  そのまま残すべし
  - たとえば，今回実装するものは
    eval_byname とする
  - 結果を比較することでデバッグの役に

問2

- 問1のインタプリタに
  - let rec f = eという構文を追加し，関数以外を再帰的に定義できるようにせよ
  - 組とリストを追加せよ
    - let rec loop () = loop () in fst (3, loop ()) や
    - … snd (loop (), 3) は3を返すように
    - let rec ones = 1 :: ones で無限リストを作れるように

ヒント

- 組やリストの評価結果を表す値を
  - (thunk1, thunk2)や
  - thunk1 :: thunk2とする
- match式はthunkを評価しながらパターンマッチするように

問3

- 前問のインタプリタ上で以下の無限リストを作成せよ
  - ones = [1; 1; 1; 1; …]
  - nats = [1; 2; 3; 4; …]
  - fibs = [1; 1; 2; 3; 5; 8; …]

* オーバフローは気にしない

問4

- 前問のインタプリタを改良し
  必要呼びで評価を行うようにせよ
  - 組やリストも扱えるようにせよ

注意

- (fun x -> (fun y -> x*y) x) (2+3)
  でも2+3が一回しか評価されないよう気をつけよ

ヒント

- 環境を，変数から「サンクまたは値」の参照への写像に
  - 変数xの評価
    - 現在の環境をルックアップし，「サンクまたは値」の参照rを得る
    - rの指すものが…
      - 値ならそれを返す
      - サンク 〈e, env〉 なら
        - envの下でeを評価しvを得る
        - r := v
        - vを返す

ヒント

- 組やリストの評価結果を表す値を
  - (サンクか値への参照,
    サンクか値への参照)や
  - サンクか値への参照 ::
    サンクか値への参照 とする

発展（Parallel Or）

- 前回のインタプリタを拡張し
  以下の演算子|||を扱えるようにせよ
  - e_1 ||| e_2の評価結果は…
    - true
      - e_1の評価結果がtrue か
        e_2の評価結果がtrue のとき
    - false
      - e_1の評価結果がfalse で
        e_2の評価結果もfalse のとき

注意

- true ||| loop () も
  loop () ||| true も
  true に評価される

ヒント

- e_1 ||| e_2の評価の難しさ
  - e_1を先に評価すると，
    loop () ||| trueが停止しない
  - e_2を先に評価すると，
    true ||| loop ()が停止しない
- 案
  - e_1とe_2を並行に評価し，片方がtrueになったら，trueを返す
    * small-step評価などを利用？
    * スレッドを使う？

発展2

- 名前呼びにおいて，sortの種類によっては，hd (sort xs)がO(xsの長さ)時間で実行可能である
  - なぜか，説明せよ
  - いくつかのsortについてどれかOKでどれがダメか考察せよ

    * O(n)とO(n log n)の違いを実行して速度比較することで確かめようとしないこと

来週からHaskell演習

- aptitude install haskell-platform
- apt-get install haskell-platform
  - Haskell PlatformはHaskell処理系ghcに構文解析生成器やドキュメンテーション生成器などをまとめたもの
  - ghcのコンパイルにはghcが必要
    - ソースからのビルドは大変
- MacやWindowsなら
  http://www.haskell.org/platform/
  からインストーラをダウンロード