



ML 演習 第 6 回

おおいわ
April 10, 2001



今回の内容

- 補足
 - デバッグ環境
 - 標準出力
- ML風言語のインタプリタ (その2)
 - 式の評価順序と評価結果

デバッグ環境

- ocaml インタプリタ
 - `#trace func;;` で `func` の呼び出しを監視
 - `#untrace func;;` で監視を解除
- ocamldebug
 - コンパイルされた実行ファイルに対してソースレベルデバッグを実現
 - emacs との協調が可能
 - 後退実行のサポート

標準出力関係

- print_int, print_string, print_float, print_char: それぞれの型の値を出力
- print_newline: 改行
- print_endline: 文字列を出力し改行
- Printf.printf: C 言語の printf 相当
 - 引数はきちんと型チェックされる
 - printf 用のちょっと特殊な型規則が存在
- その他はマニュアルを参照

式の評価順序 (1)

- 関数型言語の性質:

- 基本的には評価順は評価結果に影響しない
 - 例外1: 副作用の存在
 - 例外2: 止まらないプログラム

```
let rec iter x = iter x in fst (5, iter x)
```

式の評価順序 (2)

- 代表的な3つの評価戦略
 - Call By Value (値渡し)
 - Call By Name (名前渡し)
 - Call By Need (遅延評価)

Call by Value (1)

■ 関数などの引数を適用の前に評価

- fib (1 + 1) → fib 2 →
 - if 2 < 2 then 1 else fib (2 - 1) + fib (2 - 2)
 - if false then 1 else fib (2 - 1) + fib (2 - 2)
 - fib (2 - 1) + fib (2 - 2)
 - fib 1 + fib (2 - 2)
 - (if 1 < 2 then 1 else ...) + fib (2 - 2)
 - * 1 + fib (2 - 2) → 1 + fib 0
 - * 1 + 1 → 2

Call By Value (2)

■ 利点

- 高速な実装が可能
 - ほとんどの値は計算中に即値として現れる
- 評価順がわかりやすい
 - 副作用が扱いやすい

Call By Value (3)

■ 欠点

- 結果が定まっている式の評価が止まらないことがある

let rec loop x = loop x in fst (5, loop x) → 発散

- 対策: いくつかの special form を用意
 - if は条件節と必要な方の節しか評価しない
 - if は関数ではない
 - ||, && などと同様

Call By Value (4)

■ 評価順序の制御

■ λ 抽象は評価順の制御に使える

■ `let if_f b x y = if b then x else y`

■ `if_f true 5 (loop x)` → 発散

■ `let if_d b x y = if b then x () else y ()`

■ `if_d true (fun () -> 5) (fun () -> loop x)`

→* `if true then (fun () -> 5) () else (fun () -> loop x) ()`

→ `(fun () -> 5) ()` → 5

Call By Name (1)

- 外側の関数適用を先に評価
 - $\text{let } f\ x = x * x \text{ in } f\ (5 + 3)$
 - $f\ (5 + 3)$
 - $(5 + 3) * (5 + 3)$
 - $8 * (5 + 3)$
 - $8 * 8$
 - 64

Call By Name (2)

■ 利点

■ 計算能力が高い

- Call By Value で評価可能なものは評価可能
- さらに `let rec loop x = loop x in fst (5, iter x)`
→ `fst (5, iter x)` → `5 + 3` → `8`
- `if` も通常の組み込み関数として定義可能
 - `let (if) b x y = { b を評価し x か y を返す }`
 - `(if) true (5 + 3) (loop x) →* 5 + 3 → 8`

Call by Name (3)

■ 欠点

- 実装が遅くなる
 - 常に「式」の形で評価を進める必要がある
- 同じ式を何回も評価する
- 式の評価回数やタイミングが制御困難
 - 副作用がある言語では使いづらい

Call By Need (1)

- 外側の関数適用を先に評価
- 同じ式は1回だけ評価、結果を使い回す
 - `let f x = x * x in f (5 + 3)`
 - `f (5 + 3)`
 - `(5 + 3) * (5 + 3)`
 - `8 * 8`
 - `64`

Call By Need (2)

■ 利点

- 計算能力が高い
- 1つの式の評価は1回で済む

■ 欠点

- 実装が遅くなる
- 式の評価タイミングは依然制御困難

遅延評価の実装 (1)

- module Delayed: Call-by-Name の実現
 - delay : 遅延評価される式を表す
オブジェクトを生成
 - 使い方: delay (fun () -> 式)
 - force : delay された式の実際の値を得る

遅延評価 (2)

■ 例 (1)

```
# let eager_if b x y = if b then x else y;;  
val lazy_if : bool -> 'a -> 'a -> 'a = <fun>  
# let rec eager_fact x = eager_if (x = 0) (1)  
                                (x * eager_fact (x - 1));;  
val eager_fact : int -> int = <fun>  
# eager_fact 10;;  
(止まらない...)
```

遅延評価 (3)

■ 例 (2): if の2つの選択肢を遅延評価

```
# open Delayed;;  
# let lazy_if b x y = if b then (force x) else (force y)  
val lazy_if : bool -> 'a Delayed.delayed ->  
    'a Delayed.delayed -> 'a = <fun>  
# let lazy_fact x = lazy_if (x = 0) (delay (fun () -> 1))  
    (delay (fun () -> x * lazy_fact (x - 1))));;  
val lazy_fact : int -> int = <fun>  
# lazy_fact 10;;  
- : int = 3628800
```

課題1

- module Delayed を Call-by-Need semantics で再実装せよ。
 - 1回 force された値を記憶しておき、同じ値が2回以上 force された場合でも評価が1回しか起こらないようにする。
 - 現状:
 - `let p = delay (fun () -> print_endline "eval!"; 5 + 3)`
`in (force p) * (force p)` とやると `eval!` が2回表示される。

課題2

- 遅延 (無限) リストを表現する `module Sequence` を実装せよ。
 - tail 部分を遅延させます。
 - `head: α seq \rightarrow α` 先頭要素を取得
 - `tail: α seq \rightarrow α seq` 先頭を除いた Seq.
 - `nil: α seq` 空 Sequence
 - `cons: $\alpha \rightarrow \alpha$ seq delayed \rightarrow α seq`
 - `take: α seq \rightarrow int \rightarrow α list`
 - `take s n` : `s` の先頭最大 `n` 要素を取り出す

課題3 (optional)

- 前回のインタプリタを Call-by-need で再実装せよ。
 - データ構造は任意です: 必ずしもプリントの `type lazymlvalue` に従わなくていいです。
 - データ型を変えた際は `make clean; make` で `Reader` などを作り直してください。

課題3 (ヒント)

■ 戦略

- force → 式の形を見ながら今すぐに必要なところだけを評価
 - Primitive: 引数がすべて必要
 - Cons, Pair: 引数は delay されていてよい
 - Apply: 左辺の値 (λ 式) が必要
 - Match: パターンマッチに必要な部分だけは force されていないといけない



提出方法

- 〆切: 2001年5月28日 (月) 24:00
- 提出先: ml-report@yl.is.s.u-tokyo.ac.jp
- 題名: Report 6 (学生証番号)