ML 演習 第 1 回

飯塚大輔, 後藤哲志, 前田 俊行

June 5, 2007

今日から ML 演習

- 担当者
  - 飯塚 大輔, 後藤 哲志, 前田 俊行 (米澤研)
    - 質問アドレス: ml-query-2007@yl.is.s.u-tokyo.ac.jp
    - 課題提出アドレス: ml-report-2007@yl.is.s.u-tokyo.ac.jp
- 講義はおおむね飯塚、後藤が担当

演習の内容

- 第1回～第4回
  - ML 言語を学ぼう
    - 本演習では ML の一派である Objective Caml を使用
- 第5回～第7回
  - ML インタプリタを作ってみよう
- 第8回
  - 最終課題: リバーシ思考ルーチンの実装（予定）

連絡

- 来週は102教室で行います

演習資料について

- ML演習ホームページ
  - http://www.yl.is.s.u-tokyo.ac.jp/~sgotou/lecture/caml-enshu/
  - 講義資料(PDF, OpenOffice.org Impress)
  - 演習で使用するソースファイル
  - 参考資料へのリンク

参考資料

- OCaml の本家サイト http://caml.inria.fr/
  - マニュアル・紹介など
    - 第1章のチュートリアルは簡潔でよい
  - 処理系のダウンロード (Windows 版など)
- Developing Applications With Objective Caml
  - Online pre-release
  - http://caml.inria.fr/pub/docs/oreilly-book/

日本語の資料

- プログラミングの基礎（サイエンス社）
  - 浅井 健一 著
  - ISBN 978-4781911609
- 入門OCaml プログラミング基礎と実践理解
  - OCaml-Nagoya 著
  - ISBN 978-4839923112

今日の内容

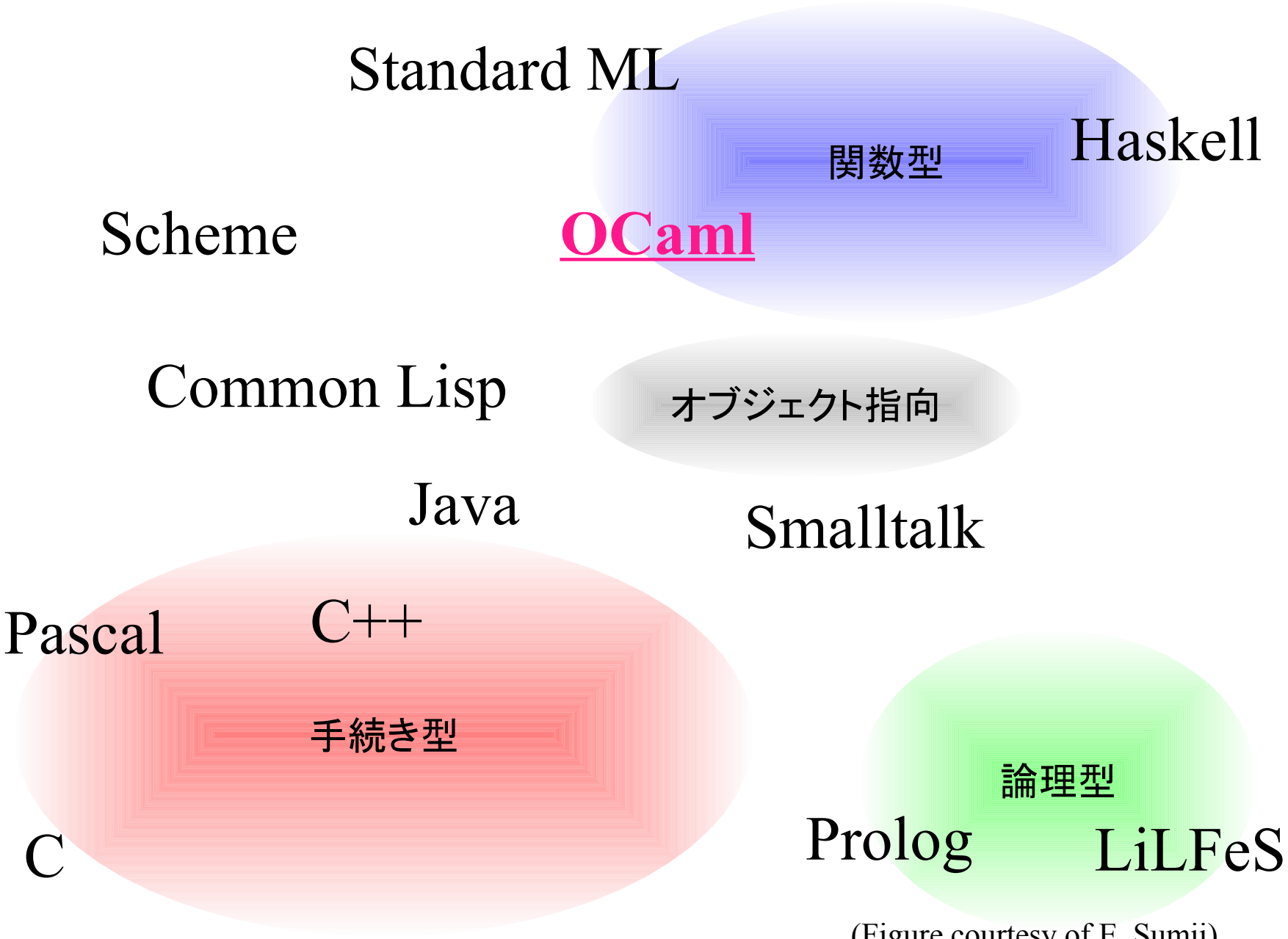
- Objective Caml とは?
- インタプリタの使い方
- 基本的な値と文法
- 評価について

今日の内容

- Objective Caml とは?
- インタプリタの使い方
- 基本的な値と文法
- 評価について

Objective Caml とは?

- 関数型言語 ML の1流派
  - 強力な型システム
  - 柔軟なデータ型定義とパターンマッチ
  - 強力なモジュールシステム
  - オブジェクト指向プログラミングのサポート



(Figure courtesy of E. Sumii)

MLの型システムの特徴

- 強い静的な型付け (Strong Static Typing)
  - 「強い」 = 型整合を強制する
    - 弱い型付け (e.g. C, C++)
  - 「静的な」 = コンパイル時に型をチェックする
    - 動的な型付け (e.g. Scheme, Perl)
- 型推論
  - プログラマは変数の型を書かなくてよい
    - C, C++などは明示的な型の指定が必要
- 型多相 (Parametric Polymorphism)
  - 第2回で解説

今日の内容

- Objective Caml とは?
- インタプリタの使い方
- 基本的な値と文法
- 評価について

インタプリタを使うには

- 地下環境(csc)にはインストール済み
- 自分のマシンを使う場合は各自でインストール
  - Unix
    - パッケージを利用する(Fedora Core 6ならyumなど)
    - ソースからコンパイルする
  - Windows
    - 配布されているバイナリを利用する
    - cygwin ならインストーラを使えば入れられる
    - 自前でソースからコンパイル (VC, mingw32)

インタプリタの使い方

```
$ ocaml
        Objective Caml version 3.09.3

# 1 + 2;;
- : int = 3
```

1+2 という式を評価すると
結果の型は int (整数型) で
値は 3 になる

ファイルに書いたプログラムの利用

```
# #use "test.ml";;
- : int = 3
- : float = 3.
# ^D
```

test.ml の中身:

```
1 + 2;;
1.0 +. 2.0;;
```

Ctrl + D(EOF) で
インタプリタを抜ける

型エラーの例

float 型の式が
int 型の使われるべき場所に現れている

```
# 1 + 2.0;;
This expression has type float but
  is here used with type int
```

今日の内容

- Objective Caml とは?
- インタプリタの使い方
- 基本的な値と文法
  - コメントの書き方
  - 変数定義
  - 組み込み型
    - int, float, bool, string, tuple
  - 関数型
  - その他の構文 (条件分岐, 再帰関数の定義, 相互再帰)
- 評価について

コメントの書き方

- (* と *) の間はコメントとして無視される

```
# (* comment *) 2 + 3;;
- : int = 5
```

- なんと，入れ子にもできる（Cのコメントではダメ）

```
# 1 (* + 2 + (* 3 + *) 4 *) + 2;;
- : int = 3
```

変数を定義して使う方法

- let 式: トップレベルの定義

```
# let a = 3;; (* 変数の定義 *)
val a : int = 3
# let f x = x + 1;; (* 関数の定義 *)
val f : int -> int = <fun>
# a + a;;
- : int = 6
# f a;; (* 関数適用 *)
- : int = 4
```

変数を定義して使う方法

- let ... in 式: ローカルな宣言

```
# let x = 2;;
x : int = 2
# let x = 3 in x + x ;;
- : int = 6
# x;;
- : int = 2
# let f x = x + x in f 5;;
- : int = 10
```

x = 3 はこの範囲のみで有効

組み込み型 (1)

- 整数 (int)

```
# (3 + 5) * 8 / -4;;
- : int = -16
# 5 / 4;;
- : int = 1
# 5 mod 4;;
- : int = 1
# 3 < 2;;
- : bool = false
```

組み込み型 (2)

- 実数 (float)

```
# (3.0 +. 5.0) *. 8.0 /. -3.0;;
- : float = -21.3333333333333321
# 1.41421356 ** 2.0;;
- : float = 1.99999999328787381
# 3.0 < 2.0;;
- : bool = false
```

組み込み型 (3)

- 真偽値 (bool)

```
# 2 < 3 && 2.0 >= 3.0;;
- : bool = false
# 2 < 3 || 2.0 = 3.0;;
- : bool = true
# not (3 < 2);;
- : bool = true
```

組み込み型 (4)

- 文字列 (string)

```
# "Str" ^ "ing";;
- : string = "String"
# print_string "Hello\nWorld\n";;
Hello
World
- : unit = ()
```

組み込み型 (5)

- 組 (tuple)

```
# (3 + 5, 5.0 -. 1.0);;
- : int * float = (8, 4.)
# fst (3, 2);;
- : int = 3
# snd (3, 2);;
- : int = 2
# (3, true, "A");;
- : int * bool * string = (3,true,"A")
```

関数型 (1)

- 関数 (function)

```
# let f x = x + 2;;
val f : int -> int = <fun>
# f 2;;
- : int = 4
# fun x -> x + 2;; (* 匿名関数 *)
- : int -> int = <fun>
# (fun x -> x + 2) 2;;
- : int = 4
```

関数型 (2)

- 多引数関数

```
# let f x y = x * (x + y);;
val f : int -> int -> int = <fun>
# f 2 4;;
- : int = 12
# fun x y -> x * (x + y)
(* 匿名の多引数関数 *);;
- : int -> int -> int = <fun>
# (fun x y -> x * (x + y)) 2 4;;
- : int = 12
```

関数型 (3)

- 多引数関数の型
  - カリー化 (Curried) 表現

```
# let f x y = x + y;;
val f : int -> int -> int = <fun>
# f 2;;
- : int -> int = <fun>
# (f 2) 4;;
- : int = 6
```

基本的な構文 (1)

- 局所定義 (let ... in ...)
- 条件分岐: if

```
# let f x = if x < 2
            then "smaller than 2"
            else "not smaller than 2";;
val f : int -> string = <fun>
# f 1;;
- : string = "smaller than 2"
```

基本的な構文 (2)

- 再帰関数

```
# let rec pow x n =
    if n = 0 then 1
    else x * pow x (n - 1);;
val pow : int -> int -> int = <fun>
# pow 3 10;;
- : int = 59049
```

基本的な構文 (3)

- 末尾再帰で書くとこうなる

```
# let rec powsub x v n =
    if n = 0 then v
    else powsub x (v * x) (n - 1);;
val powsub : int -> int -> int -> int =
 <fun>
# let pow x n = powsub x 1 n;;
val pow : int -> int -> int = <fun>
# pow 3 10;;
- : int = 59049
```

基本的な構文 (4)

- powの中でpowsubを定義できる

```
# let pow x n =
    let rec powsub v n =
      if n = 0 then v
      else powsub (v * x) (n - 1)
    in powsub 1 n;;
val pow : int -> int -> int = <fun>
# pow 3 10;;
- : int = 59049
```

基本的な構文 (5)

- 相互再帰関数の同時定義

```
# let rec even x = if x = 0 then true
                   else odd (x - 1)
  and odd x = if x = 0 then false
              else even (x - 1);;
val even : int -> bool = <fun>
val odd : int -> bool = <fun>
# odd 5423;;
- : bool = true
```

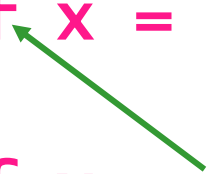
基本的な構文 (5)

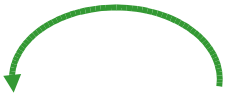
- let と let rec の違い…

```
# let f x = x + 3;;

# let f x = f (x - 1);;
# f 2;;
- : int = 4

# let rec f x = f (x - 1);;
# f 2;; (* 止まらない *)
```





今日の内容

- Objective Caml とは?
- インタプリタの使い方
- 基本的な値と文法
- 評価について

レポートについて

- 毎回 5 問程度
- 出題後 2 週間以内に提出
- 締め切り**厳守**
- 質問は以下のアドレスまで:
  ml-query-2007@yl.is.s.u-tokyo.ac.jp

単位取得条件

- 毎回の必須課題をすべて解くこと
  - optional課題は解かなくても良いが，解いたら加点

レポート提出上の注意 (1)

- 提出方法: 電子メール
  - 宛先: ml-report-2007@yl.is.s.u-tokyo.ac.jp
    - 受領通知が届くと思うので確認のこと
- Subject を
  Report <レポート番号> <学生証番号>
  とすること
  - 今回の場合 Report 1 710xx

レポート提出上の注意 (2)

- レポートに含めるべきもの
  - 氏名, 学生証番号
  - ソース
    - 添付ファイルにせず, メール本文にコピペすること
    - コメントを適宜補い, 各関数の動作を説明すること
  - 動作例
    - プログラムが正しく動作することを示すのにふさわしい例を考えること
  - 考察
    - 考察不要と指定されている場合を除き, 必ず入れる

第1回 課題

締め切り: 6/19 13:00 (日本標準時)

課題1 (必須)

1. 整数 n を受け取って n の階乗を計算して返す関数 fact: int → int を書け
2. 非負整数2つの最大公約数を求める関数 gcd : int → int → int を書け
3. 整数 n を受け取って n が素数であれば true を素数でなければ false を返す関数 isprime: int → bool を書け
   - 考察不要

課題2 (必須)

1. 関数 f を受け取って, f を 2 個合成する関数を返す関数 double を書け
2. 関数 f と整数 n を受け取って, f を n 個合成する関数を返す関数 repeat を書け
   - 考察不要

課題2の例

```
# let add2 =(* fun x -> x + 1 を2つ合成 *)
      double (fun x -> x + 1);;
val add2 : int -> int = <fun>
# add2 3;;
- : int = 5
# let add5 =(* fun x -> x + 1 を5つ合成 *)
      repeat (fun x -> x + 1) 5;;
val add5 : int -> int = <fun>
# add5 3;;
- : int = 8
```

課題3 (Optional)

- 関数 f を受け取って, f を再帰的に無限回合成する関数を返す関数 inf を書け
  - 以下のようなイメージ

```
  inf f ≡ f (f (f (f (f (f (f ...))))))
# let fib = inf (fun g n ->
                 if n <= 2 then 1
                 else g (n - 1) + g (n - 2));;
  val fib : int -> int = <fun>
# fib 10;;
  - : int = 55
```

  - let rec を使って書いてよい

課題4 (Optional)

- 関数の生成と適用のみを用いて自然数を表現する課題
- 詳細は別紙参照のこと

レポート提出上の注意 (再掲)

- 提出方法: 電子メール
  - 宛先: ml-report-2007@yl.is.s.u-tokyo.ac.jp
    - 受領通知が届くと思うので確認のこと
- Subject を
  Report <レポート番号> <学生証番号>
  とすること
  - 今回の場合 Report 1 710xx