




# ML 演習 第 4 回

---

おおいわ  
May 1, 2001

# 今回の内容 (1)

- 各種補足
  - 識別子の文法について
  - alias pattern
  - 等値演算子: = と ==
    - structural equivalence と physical equivalence
  - 比較演算子
  - value restriction (参考資料)



# 今回の内容 (2)

- 
- 
- 
- 
- Ocaml のモジュールシステム
  - structure
  - signature
  - functor

# 識別子について

- 利用可能文字
  - 先頭文字: A~Z, a~z, \_ (小文字扱い)
  - 2文字目以降: A~Z, a~z, 0~9, \_, ' ,
- 先頭の文字の case で2つに区別
  - 小文字: 変数, 型名, レコードの field 名  
(ラベル, クラス名, クラスメソッド名)
  - 大文字: Constructor 名, モジュール名
  - 任意: モジュール型名

# alias pattern

## ■ パターンマッチの結果に別名を与える

```
# match (1, (2, 3)) with (x, (y, z as a)) -> a  
- : int * int = (2, 3)
```

- 結合が弱いので注意。必要なら ( ) を。  
(上の例では y, (z as a) ではなく  
(y, z) as a と結合している)

# 等値演算子 (1)

## ■ 2つの等値演算子

- =: 「構造的な一致」 (否定: <>)
  - scheme の equal?
- ==: 「物理的な一致」 (否定: !=)
  - scheme の eq?
- == の方が識別力が強い

# 等值演算子 (2)

## ■ 例

```
# let test x y = (x = y, x == y)
val test : 'a -> 'a -> bool * bool
# test 1 1;;
- : bool * bool = true, true
# test 1.0 1.0;;
- : bool * bool = true, false
# test "string" "string";;
- : bool * bool = true, false
# test (ref 1) (ref 1);;
- : bool * bool = true, false
```

# 等值演算子 (3)

```
# (fun x -> x) = (fun x -> x);;
```

Uncaught exception:

Invalid\_argument "equal: functional value".

```
# (fun x -> x) == (fun x -> x);;
```

```
- : bool = false
```

```
# let f = (fun x -> x) in test x x;;
```

```
- : bool = true, true
```

```
# let r = (ref 1) in test r r;;
```

```
- : bool = true, true
```

```
# let (x, y) as pair = (ref 1, ref 2) in x (fst pair);;
```

```
- : bool = true, true
```



# 比較演算子

- 多相比較演算子  $<$ ,  $>$ ,  $<=$ ,  $>=$ 
  - 型:  $\alpha \rightarrow \alpha \rightarrow \text{bool}$
  - 整数・実数  $\rightarrow$  数値比較
  - 文字列・文字  $\rightarrow$  辞書順比較
  - その他のオブジェクト  $\rightarrow$  実装依存
    - 作成時に順序が決定
    - 内容変更しても順序は入れ替わらない
      - $\rightarrow$  MultiSet のような構造に利用可能
        - 注: 循環参照を持つデータでは止まらないことがある

# 大規模プログラミングと モジュール

- 大規模プログラミングに必要な機能
  - 名前の衝突の回避
    - 適切な「名前空間」の分離
  - 仕様と実装の切り分けの明確化
    - 細かい実装の変更から利用者を守る
    - 仕様を変えない範囲で実装の変更を自由にする

# Ocaml の モジュールシステム

- structure : 名前空間を提供
  - プログラムをモジュールとして分離
- signature : interface 仕様を定義
  - プログラムの実装の隠蔽
- functor : structure に対する「関数」
  - 共通の構造をもった structure の生成

# structure (1)

- 変数や型などの定義の集合
  - 例: MultiSet (lecture4-1.ml)
  - 内部の変数には . 表記でアクセス

```
# MultiSet.empty;;
```

```
- : 'a MultiSet.set = MultiSet.Leaf
```

```
# let a = MultiSet.add MultiSet.empty 5;;
```

```
val a : int MultiSet.set = MultiSet.Node  
                                (5, MultiSet.Leaf, MultiSet.Leaf)
```

```
# MultiSet.member a 5;;
```

```
- : bool = true
```

# structure (2)

- open: structure を「開く」
  - structure 内の定義を . 無しでアクセス

```
# open MultiSet;;
```

```
# add empty 5;;
```

```
- : int MultiSet.set = MultiSet.Node  
                                (5, MultiSet.Leaf, MultiSet.Leaf)
```

```
# member (add empty 5) 10;;
```

```
- : bool = false
```

# signature

- structure に対する「型」
  - 公開する/隠蔽する変数や型の指定
  - 例: MULTISSET: 重複集合の抽象化
    - type 'a set は存在**だけ**が示されている
    - remove\_top は定義にない

# signature の適用 (1)

## ■ signature を structure に適用

```
# module AbstractMultiSet = (MultiSet : MULTISET);;  
module AbstractMultiSet : MULTISET  
# let a = AbstractMultiSet.empty;;  
val a : 'a AbstractMultiSet.set = <abstr>  
# let b = AbstractMultiSet.add b 5;;  
val b : int AbstractMultiSet.set = <abstr>
```

抽象データ型の内容は隠蔽される

# signature の適用 (2)

```
# open AbstractMultiSet;;  
# let a = add (add empty 5) 10;;  
val a : int AbstractMultiSet = <abstr>  
# AbstractMultiSet.remove_top;;  
Unbound value AbstractMultiSet.remove_top;;  
# MultiSet.remove_top a;;  
This expression has type int AbstractMultiSet.set  
but it is used with type 'a MultiSet.set
```



# functor の定義

- structure から structure への「関数」
  - 例: lecture4-2.ml
    - signature ORDERED\_TYPE
      - 一般の全順序・等値関係つきの型
    - functor MultiSet2
      - ORDERED\_TYPE を持つ structure に対する集合の定義

# functor と signature

- functor に対する signature の定義
  - SETFUNCTOR: MultiSet2 に対する  
functor signature
    - elem の型は concrete (Elt.t)
    - t の型は abstract
  - AbstractSet2: SETFUNCTOR で制限した  
functor MultiSet2

# functor & signature (2)

```
# module AbstractStringSet =  
    AbstractSet2(OrderedString);;  
module AbstractStringSet : sig ... end  
# let sa = AbstractStringSet.add  
    AbstractStringSet.empty "OCaml";;  
val sa : AbstractStringSet.t = <abstr>  
# AbstractStringSet.member sa "ocaml";;  
- : bool = false
```

# functor & signature (3)

```
# module NCStringSet = AbstractSet2(NCString);;
```

```
module NCStringSet : sig ... end
```

```
# let sa = NCStringSet.add NCStringSet.empty  
  "OCaml";;
```

```
val sa : NCStringSet.t = <abstr>
```

```
# NCStringSet.member sa "ocaml";;
```

```
- : bool = true
```

```
# AbstractStringSet.add sa "ocaml";;
```

This expression has type `NCStringSet.t = AbstractSet2(NCString.t)` but is here used with type `AbstractStringSet.t = AbstractSet2(OrderedString.t)`

# 課題1

- リストなどの別のデータ構造を使って signature MULTISSET に対する別の実装を与えよ。
  - structure の書き方の練習。そんなに難しくはないと思います。

## 課題2

- lecture4-ex2 は簡単なパスワード付き銀行口座の例であるが、fst a1 や BankAccountImpl.accounts など、秘密の情報である暗証や口座一覧が操作可能である。そこで、この module に適用する signature を作り、これらの情報を隠蔽せよ。
  - signature の練習。割と簡単。

## 課題3 (optional)

- ORDERED\_TYPE で表現される型の key と、任意の型の値についての連想配列を作り出す functor を作れ。
  - functor の練習。前2問よりは難しいか?

# 課題3 (例1)

```
# module NCStringAssociation = Association(NCString);;
module NCStringAssociation :
  sig
    type key = NCString.t
    and 'a t = 'a Association(NCString).t
    val empty : 'a t
    val add : 'a t -> key -> 'a -> 'a t
    val remove : 'a t -> key -> 'a t
    val get : 'a t -> key -> 'a
    exception Not_Found
  end
```



## 課題3 (例2)

```
# open NCStringAssociation;;  
# let sa = add empty "C" "/* */";;  
val sa : string NCStringAssociation.t = <abstr>  
# let sa = add sa "OCaml" "(* *)";;  
val sa : string NCStringAssociation.t = <abstr>  
# let sa = add sa "Perl" "#";;  
val sa : string NCStringAssociation.t = <abstr>  
# get sa "ocaml";;  
- : string = "(* *)"
```



# 提出方法

- 〆切: 2001年5月14日 月曜日 24:00
- 提出先: [ml-report@yl.is.s.u-tokyo.ac.jp](mailto:ml-report@yl.is.s.u-tokyo.ac.jp)
- 題名: Report 4 (学生証番号)