



ML 演習 第 4 回

おおいわ
Mar 6, 2003

今回の内容

- 補足
- Ocaml のモジュールシステム
 - structure
 - signature
 - functor
- Ocaml コンパイラの利用

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)

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
 - 部品の再利用
 - 同じ構造を持つコードを共通化する

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 (値・型など) 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 = Node (5, Leaf, 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: MULTiset: 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 : MULTISSET);;
```

```
module AbstractMultiSet : MULTISSET
```

```
# let a = AbstractMultiSet.empty;;
```

```
val a : 'a AbstractMultiSet.set = <abstr>
```

```
# let b = AbstractMultiSet.add a 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.set = <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 と 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 と 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)
```

Ocaml のコンパイラ (1)

- モジュール単位の分割コンパイルをサポート
- Unix の実行形式ファイルを作成
 - 複数の backend
 - ocamlc: バイトコードコンパイラ
 - バイトコードインタプリタ (ocamlrun) を実行に使用
 - デバッガをサポート
 - ocamlpt: ネイティブコードコンパイラ
 - SPARC や x86 などの機械語を直接生成

Ocaml のコンパイラ (2)

■ 拡張子一覧

■ ソースファイル

- .ml → module の実装 (structure)
- .mli → module のインタフェース (signature)

■ オブジェクトファイル

- .cmo → 実装のバイトコード
- .cmi → インタフェース定義のバイトコード
- .cmx → 実装のネイティブコード (.o と一組)
- .cma, .cmxa → ライブラリ

分割コンパイル (1)

- *.ml と *.mli
 - 実装とインタフェースをそれぞれ記述
 - module Something :
sig [something.mli の内容] end
= struct [something.ml の内容] end
に相当 (モジュール名の先頭を小文字化)
 - .mli をコンパイル → .cmi を生成
 - .ml をコンパイル → .cmi が無ければ
制約無しで生成、あれば型チェック

分割コンパイル (2)

■ 例

- mySet.mli, mySet.ml
 - module MySet の定義
- uniq.ml
 - メインプログラムのモジュール

分割コンパイル (3)

■ 実行例 (1)

```
% ocamlc -c mySet.mli
```

```
% ocamlc -c mySet.ml
```

```
% ocamlc -c uniq.ml
```

```
% ls -F *.cm*
```

```
mySet.cmi mySet.cmo uniq.cmi uniq.cmo
```

```
% ocamlc -o myuniq mySet.cmo uniq.cmo
```

```
% ls -F myuniq
```

```
myuniq*
```

順序が重要:
モジュールの定義/依存順

分割コンパイル (4)

■ 実行例 (1)

% ./myuniq

OCaml

Standard ML

C++

OCaml

^D

1 C++

2 OCaml

1 Standard ML

%

分割コンパイル (5)

■ .cmo ファイルのインタプリタでの利用

```
# #load "mySet.cmo";;
```

```
# MySet.empty;;
```

```
- : 'a MySet.set = <abstr>
```

```
# MySet.remove_top;;
```

```
Unbound value MySet.remove_top
```

```
# open MySet;;
```

```
# empty;;
```

```
- : 'a MySet.set = <abstr>
```

課題0

- myuniq の例を自分でやってみること。
 - 実行ファイル生成
 - mySet.cmo をインタプリタに読み込んでみる
- 次回以降使えないと困ります。
- 自習課題です。

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2 (仕様)

- 公開すべきもの
 - deposit, withdraw, balance, bank_statistics
 - 型 account の存在
- 隠蔽すべきもの
 - 型 t の実装: 残高を操作できる
 - 値 accounts: 他口座の instance が得られる
 - その他の関数: 認証を回避できる

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: 2003年5月20日 火曜日 13:00
- 提出先: ml-report@yl.is.s.u-tokyo.ac.jp
- 題名: “Report 4 xxxxx” (学生証番号)