Functional and logic programming lab 3rd report

Yoshiki Fujiwara, 05-191023

1　Q.1: 分割コンパイル

1.1

まず、正常に動作するコードを示す。

Listing 1　正常に動作するコード

```
sort: strSet.cmi strSet.cmo sort.cmo
  ocamlc -o sort strSet.cmo sort.cmo
strSet.cmi: strSet.mli
    ocamlc -c strSet.mli
strSet.cmo: strSet.ml
    ocamlc -c strSet.ml
sort.cmo: sort.ml
    ocamlc -c sort.ml

clean:
    rm -rf *.cmi *.cmo sort
```

1.2

次に、.cmo file をインタプリタで使ってみる。いかに出力を表す。

Listing 2　出力

```
#load "strSet.cmo";;
# StrSet.empty ;;
- : StrSet.t = <abstr>
# StrSet.count_sub ;;
```

```
Error: Unbound value StrSet.count_sub
```

strSet.mli に定義していない関数は使えない、すなわち、count_sub は module となっていない。

このことから、mli file をコンパイルせずに、.cmo file を作ると結果が違うこともわかり、実際にそれを試した結果が以下となる。

Listing 3　出力

```
#load "hoge.cmo";;
# Hoge.count_sub;;
- : String.t -> int -> String.t list -> int = <fun>
```

cmi file が自動生成されたが、module として定義できていない。

このとき、ocaml -c hoge.ml を実行するときに、同じ名前の mli file(hoge.mli) が同じ directory 内に存在して、なおかつ.cmi の file を作成していないとエラーを返すので、その点には注意をして実行する必要がある。

1.3

次に mli file をコンパイルせずに、make を行なってみる。

Listing 4　make file の中身

```
sort: strSet.cmo sort.cmo
    ocamlc -o sort strSet.cmo sort.cmo
strSet.cmo: strSet.ml
    ocamlc -c strSet.ml
sort.cmo: sort.ml
    ocamlc -c sort.ml


clean:
    rm -rf *.cmi *.cmo sort
```

これに対してはエラーを返す。make: *** No rule to make target ‘strSet.cmi’, needed by ‘sort’. Stop.

これは sort で、strSet の関数を使用しようとするが、cmi file がないので、signiture が定義できず、エラーが起きる。

1.4

sort.cmo を先に作ってみる

Listing 5　make file の中身

```
sort: sort.cmo strSet.cmi strSet.cmo
  ocamlc -o sort sort.cmo strSet.cmo
strSet.cmo: strSet.ml
  ocamlc -c strSet.ml
strSet.cmi: strSet.mli
  ocamlc -c strSet.mli
sort.cmo: sort.ml
  ocamlc -c sort.ml

clean:
  rm -rf *.cmi *.cmo sort
```

これもエラーを返す。make: *** [sort.cmo] Error 2

sort.cmo を作る際に、strSet と言う module を参照するが、参照できないため、エラーを起こしてしまう。

1.5

リンク時の順番を入れ替えてみる。

Listing 6　make file の中身

```
sort: strSet.cmi strSet.cmo sort.cmo
  ocamlc -o sort sort.cmo strSet.cmo
strSet.cmo: strSet.ml
  ocamlc -c strSet.ml
strSet.cmi: strSet.mli
  ocamlc -c strSet.mli
sort.cmo: sort.ml
  ocamlc -c sort.ml

clean:
  rm -rf *.cmi *.cmo sort
```

File "_none_", line 1: Error: Required module 'StrSet' is unavailable make: *** [sort] Error 2

これも、依存関係に違反しているため、エラーを返す。

1.6

Ocaml Make file を ocaml make file という名前で保存した。

2 Q.2: スタックの実装

2.1 動作例

"make"を行ったあと、./main を実行すると 7,1 と表示される。

Listing 7 main.ml の中身の一部

```
let s = empty;;
let t = push 5 s;;
Printf.printf "%d\n" (size t);;

let u = push 7 t;;
let v = pop u;;
let w = fst(v);;
Printf.printf "%d\n" (w);;
```

このような動作を行い、出力は 7 と 1 となっている。

2.2 考察

この課題では分割コンパイルを試してみるべく、実装を main.ml と stack.ml と stack.mli に分けて行うことにした。分割コンパイルの際は、sig - end、 struct - end の中身の部分のみを書くことに注意しなければならなかった。また、コンパイルの順番は Q.1 同様注意しなければならなかった。

今回、stack.ml の中身では list を使った実装を行った。しかし、このことは、module 化しているので、使うユーザーの視点からはわからないものとなっている。

3 Q.3: functor

3.1 動作例

3.ml を use する

Listing 8 動作例

```
let s = Bst.empty;;
val s : Bst.t = <abstr>

let t = Bst.add 1 s;;
val t : Bst.t = <abstr>

let u = Bst.add 2 t;;
val u : Bst.t = <abstr>

let v = Bst.add 5 u;;
val v : Bst.t = <abstr>

let w = Bst.add 4 v;;
val w : Bst.t = <abstr>

let x = Bst.remove 4 w;;
val x : Bst.t = <abstr>

Bst.count 4 x;;
- : int = 0

# Bst.count 5 x;;
- : int = 1
```

3.2 考察

二分木を実装した。木の作り方は前回の課題で行った方法と同じ variant を使う方法を用いた。add と count は愚直に実装するだけであった。remove について、消したいノードの下に子のノードが二つあった時にするべきことは右の子の最小値を持ってきて、消し

たノードの場所に入れることである。searchmin という関数で最小値を探し、その値を再帰呼び出しで remove するという方法で実装した。

functor は別の module でパラメータ化される module である。module の動作にそぐわないものが入力されたときにエラーを返すことができるようになる。例えばここでは、compare ができないものが入力できないようになっている。

4 Q4: 連想配列

4.1 動作例

Listing 9 動作例

```
let s = Map.empty;;
val s : 'a Map.t = <abstr>

let t = Map.add 1 "one" s;;
val t : string Map.t = <abstr>

let u = Map.add 2 "two" t;;
val u : string Map.t = <abstr>

let v = Map.add 5 "five" u;;
val v : string Map.t = <abstr>

let w = Map.add 4 "four" v;;
val w : string Map.t = <abstr>

Map.lookup 4 w;;
- : string = "four"

let x = Map.remove 4 w;;
val x : string Map.t = <abstr>

Map.lookup 4 x;;
Exception: Multiset2(T).Eval_error.
```

4.2 考察

Q.3 で二分木を実装していたので、それを用いた実装を行なった。注意した点は、同じ key の値が入力されたときに上書きするように実装したことである。

5 Q.5: Matrix module

5.1 動作例

Listing 10 Bool matrix の加算

```
let x = Boolmatrix.create[[true;false;];[true;false;];];;

let y = Boolmatrix.create [[false; true]; [true; false]];;

let z = Boolmatrix.addition x y;;

Boolmatrix.show z;;

output
- : Bool_ring.t list list = [[true; true]; [true; false]]
```

この演算は以下に対応している。

$$\begin{pmatrix} true & false \\ true & false \end{pmatrix} + \begin{pmatrix} false & true \\ true & false \end{pmatrix} = \begin{pmatrix} true & true \\ true & false \end{pmatrix}$$

この後も同様に、リストのリストとして入力を与える仕様となっている。

Listing 11 Bool matrix と vector の乗算

```
let x = Boolmatrix.create [[true;false;true];[false;true;true;];
[true;true;true];];;

let y = Boolmatrix.create [[true;];[false;];[false;];];;

let z = Boolmatrix.multiplication x y;;

Boolmatrix.show z;;
```

```
output
- : Bool_ring.t list list = [[true]; [false]; [true]]
```

$$\begin{pmatrix} true & false & true \\ false & true & true \\ true & true & true \end{pmatrix} * \begin{pmatrix} true \\ false \\ false \end{pmatrix} = \begin{pmatrix} true & false & true \end{pmatrix}$$

Listing 12　Bool matrix 同士の乗算

```
let x = Boolmatrix.create[[true;false;];[false;false;];];;

let y = Boolmatrix.create [[true; true]; [true; true]];;

let z = Boolmatrix.multiplication x y;;

Boolmatrix.show z;;

output
- : Bool_ring.t list list = [[true; true]; [false; false]]
```

課題の要件ではないが、行列同士の乗算も実装した。

Listing 13　tropical matrix 同士の加算

```
let x = Tmatrix.create [[Int 3; Inf; Int 5;];[Int 2;Int 3;Int 4;];
[Int 9;Inf;Int 4;];];;

let y = Tmatrix.create [[Int 2; Inf; Int 5;];[Inf; Int 4; Int 9;];
[Int 12; Int 8; Int 3;];];;

let z = Tmatrix.addition x y;;

Tmatrix.show z;;

output
- : Tropical_ring.t list list =
[[Tropical_ring.Int 2; Tropical_ring.Inf; Tropical_ring.Int 5];
 [Tropical_ring.Int 2; Tropical_ring.Int 3; Tropical_ring.Int 4];
 [Tropical_ring.Int 9; Tropical_ring.Int 8; Tropical_ring.Int 3]]
```

Listing 14　tropical matrix と vector の乗算

```
let x = Tmatrix.create [[Int 9; Inf;];[Int 2; Int 3;];];;

let y = Tmatrix.create [[Int 7;] ;[Int 5;];];;

let z = Tmatrix.multiplication x y;;

Tmatrix.show z;;

output
[[Tropical_ring.Int 16]; [Tropical_ring.Int 8]]
```

Listing 15　tropical matrix 同士の乗算

```
let x = Tmatrix.create [[Int 5;Int 2;Inf;];[Int 8;Int 3;Int 7;];
[Inf; Int 5; Int 2;];];;

let y = Tmatrix.create [[Inf; Int 9; Int 5;];[Inf; Int 8; Int 3;];
[Int 3; Int 2; Int 9;];];;

let z = Tmatrix.multiplication x y;;

Tmatrix.show z;;

output
- : Tropical_ring.t list list =
[[Tropical_ring.Inf; Tropical_ring.Int 10; Tropical_ring.Int 5];
 [Tropical_ring.Int 10; Tropical_ring.Int 9; Tropical_ring.Int 6];
 [Tropical_ring.Int 5; Tropical_ring.Int 4; Tropical_ring.Int 8]]
```

以上が動作例となる。

5.2　考察

今回は 1 行 n 列の行列として横ベクトルを、n 行 1 列の行列として縦ベクトルを、それぞれ扱うこととして実装した。中の実装もリストのリストとして行なった。行列の和や積が定義されない入力が与えられるとエラーが出力されるようになっている。module 化し

ているので、リストのリストとしてユーザーは入力するが、実際の中身がリストのリストとして動いているのかはわからないようになっている。

tropical semi ring の実装については、inf を別に定義する必要があったため、variant 型を用いて実装を行なった。

6 Q.6: EQ1

6.1 動作例

Listing 16 EQ1 の出力

```
let c1 = EConstInt (Eq.refl, 1);;
val c1 : int expr = EConstInt ((<fun>, <fun>), 1)

let c2 = EConstInt (Eq.refl, 2);;
val c2 : int expr = EConstInt ((<fun>, <fun>), 2)

let add12 = EAdd (Eq.refl, c1, c2);;
val add12 : int expr =
  EAdd ((<fun>, <fun>), EConstInt ((<fun>, <fun>), 1),
   EConstInt ((<fun>, <fun>), 2))

let ct = EConstBool (Eq.refl, true);;
val ct : bool expr = EConstBool ((<fun>, <fun>), true)

let cif = EIf (ct, add12, c2);;
val cif : int expr =
  EIf (EConstBool ((<fun>, <fun>), true),
   EAdd ((<fun>, <fun>), EConstInt ((<fun>, <fun>), 1),
    EConstInt ((<fun>, <fun>), 2)),
   EConstInt ((<fun>, <fun>), 2))

let c1 = EConstInt (Eq.refl, 1);;
val c1 : int expr = EConstInt ((<fun>, <fun>), 1)

let ct = EConstBool (Eq.refl, true);;
val ct : bool expr = EConstBool ((<fun>, <fun>), true)
```

```
let cadd = EAdd (Eq.refl, c1, ct);;
Error: This expression has type bool expr
       but an expression was expected of type int expr
       Type bool is not compatible with type int
```

6.2 考察

この方法を用いると、型が整合しない式を事前に省くことが可能となる。前回の式だと、以下の式のようになり、EAdd は定義できてしまっている。

Listing 17 前回の問 5

```
# let c1 = EConstInt (1);;
val c1 : expr = EConstInt 1
# let c2 = EConstBool (true);;
val c2 : expr = EConstBool true
# let cadd = EAdd (c1,c2);;
val cadd : expr = EAdd (EConstInt 1, EConstBool true)
```

この問題では、Eq module を作成することで、式を評価する前に、定義できない式を省くことができるようになっている。