『プログラム言語論』 期末試験 解答 (例) 2017/7/13 改訂版

問 1. (配点 35 点)

授業の演習では、micro-ML 言語 (一階の関数型言語、ML のサブセット) に対する環境渡しインタプリタを用いた。これに関して以下の設問に答えなさい。

1-a. 下の図は, Fibonacci 関数 f (つまり、$f(1) = f(2) = 1$ および $x > 2$ となる x に対して $f(x) = f(x-2)+f(x-1)$ となる関数) を定義して，$f(4)$ を計算するプログラム (左が OCaml, 右が micro-ML) である。ただし、整数の大小比較>=をプリミティブ演算として持つよう拡張されているものとする。

このプログラムを、演習で用いた環境渡し方式のインタプリタ (eval) で実行したとき、$f(4)$ の実行過程で、環境がどのように変化するか書きなさい．つまり，環境が変化するごとに，その環境を書きなさい。ただし、最初の時点では環境は空であるとし、静的束縛かつ値呼びであるとする。環境の表記は自分流でよいが、たとえば、$[x \to 10,\ x \to 20]$ と書けばよい。

```
let rec f x =
  if x <= 2 then 1
  else f (x-2) + f (x-1)
in   f 4
```

```
Letfun("f", "x",
 If(Prim("<=", Var("x"), CstI(2)),
   CstI(1),
   Prim("+",Call(Var("f"),Prim("-",Var("x"),CstI(2))),
            Call(Var("f"),Prim("-",Var("x"),CstI(1))))),
 Call(Var("f"),CstI(4)))
```

1-a の答 (10 点)

最も精密に書けば以下の通りとなる．(答案では，ある程度，省略していてもよい．また，本問では、f に対する束縛を書いていなくても良いものとする．)

[2017/7/13] 以前のバージョンの「解答例」では、f の定義を間違えていたため、以下のものと (少しだけ) 違うものを書いてしまったので修正した。

	初期値	[ ]
1.	f 定義後	[f=Closure(”f”...)]
2.	f(4) 呼び出し	[x=4; f=Closure(”f”...)]
3.	f(2) 呼び出し	[x=2; f=Closure(”f”...)]
4.	f(2) が終了	[x=4; f=Closure(”f”...)]
5.	f(3) 呼び出し	[x=3; f=Closure(”f”...)]
6.	f(1) 呼び出し	[x=1; f=Closure(”f”...)]
7.	f(1) が終了	[x=3; f=Closure(”f”...)]
8.	f(2) 呼び出し	[x=2; f=Closure(”f”...)]
9.	f(2) が終了	[x=3; f=Closure(”f”...)]
10.	f(3) が終了	[x=4; f=Closure(”f”...)]
11.	f(4) が終了	[f=Closure(”f”...)]
12.	全部終了	[ ]

ポイントの解説: 再帰的に定義された関数なので，環境が [x=1; x=2; x=4; f=...] のように，どんどん大きくなっていくように思うかもしれないが、それは間違いである．

静的束縛なので、実行時の環境は，もとのプログラムにおける束縛変数と同じ数の束縛しかもたない。上記の例では、関数 f の定義の本体では，変数の束縛は f 自身と引数 x に対するものだけなので、f の実行中の環境も，[x=...; f=...] といった形であり，x に対する束縛が 2 つ以上重なることはない．(f の束縛を省略するのは減点しないが、x の束縛が 2 個以上ある場合は、静的束縛における環境がわかっていないので、この問題の答えとしては 0 点である。)

1-b. 前問のように静的束縛かつ値呼びで $f(4)$ を実行すると，足し算 (加算 +) が合計して何回実行されるか、答えなさい。

1-b の答 (5 点 これは，f の数学的定義さえわかればよく，$f(4) = f(2)+f(3) = 1+(f(2)+f(1)) = 1+(1+1) = 3$ であり，結局 1 を足しているだけなので値が 3 ということは，2 回足し算が行われた.

[2017/7/13] この問題でも、以前のバージョンの「解答例」では、f の定義を間違えていたため、答えが違っていたので修正した。

1-c. 下のように数学的表記で定義される関数 (を micro-ML で書いたもの) を g および h とする。

$$g(x) = x * x + x * 2 + 1$$
$$h(x) = 13$$

これらの定義のもとに $g(g(4))$ および $h(g(4))$ を実行するとき、(1) 静的束縛かつ値呼びのとき、(2) 静的束縛かつ名前呼びのとき、(3) 静的束縛かつ必要呼びのとき、の 3 通りの場合において、足し算が実行される回数を答えなさい。($g(g(4))$ と $h(g(4))$ の 2 つに対して、3 通りがあるので、合計で 6 個の数を答えることになる。)

1-c の答 (10 点) 基本問題なので答えだけを以下に記す.

	$g(g(4))$	$h(g(4))$
(1)	4	2
(2)	8	0
(3)	4	0

1-d. 今度は、micro-ML を、高階関数を持つ言語に拡張した言語を考える．これを micro-ML+と呼ぶことにして，その言語で記述された以下のプログラムの実行について考える。 (左は OCaml での記述，右は micro-ML+での記述)

```
let x = 3 in
let f y = x + y in
let x = 5 in
let g z = x + (f z) in
let x = 7 in
    f (g 20)
```

```
Let("x", CstI(3),
Letfun("f","y",Prim("+",Var("x"),Var("y")),
Let("x", CstI(5),
Letfun("g","z",Prim("+",Var("x"),Call(Var("f"),Var("z"))),
Let("x", CstI(7),
 Call(Var("f"),Call(Var("g"),CstI(20)))))))
```

このプログラムを静的束縛かつ値呼びで実行する．最後の行である f(g 20) を実行する直前の環境，および，上記プログラムの実行結果 (返す値) を書きなさい．関数クロージャは Closure(...) といった形で適宜書けばよい.

1-d の答 (5 点) f(g 20) を実行する直前の環境は以下のものである．(クロージャは C(....) と表記する．)

[x=7; g=C(g,z,x+(f z),E); x=5; f=C(f,y,x+y,[x=3]); x=3]

ただし E は環境 [x=5; f=C(f,y,x+y,[x=3]); x=3] である.

また，上記プログラムを静的束縛，値呼びで実行したときに返す値は,

$$f(g(20)) = f(5 + (f(20))) = f(5 + (3 + 20)) = f(28) = 3 + 28 = 31$$

である．ここで f や g の定義に含まれる自由変数の値は，定義された瞬間のものが使われることに注意せよ.

1-e. 前問のプログラムを、動的束縛かつ値呼びで実行したとき、計算結果が何になるか答えなさい。(環境を書く必要はない．)

1-e の答 (5 点) 動的束縛であるので，式を (自由変数こみで) そのまま展開していけばよい．f(g 20) を実行する瞬間の x の値は 7 である.

この計算結果は以下の通りである．

$$f(g(20)) = f(x + (f(20))) = f(7 + (f(20))) = f(7 + (x + 20)) = f(7 + (7 + 20)) = f(34) = x + 34 = 41$$

問 2. (配点 26 点)

Java は静的型付けを行なうオブジェクト指向言語である．下記の Java プログラムについて考えなさい。ただし、String は文字列型を表し、(後のプログラムで) System.out.println(s) は 文字列 s を印刷するメソッドである．ClassB は ClassA を継承している。toString, foo 等はメソッドの名前である。public というキーワードは無視してよい．

```
class ClassA {
  public String toString () { return "ClassA";    }
  public String foo ()      { return "A-foo";     }
  public String goo ()      { return "A-goo";     }
  ClassA ()                 {} // construcing an object of ClassA
}
class ClassB extends ClassA {  // inheritance
  public String toString () { return "ClassB";    }  // overwrite
  public String hoo ()      { return "B-hoo";     }
  public String foo (String s){ return "B-foo";   }  // overload
  ClassB ()                 {} // construcing an object of ClassB
}
```

2-a. 上記プログラムとあわせて以下のプログラムを実行する．以下のプログラムで testN-M というコメントがあるすべての行について、(1) コンパイルエラーを起こす，(2) 実行時エラーを起こす，(3) エラーは起こさない，のいずれであるか、また、System.out.println がある行がエラーを起こさない場合、何が印刷されるかを答えなさい。なお、たとえば test2-1 がコンパイルエラーのときは、その行と一緒にコンパイルする必要がある test2-2 などの行もコンパイルエラーと判断しなさい。

```
class Test1 {
  public static void main(String args[]) {
  ClassA  a;  ClassB  b;      // declare variables
  ClassA  a2;  ClassB  b2;
  a = new ClassA();   // create an object of ClassA and store it in a
  b = new ClassB();
  System.out.println(b.foo());       // test1-1
  System.out.println(b.foo("abc"));  // test1-2
  System.out.println(b.goo());       // test1-3
  a2 = b;                            // test2-1
  System.out.println(a2.toString()); // test2-2
  System.out.println(a2.foo());      // test2-3
  System.out.println(a2.hoo());      // test2-4
  b2 = a;                            // test3-1
  System.out.println(b2.toString()); // test3-2
  System.out.println(b2.foo());      // test3-3
  System.out.println(b2.hoo());      // test3-4
} }
```

2-a の答 (各 2 点、合計 16 点、ただし最後の text3-2 以降は採点対象外) 以下の通りである。表の右側に若干の解説をのせた。

テスト	結果	印刷される文字列	備考
test1-1	(3)	”A-foo”	継承 (オーバーロードでない)
test1-2	(3)	”B-foo”	オーバーロード
test1-3	(3)	”A-goo”	継承
test2-1	(3)		
test2-2	(3)	”ClassB”	動的ルックアップ
test2-3	(3)	”A-foo”	継承 (オーバーロードでない)
test2-4	(1)		クラス A 変数 a2 は hoo メソッドを持たない
test3-1	(1)		逆向きのサブタイピングはできない
test3-2	同上		
test3-3	同上		
test3-4	同上		

2-b. 上記プログラム例 (の適当な部分) を用いて，動的ルックアップとサブタイピングとは何かを、簡潔に説明しなさい.

2-b の答 (各 5 点、合計 10 点) 動的ルックアップは、メソッド名とその実体 (実装) の対応付け (ルックアップ) が動的に決定されることであり、上記の例では、test2-2 において、toString メソッドの実体が、プログラム中の変数 a2 の型 ClassA ではなく、実行時に a2 に格納されている値 (ClassB 型のオブジェクト) がもつメソッドであることから動的ルックアップであることがわかる。

サブタイピングは、ClassA の要素を書ける (プログラム中の) 場所に、ClassA のサブタイプ (部分型) の任意の要素も書いてよい、という意味である。上記の例では、a2 = b; という文がそれにあたり、ClassA 型の変数 a2 に、そのサブタイプである ClassB 型のオブジェクトを代入している。

問 3. (配点 15 点)

以下のプログラムに、型がつくか (型を整合的に与えることができるか) どうかを答えなさい。(YES/NO だけでなく、その理由と、型がつく場合はその型を答えなさい。) ただし、真理値型を `bool`, 整数型を `int` と記しなさい。

3-a. 以下の式の型付け (ヒント: この式は関数 f そのものを返そうとしているので、型がつくとすれば $A \to B$ の形である。)

(micro-ML+の構文)

```
Letfun("f","x",
       Prim("*",CstI(2),Call(Var("f"),Prim("+",Var("x"),CstI(3)))),
       Var("f"))
```

(OCaml の構文) `let rec f x = 2 * f (x + 3) in f`

3-a の答 (5 点) 型がつき、int → int である。まず、f の引数は x および x+3 であるが、(OCaml では) + は int 型の要素にしか適用できないので、x は int 型である。よって f の引数型は int である。f の返り値型は 2* f(x+3) とあることから、やはり int である。さらに、2*f(3+x) 全体の型も int であり、それは f の返り値の型と一致するので、型が全体として整合し、f は int → int という型となる。(注. このように言葉で長々と書かなくても、型推論図を書いてもよい。)

3-b. 以下の式の型付け

(micro-ML+の構文) `Letfun("f","x",Prim("+",Var("x"),CstI(1)),Call(Var("f"),Var("f")))`

(OCaml の構文) `let f x = x + 1 in f f`

3-b の答 (5 点) このプログラムは型がつかない。この理由を考えるため、まず、f の型を考えると int → int である。次に (f f) を考えると、左の f の型が int → int であるので、その引数は int 型でなければいけない。しかし、引数として f が与えられており、その型は int 型でないので、型が整合しない。

3-c. 以下の式の型付け (型がつくとすれば多相型を使う)

(micro-ML+の構文)

```
Letfun("f","x",Var("x"),
 If(Call(Var("f"),CstB(true)),
    Call(Var("f"),CstI(10)),
    Call(Var("f"),CstI(20))))
```

(OCaml の構文) `let f x = x in if (f true) then (f 10) else (f 20)`

3-c の答 (5 点) このプログラムに含まれる関数 f は多相型 $\forall\alpha.\ \alpha \to \alpha$ を持ち、(f true) では α=bool となり、(f 10) や (f 20) では α=int となる。これによってプログラム全体の型が整合し、最終的に返す値の型は、int である。(注. 型推論図を書いてもよい。)

問 4. (配点 24 点)

以下の問のうち 3 つ以上に、それぞれ、3 行以上答えなさい。(4 つとも答えた場合は、点数が良いものから 3 つを選ぶ。)

4-a コンパイラとインタプリタの定義を述べた上で、コンパイラで作成したコードの実行がインタプリタでの実行より高速であることが多い理由を説明しなさい。

4-a の答 (8 点) コンパイラは変換系あるいは翻訳系であり、言語 A で書かれたプログラムを言語 B で書かれたプログラムに変換するプログラムを指す。インタプリタは解釈系あるいは実行系であり、言語 A で書かれたプログラム (とそれに対する引数) を受けとり、実行結果を返すプログラムを指す。多くのコンパイラは、プログラム全体から得られる静的情報を収集して、それに基づいた最適化を行ったプログラムを出力する一方、多くのインタプリタは、そのような静的情報の収集をおこなわないため、実行効率は前者で得られたプログラムの方が良いことが多い。

4-b. 末尾呼び出しの関数の例を 1 つあげた上で (例を書く言語は OCaml でも micro-ML でも C 言語でも Lisp でもよい)、末尾呼び出しとは何か、また、末尾呼び出しはどのように有用なのかを簡潔に述べなさい。

4-b の答 (8 点) 以下の関数 f は末尾呼び出し (だけ) で構成されている関数であり、g はそうでない。

```
let rec f x = if x = 0 then f (x - 1) else f 5
let rec g x = if x = 0 then 1 else (g (x - 1)) * 1
```

末尾呼び出しは、関数を引数に適用するのが「計算の最後 (末尾)」に出現することであり、末尾呼び出し「のみ」で記述された関数は、実行時に呼び出し元にもどる必要がないので、現在の環境を破壊して関数呼び出しをしてしまってよく、再帰関数がループなどの繰返し構造と同程度の実行効率で実行できる。

4-c. 抽象データ型の概念と情報隠蔽について説明しなさい。また、これらが、どのように有用であるかを簡潔に述べなさい。

4-c の答 (8 点) 抽象データ型は、データ型を使うための「インタフェース」とそれに対する具体的な「実装」を分離したものであり、インタフェースは、そのデータ型を使う関数の名前、関数の型および、それらの関数群が見たす性質から構成される。抽象データ型を使うプログラムは、このインタフェースに含まれる情報だけを使うことが許され、具体的な実装に言及することはできないため、情報が隠蔽される。情報隠蔽により、抽象データ型を利用するプログラムと抽象データ型自身の実装を分離することができ、大規模プログラムの開発において分業しやすくなるという利点がある。(補足: プログラムの再利用においても、インタフェースが明示されている点は有利である。)

4-d. 多相型の一種であるパラメータ多相とサブタイプ多相について説明しなさい。また、これらが、どのように有用であるかを簡潔に述べなさい。

4-d の答 (8 点) パラメータ多相は、プログラムの型が、「任意の型」をあらわす型変数を含むことができる仕組みであり、たとえば、$\lambda x.x$ というプログラムの型は、型変数 α を使って $\alpha \to \alpha$ と表示することができる。さらに α が任意の型を表すことを明示するため $\forall\alpha.\ \alpha \to \alpha$ と表す体系もある。パラメータ多相は ML,Haskell などの関数型言語のほか Java Generics など他のプログラム言語でも利用可能なものがある。

サブタイプ多相は、オブジェクト指向言語において、プログラムの型が、「サブタイプ関係にある任意の型」をあらわす型変数を含むことができる仕組みである。たとえば、この試験問題の問 2 の toString メソッドは、(そのメソッドを受け取るオブジェクト自身を引数に取った関数であると考えると) ClassA の任意のサブタイプ β に対して、$\beta \to \text{unit} \to \text{String}$ という型を持つと考えられる。

以上.