主専攻実験[S-8] 関数プログラミング

課題5-1,5-2,5-3

情報科学類 202113564 三村潤之介

2021/12/03 再提出

# 課題5-1

・示されたプログラムを静的束縛および動的束縛のもとで、手で実行し、計算結果を導け。

最後の行の関数適用部から等号でつないで計算する。

<静的束縛>

<動的束縛>

＜以下変更部＞

let x=10 in x=10

let f = fun y -> x+y in x=10, f=y->x+y

let y = f x in x=10, f=y->x+y, y=(x+10)=20

let x = 20 in x=20, f=y->x+y, y=20

f(x\*y) =420

＜ここまで＞

課題5-2

・示されたプログラムをミニOCaml言語のプログラムとして表現し、eval4で計算せよ。

示されたプログラムをミニOCaml言語のプログラムとして表現し、実行すると以下のようになる。ここで、eは何も変数が宣言されていない環境である。

# eval4

(

Let("x", IntLit(1),

Let("f", Fun("y", Plus(Var("x"),Var("y"))),

Let("x", IntLit(2),

App(Var("f") , Plus(Var("x"), IntLit(3)))

)

)

)

)

(e)

;;

- : value = IntVal 6

計算結果として、IntVal 6が返ってくることが確かめられた。

・eval4のAPPケースで、env1をenvに変更したとするとき、プログラム例がどう計算されるか確かめよ。

以下に計算結果を示す。

# eval4

(

Let("x", IntLit(1),

Let ("f", Fun("y", Plus(Var("x"),Var("y"))),

Let("x", IntLit(2),

App(Var("f") , Plus(Var("x"), IntLit(3)))

)

)

)

)

(e)

;;

- : value = IntVal 7

計算結果はIntVal 7になる。Appにて、本来env1を用いて評価していたのは、Funで定義された時点での環境を呼び出すためである。上の例で言えば、もともとは、env1を用いて、let f fun y -> x+y という定義がされた時点での環境、すなわちx=1とされている環境が呼び出されていた。Appにて、これを今回のようにenvを用いて評価すると、Funで定義された時点での環境を参照することなく、現時点(Appが呼び出された時点)での環境、すなわちx=2とされている環境が呼び出され、評価される。

この動作は動的束縛であるといえると考える。

・ここまでに出てきたミニOCamlの構文だけを使って、階乗を求める関数を定義できるか、考察せよ。

＜以下変更部＞

初週に、複数引数を含む再帰関数によって階乗を求める関数を考えたことがあった。このミニOCamlの構文を用いて、LetRec(f, x (Fun(y, (Fun z, ...))), exp)という形で書けば、複数引数を含む再帰関数を定義できるため、定義できると考察できる。

＜ここまで＞

・eval4の関数適用の計算の順序は、OCaml処理系の評価順序とは一致していない。例に示すOcamlプログラムを実行してみよ。

示される2つのプログラムを実行した結果を以下に示す。

# (print\_string "1"; 10) + (print\_string "2"; 20);;

21- : int = 30

# (print\_string "1"; (fun x -> x))(print\_string "2"; 20);;

21- : int = 20

ここから、和の演算および関数適用において、後ろの引数から評価が行われていることが分かる。

・eval4が(e1+e2)や(e1 e2)という式の計算をどういう順序で行うか答えなさい。また、OCamlと一致させるようにeval4を変更しなさい。

eval4内では、基本的にeval4 e1の方がeval4 e2よりも先に出現する。よってe1→e2の順序で計算される。これを逆転させるためにはeval4 e2の方が先に出現させる必要がある。

Plus(e1, e2)においては、関数binopを用いるので、binopを変更した。

let binop f e1 e2 env =

match (eval4 e2 env, eval4 e1 env) with

| (IntVal(n2),IntVal(n1)) -> IntVal(f n1 n2)

| \_ -> failwith "integer value expected"

in

Fun(e1 e2)においては、e2を評価したものをargとして先に宣言することで、e2の方が先に評価されることになる。

| App(e1,e2) ->

begin

let arg = (eval4 e2 env) in

match (eval4 e1 env) with

| FunVal(x,body,env1) ->

eval4 body (ext env1 x arg)

| \_ -> failwith "function value expected"

end

# 課題5-3

・再帰関数の動きを見るため、例に示すテストプログラムを空の環境のもとで評価し、その実行過程を考えよ。

# eval6

(

LetRec("f", "x", Var("x"), IntLit(0))

)

(emptyenv())

;;

- : value = IntVal 0

これはfを定義しているがAppなどにより呼び出されることはない。この場合でもfはRecFunValとして新たな環境env1に登録され、このenv1を用いてIntLit(0)を評価している。

# eval6

(

LetRec("f", "x", Var("x"), App(Var("f"), IntLit(0)))

)

(emptyenv())

;;

これは実際に定義されたfを再帰呼び出ししている。

- : value = IntVal 0

# eval6

(

LetRec("f", "x", If(Eq(Var("x"),IntLit(0)), IntLit(1), Plus(IntLit(2), App(Var("f"), Plus(Var("x"), IntLit(-1))))), App(Var("f"), IntLit(0)))

)

(emptyenv())

;;

- : value = IntVal 1

# eval6

(

LetRec("f", "x", If(Eq(Var("x"),IntLit(0)), IntLit(1), Times(Var("x"), App(Var("f"), Plus(Var("x"), IntLit(-1))))), App(Var("f"), IntLit(3)))

)

(emptyenv())

;;

- : value = IntVal 6

# eval6

(

LetRec("f", "x", If(Eq(Var("x"),IntLit(0)), IntLit(1), Times(Var("x"), App(Var("f"), Plus(Var("x"), IntLit(-1))))), App(Var("f"), IntLit(5)))

)

(emptyenv())

;;

- : value = IntVal 120

2つめ以降は基本的な処理の流れは似ていて、違うところはいわゆるbody部である。body部はAppが呼び出されたときに、評価される。

・上記以外に再帰関数を定義し、eval6を使って実行せよ。

例に示されているfibを定義し、実行した。

# eval6

(

LetRec("fib", "x",

If (Greater(IntLit(3), Var("x")), IntLit(1), Plus(App(Var("fib"), Plus(Var("x"), IntLit(-1))), App(Var("fib"), Plus(Var("x"), IntLit(-2))))),

App(Var("fib"), IntLit(6))

)

)

(emptyenv())

;;

- : value = IntVal 8

課題1での実装とは異なり、定義通りの実装である。正しく計算されている。

もう一つ例として示されているgcdを定義しようと試みたが、2つ引数をもつ再帰関数をこのOcamlでどう実装するかが分からなかった。valueとしてvalueのリストであるListValが定義されているため、これを用いようとしたが、LetRec自体が引数を1つしかもたないよう設計されている。よって2引数をもつgcdは実装できないと考えた。