主専攻実験[S-8] 関数プログラミング

課題3-1,3-2

情報科学類 202113564 三村潤之介

課題3-1

・環境を操作する関数を使って[("x",IntVal 1); ("y",BoolVal true); ("z", IntVal 5)] という環境を作りなさい。

環境に変数を追加するext式は、受け取ったペアを環境の先頭に追加するため、最初に追加したものがリストの最後部となる。よって、この場合、変数”z”から環境に追加していく必要がある。以下のように操作した。

# let env = emptyenv();;

val env : 'a list = []

# let env = ext env "z" (IntVal 5);;

val env : (string \* value) list = [("z", IntVal 5)]

# let env = ext env "y" (BoolVal true);;

val env : (string \* value) list = [("y", BoolVal true); ("z", IntVal 5)]

# let env = ext env "x" (IntVal 1);;

val env : (string \* value) list =

[("x", IntVal 1); ("y", BoolVal true); ("z", IntVal 5)]

最初に環境envを初期化して定義し、以降そのenvについてz→y→xの順に環境に情報を追加していった。最後の行から、適切に環境を作れていることがわかる。

・この環境にy及びwという変数が登録されているかどうか、lookup関数で調べなさい。

以下のように操作した。

# lookup "y" env;;

- : value = BoolVal true

# lookup "w" env;;

Exception: Failure "unbound variable: w".

上部は、yがenv内に存在し、値はBoolVal trueであることを示している。下部は、wがenv内に存在しない変数であることを示している(エラー)。これらは正常な動作である。

・1つの環境の中に同じ変数が2回現れると、lookupのときにどちらの値が使われるか。

例として、 [("z", BoolVal false); ("x", IntVal 1); ("y", BoolVal true); ("z", IntVal 5)] である環境を用意した。これは先程までに作っていた環境の先頭に、すでに登録されているzをもう一度、別の値で登録させたものである。ここでzに対してlookupを行うと、以下のようになった。

# lookup "z" env;;

- : value = BoolVal false

これは、zがBoolVal falseとして登録されていることを示す。つまり、リストの先頭側にあるもののほうが、選ばれている。lookupが先頭からリストを見ていって、該当する変数が見つかったら、それに対応する値を返し、以降探索を行わないため、こうなると予想する。…(ref.1)

・関数extより、「新しく環境に定義された変数と値のペア」が環境の先頭と末尾のどちらに追加されるか考えなさい。また、ある変数に対して異なる値を、続けて環境に登録すると、古い値と新しい値のどちらが生き残るか考えなさい。

extにおいて、「::」はリストの結合を表しており、これにより、与えられたenvの前に与えられたデータペアを結合するため、環境の先頭に追加されることになると考える。

また、同じ変数が同環境内に存在したとき、lookupは、前述した理由(ref.1)によって、先頭側の方を返すので、「新しい値が生き残る」と表現できると考える。

# 課題3-2.

・eval3に種々の例題を与えてテストしなさい。

環境envで、(“x”,3)が登録されているとき、以下のように例を与えた。

# eval3

(If(Eq(Var("x"),Plus(IntLit(1), IntLit(2))), IntLit(1), IntLit(0)))

(env)

;;

- : value = IntVal 1

これは、 もしx = (1+2) であれば 1、そうでなければ0を返す式である。xには予め3が与えられているので、1が返されている。

・Ocamlと同様にeval3でも内側のletが優先されていることを確かめよ。

以下は let x=1 in let x=2 in x を表した式を与えた結果である。

# eval3

(Let("x", IntLit(1), Let("x", IntLit(2), Var("x"))))

(env)

;;

- : value = IntVal 2

2、すなわち内側のlet文が反映された状態でのxの値が返された。このことから、内側のletが優先されていることがわかる。

・Ocamlと同様に、eval3でも外側のletで定義された変数を内側のlet式の中で使うことができることを確かめよ。また、処理の過程で、環境がどのように変化していくか考えなさい。

実行したeval3式とその結果を示す。

# eval3

(Let("x", IntLit(1), Let("y", Plus(Var("x"), IntLit(1)), Plus(Var("x"), Var("y")))))

(env)

;;

- : value = IntVal 3

よって、内側のLet文内のxについて、外側のLetの内容が反映されていることがわかる。値は以下のように計算できることから、正しい。

x + y y:=x+1

=x+x+1 x:=1

=1+1+1

=3

環境は、Let文によって、その内側を評価するタイミングで、変化すると考える。この例で言えば、外側のLetによって、その第三引数であるLet("y", Plus(Var("x"), IntLit(1)), Plus(Var("x"), Var("y")))内でxの値を参照することができるようになっている。

・eval3では「環境から、束縛を除去する」という操作は一切使っていないのに、どうして「解除」の操作が実現できているのか考えなさい。

例(let x=1 in (let x=2 in x+1)+(x\*2))の式をeval3用に書き換えると、

(Let("x", IntLit(1), Plus(Let("x", IntLit(2), Plus(Var("x"), IntLit(1))), Times(Var("x"), IntLit(2)))))

となる。

この例で言えば、「解除」の動作は、1つめのVar(“x”)の出現はIntLit(2)であるのに、2つめの出現はIntLit(1)であることに当たる。このようなLet文の入れ子では、envをスタックを積み上げるように複数保持しながら、適切なところでポップするという、再帰処理の特性がこの「解除」の操作に見えると考える。