主専攻実験[S-8] 関数プログラミング

課題 2-4, 2-5

情報科学類 202113564 三村潤之介

課題 2-4.

・テストプログラムをすべて実行しなさい。

示されているプログラムリストはすべてエラー無く実行された。

例えば、

```
let _ = eval2 (If (Eq(IntLit 2, IntLit 11),
                Times(IntLit 1, IntLit 2),
                Times(IntLit 1, Plus(IntLit 2,IntLit 3))))
```

を見てみる。

実行結果は以下のようになる。

```
# let _ = eval2(If (Eq(IntLit 2, IntLit 11), Times(IntLit 1,
IntLit 2), Times(IntLit 1, Plus(IntLit 2, IntLit 3))));;
- : value = IntVal 5
```

eval2 によって、If は 3 つの引数をとり、1 つめの引数が真ならば 2 つめの引数、偽ならば 3 つめの引数を返すよう定義されている。この例では、1 つめの引数として、Eq 文が評価され、これが偽であるため、3 つめの引数である(1 * (2 + 3))を示す Times 文が評価されたものが返されている。

・エラーが起こる例について、調べなさい。

エラー起こるというのは、ここでは、failwith 文が与えられているケースに match することである。

一番外側の match 文の”unknown expression”について、これに match する式を考える。「Plus(IntLit 1, IntLit 2, **IntLit 3**)」や「**Prus**(IntLit 1, IntLit2)」などが該当するだろうか、と予想したが、これらはそもそも exp で定義されていないため、e が exp であると予期されている eval2 e では「e が exp でないため」という理由でエラーになってしまう。そのため、外側の match でパターンマッチングが発生しながら、例外に match する式は無いと考える。このことは、コンパイル時に

```
53 |   | _ -> failwith "unknown expression"
          ^

Warning 11: this match case is unused.
```

のように警告が出ることからも同様に考えられる。

その他、それぞれの評価部内の”wrong value”例外については、命令や仮引数の数が対応していながら、その仮引数の内部が予期される値の型と整合しない場合に発生すると考える。

・整数値と真理値を Eq で比較する式を eval2 に与えると、何が起きるか。

実行結果は以下のようになる。

```
# eval2(Eq(IntLit 1, BoolLit true));;
Exception: Failure "wrong value".
```

前述したとおり、Eq 文では中の仮引数がどちらも整数値またはどちらも真理値であることが予期されるため、型が整合せずに Eq 文内の”wrong value”例外にパターンマッチする。

・整数同士を比較する式として Greater(e1, e2)を追加したときの処理を追加しなさい。

Greater を実装するためには、まず Greater 文が exp として扱われる必要がある。

仮引数はどちらも exp であるため、exp の定義部に以下のように追加する。

```
type exp =
  | IntLit of int
   …
  | Greater of exp * exp
```

次に eval2 内において、実際の処理を記述する。Greater(e1, e2)において、e1, e2 は整数の値であるので、どちらも整数であるときに評価された仮引数の値を比較して Bool 値を返し、片方または両方が真理値であるときは”wrong value”として例外を返せば良い。

以下のように記述した。

```
  | Greater(e1, e2) ->
    begin
      match (eval2 e1, eval2 e2) with
        | (IntVal(n1), IntVal(n2)) -> BoolVal(n1>n2)
        | _ -> failwith "wrong value"
    end
  | _ -> failwith "unknown expression"
```

例を与えて実行した結果を示す。

```
# eval2(Greater(IntLit 3, IntLit 4));;
- : value = BoolVal false
# eval2(Greater(IntLit 4, IntLit 2));;
- : value = BoolVal true
```

上部は 3>4 の値である False、下部は 4>2 の値である True が Bool 値で返されていることがわかる。

課題 2-5

・binop という関数は、eval2b の外からは使えないことを確かめよ。

binop が外部から使えるならば、コンソールモード(?)で例えば以下のような式

```
binop((+) IntVal 3, IntVal 2);;
```

によって、IntVal(5)が返ってくるはずである。しかし、実際には、

```
# binop((+) IntVal 3, IntVal 2);;
Error: Unbound value binop
```

となり、binop が開放されている変数としてエラーが出てしまう。これは eval2b の外では binop が関数として定義されていないことを示す。

・「外から使える」ことがメリットではなく、「外から使えないこと」がメリットとして主張されている理由を考えよ。

実験資料 2-5.より、以下の記述を引用する。

このようにすることにより、eval2b の外から binop という関数は使えない。 従って、他に binop という名前の関数があっても名前が衝突しない、というメリットがある。

おそらくこれがその大きなメリットであると考える。

「外から使えないこと」自体のメリットではないが、このように記述することのメリットとしては、カプセル化との類似性があることから類推して、「使用者が内部構造を理解しなくとも利用可能である」こともメリットに挙げられるだろう。この例で言えば、使用者は binop の厳密な定義を理解しなくとも、二項演算子及びオペランドを与えることさえ知っていれば十分に利用できる。

・eval2 に対して、eval2b への修正をほどこし、 eval2b を完成させよ。

eval2b の大きな特徴としては binop によって二項演算子を渡すだけで二項演算ができるようになっていることである。Greater, Eq, If のうち、二項演算であるのは Greater および Eq であるため、それらについて同様な処理を行うことを考えた。

最終的に、Greater に対して binop_bool という関数を定義することにした。binop f e1 e2 自体は(int -> int -> int) -> exp -> exp -> value であるため、二項演算子にあたる f は int -> int -> int として型が推定される。よって、int -> int -> bool を望む「>」と同じ関数で扱うことができないと考えたためである。

binop_bool f e1 e2 は(int -> int -> bool) -> exp -> exp -> value であるよう定義し、「>」や「<」などの bool を出力とする二項演算子はこれを用いて評価するようにした。これによって Greater の他、(a < b)を評価する Less(a, b)も実装できるようになった。

Eq については、「=」が引数によって(int -> int -> bool)と(bool -> bool -> bool)のど

ちらであるかが変わってくるため、いわゆるカプセル化が難しいと考え、eval2 の実装をそのまま流用した。

If については、前述したとおり二項演算子の類いではないため、eval2 の実装をそのまま流用した。

・binop の定義を eval2b の中から外に出し、その型がどうなるかを調べよ。

binop をそのまま eval2b 外側にコピーした。以下のように出力された。

```
# binop;;
- : (int -> int -> int) -> exp -> exp -> value = <fun>
```

これは、int を返す二項演算子、及び 2 つの exp から value を返す式であることを示している。ここで、binop に(+)や(>)などの演算子を渡してみる。

```
# binop (+);;
- : exp -> exp -> value = <fun>
# binop (>);;
Error: This expression has type int -> int -> bool
       but an expression was expected of type int -> int -> int
       Type bool is not compatible with type int
```

上部は、先程の(int -> int -> int)部をとった形に等しくなっている。一方で、下部はエラーが出力されている。これは(int -> int -> bool)である「>」が整合していないことを示している。