主専攻実験[S-8] 関数プログラミング

課題4-1

情報科学類 202113564 三村潤之介

課題4-1

・parse関数とこれまでに作成した eval関数を組み合わせなさい。

parse関数とeval関数を組み合わせたrunは、str型のデータを受け取って、exp型のデータを返す関数である。parse関数によってstrから生成・出力されたexpを前回までに作成したeval関数によって評価すれば良い。

すでにMakefileなどでは、コンパイルするファイルとして、インタープリタ用のeval.mlファイルが用意されているため、ここにeval関数を書き込んだ。このeval関数は、1つのexpのみを引数として受け取るものが望ましいため、課題3で作成したような、環境も引数とするものではなく、課題2で作成したものを採用した。

ただし、eval関数だけをコピーすると、eval関数内のexp表現が読み込めないため、エラーとなってしまう。ここで、課題で作成したexp定義やvalue定義を持ってくるのではなく、syntax.mlの定義を使用させる。こうすることで、main.ml内で共通したexpデータをSyntax.expとして扱えるようになる。

以下に実行例を示す。

# Main.run "1+2\*3";;

- : Syntax.value = Syntax.IntVal 7

run関数によって、”1+2\*3”という文字列が評価され、7と出力されていることが分かる。

・syntax.ml に記述されたexp型の式を拡張するためには、 lexer.mll と parser.mly をどう変更すればよいか。例として、2つの数が等しくないことを意味する演算子「<>」を追加してみよ。

exp表現としては、２つのexpをもつ「Neq」とし、「<>」がトークン「NEQUAL」として表現されるとした。

まず、syntax.mlにて、expの定義として、Neqを以下のように追加した。

type exp =

…

| Neq of exp \* exp (\* e <> e \*)

次に、lexer.mllにトークンを追加した。

rule token = parse

…

| "<>" { NEQUAL }

次に、parser.mlyに使うトークンの登録、及びexp表現への変換を記述した。

…

%token NEQUAL // "<>"

…

exp:

…

// e1 != e2

| exp NEQUAL exp

{ Neq ($1, $3)}

以上の記述をした上で、コンパイルした。以下に実行例を示す。

# Main.parse "1<>2";;

- : Syntax.exp = Syntax.Neq (Syntax.IntLit 1, Syntax.IntLit 2)

parseによって、構文解析が行われ、「1と2が異なる」ということを表すexp表現が出力されていることが分かる。ただし、eval関数に「<>」に関する記述をしていないため、このexp文を評価することはできない。