Java コンパイラの算出に向けて

鈴木 凌 202003515

1　概要

本稿では，関数型プログラミング言語である Haskell を用いて、コンパイラを算出する．この分野の先行研究 [1] として算術式やラムダ式などに対するコンパイラの実装がある．そこで本研究では，プログラムを Java 仮想マシン (以降 JVM と表記する) で扱うコードに翻訳するコンパイル関数 comp，翻訳したコードを実行する関数 exec，プログラムを受け取り中間言語に変換せず直接実行する評価関数 eval の実装を行う．実装にあたり，Haskell を用いる理由は，Haskell の豊富な型システムにより，型をほとんど明記せずにプログラムの実行前に型エラーを検出してくれることに加え，Haskell では純粋な関数を扱うため，等式推論を利用しプログラムの実行，導出，プログラムの論証に有用であるからである．なお，ここで定義する関数 comp，exec，eval は，次の関係を満たすことが望ましい．

```
exec (comp Program) = eval Program
```

2　文法

今回扱う Java のサブセットは，以下である．

```
Statement = Assign Type String Expr
          | Assignf String Expr
Expr = Val Int | Var Type Key
     | Field String Key | Add Expr Expr
     | New Classname
     | Call Methodname [Expr]
```

Statement は，代入文である．Expr は，Val が数値リテラル，Var が変数値，Field がフィールド値，Add が加算，New がオブジェクト生成，Call がメソッド呼び出しである．以下にプログラムの一例を示す．

```
Class Main []
  [Method "main"  []
          [("sub", ClsTy "Sub")...]
          [(Assign (ClassTy "Sub")
                   (New "Sub")),
          :
   Method "Main" ... ]
   Class Sub ["a" IntTy] ...
```

プログラムは，1 個以上のクラスからなり，クラスはフィールド，メソッド，返り値等の情報を持つ．ただし，プログラムには ‘‘Main’’ という名前のクラス，そのクラスには ‘‘main’’ という名前のメソッドがあるものとする．

3　コンパイル関数：comp

まず，Java コンパイラがプログラムの情報を記憶しておくメモリについて説明する [2]．JVM が，クラスファイルを実行する際，プログラムはメモリに展開され，クラスの情報はメソッドエリア，クラスのインスタンスはヒープに置かれる．他にも，スレッドスタックというメモリ領域がスレッド毎に割り当てられる．スレッドスタックとは，フレーム（メソッド用の作業用メモリ）が積み上げられたものである．以下が，Haskell の型で表現したメモリの構造である．

```
Mem        = (Tstack, Heap, Methodarea)
Tstack     = [Frame]
Heap       = [(Address, Object)]
Methodarea = [Classinfo]
```

先述したように，JVM はクラスファイルを読み込み実行するが，より正確に述べるならば，そのクラスファイル中のメソッドのバイトコード Code を実行することである．Code を表す構造を以下に抜粋して記述する．

```
Code = ILOAD Key Code | IRETURN
     | INVOKEVIRTUAL Methodname Code
     | GETFIELD String Code
     | NEW Classname Code | ASTORE Key ...
```

ILOAD，GETFIELD はスタックに値を積む，ASTORE はスタックから値をリストに積む．INVOKEVIRTUAL，INVOKESPECIAL はメソッドの呼び出し，NEW は新しいオブジェクトを生成，IRETURN は実行中のメソッド呼び出しの終了を表すものである．ここで，継続がある Code は，再帰的に引数として Code を持ち，JVM では，扱うデータの型によって別々の Code を用意されている [2].

次に，関数 comp について説明する．この関数はソースプログラムを受け取り，Code へコンパイルする関数である．以下が関数 comp の定義である．

```
comps :: Statement -> Code
comps (Assign t x e)
 | t == IntTy = comp e (ISTORE x)
 | otherwise  = comp e (ASTORE x)
comps (Assignf f e) = comp e (PUTFIELD f)
comp     :: Expr -> Code -> Code
comp (New x) c =
    NEW x (DUP (INVOKESPECIAL c))
comp (Call k mn es) c =
  ALOAD k
  (compList es (INVOKEVIRTUAL mn c))
            :
compList :: [Expr] -> Code -> Code
```

関数 comps は，Statement を受け取り Code を返す関数である．例えば，Assign は，代入される変数の型によって場合分けをして，代入する式：e をコンパイルし，その後，継続の Code を実行する．関数 comp は，Expr と継続の Code を受け取り Code を返す関数である．Expr それぞれに対して，関数 comp を実装しているが，本稿では抜粋している．一つ目の New は，NEW で新しいオブジェクトを生成し，そのオブジェクトに対して，INVOKESPECIAL でコンストラクタを呼び出す．二つ目の Call は，呼び出すメソッドのオブジェクト参照をスタックに積み，メソッドの引数を compList で評価し，INVOKEVIRTUAL でメソッドを呼び出す．関数 compList は，Expr のリストと継続の Code を受け取り，リストの要素に関数 comp を実行するものである．

4 実行関数：exec

実行関数 exec は Code を実行する JVM を表し，Code とメモリの情報を受け取り，メモリの情報を返す関数である．以下が関数 exec の定義である．

```
execstart :: Mem -> Mem
exec      :: Code -> Mem -> Mem
exec (GETFIELD f c)
     ((p, Obj a : s,l) : ts, h, ma)
    = exec c ((p, f : s, l) : ts, h, ma)
exec (INVOKEVIRTUAL m c) mem =
     exec c
     ((p+1, [fst (execList [Code] mem)],
     ("this", Obj a) : para) : (p, s, l)
     : ts,
     snd3 (snd (execList [Code] mem), ma)
               :
execList :: [Code] -> Mem -> (Value, Mem)
```

関数 execstart は，メソッドエリアの中から ‘‘Main’’ という名前のクラスの ‘‘main’’ という名前のメソッドを実行する，プログラムの始まりを決める関数である．関数 exec は，Code それぞれに対して定義があるが本稿では抜粋する．一つ目は，GETFIELD である．これは，スタックの先頭にあるオブジェクト参照の値から，ヒープのオブジェクトを参照する．その後，引数である String 値と一致する名前のフィールド値：f をスタックに積む．オブジェクトは連想リストで実装している．二つ目は，INVOKEVIRTUAL である．この Code ではまず，スレッドスタックには新しく呼び出されたメソッド用のフレームが生成され，メソッドを実行した結果と，メソッド引数とローカル変数と自分自身への参照のリストを表現した変数：para が積まれる．ヒープは，メソッド実行後のヒープの状態に変更される．関数 execList は，Code のリストとメモリを受け取り値とメモリの組を返す関数である．

5 まとめと課題

評価関数 eval は，関数 comp と関数 exec の実行を一度に直接行うものだが，関数 eval もプログラムを受け取りメモリを返す関数となるので，クラスの情報を格納するメソッドエリア，実行時に生成されるオブジェクトの情報などを格納するヒープのような構造が必要であり，メモリの構造は似たものとなる．しかし，関数 eval は一度 Code に翻訳する手順を踏まないので，メモリに格納されるデータは Code ではなく，Statement を格納することになる．最後に，今後の課題として，型に真偽値の追加，Expr に親クラスの継承の追加や，[1] で行われている算出したコンパイラに対して，等式の正しさを等式推論，数学的帰納法の技術を用いた証明などがある．

参考文献

[1] Patric Bahr, Graham Hutton, *Calculating correct compilers*, Jounal of Funtional Programming, 25, 2015

[2] Jon Meyer, Troy Downing, *Java Virtual Marchine*, 1997