Fortran トランスレータを生成するコンパイラ・コンパイラの試作

岩　下　英　俊†

1. はじめに

Fortran は HPC 分野で広く使われているプログラミング言語であり，その仕様は急速な拡張・発展をつづけている．最新の仕様である Fortran2008 は，926 個のルールから成る膨大なものである．

このような Fortran 仕様の規模に対応するため，新しい **Fortran パーサ**を提案する．そして，パーサが生成した構文木を簡単にアクセスする手段としての**プログラマブルなインタフェース**を提案する．

2. Fortran パーサ

Fortran2008 の文法書からコピーした構文ルールに対応するパーサを作成した．出力はプログラムの構造をそのまま反映した構文木となる．

2.1 Fortran の規模と複雑さへの対応

Fortran の規模に対応するには，できるだけ自動化する必要がある．しかし Fortran の文法規則では，予約語を持たないためキーワードと名前が文脈で区別される，空白は無視されてトークンの区切りの意味を持たない，などの扱い難さがあり，yacc や ANTLR などのパーサ生成器を用いても簡単には対応できない．これに対し，我々は以下のような対応を行った．

2.1.1 BNF ルールの自動生成

Fortran 仕様書の構文規則には，`[ ] ...` などの記号や，*xxx*`-list`（*xxx* をコンマで区切った並び）などのメタルールが多用されている．これを BNF 記法に変換して yacc プログラムを生成するプログラムを作成した．yacc のアクション部についても 95%のルールについては自動生成している．

2.1.2 文の解析

Fortran の字句解析は文脈に依存する．lex による字句解析の結果に，後処理を加えて修正する方法を採った．改行文字とセミコロンを境にトークン列を文に切り分けた後，各文について以下の処理を行う．

† (株) 富士通研究所 IT システム研究所 システムミドルウェア研究部
IT Systems Laboratories, Fujitsu Laboratories Limited

```
!== subroutine ======
      subroutine co_loop_test  &
         (a, b, c, n, k1, k2)
      real*8 :: a(n)[*], b(n), c(n)
      integer :: k1, k2, n

!  THIS LOOP SHOULD BE OPTIMIZED
      do 300 i = 2, n-1
         b(i) = a(i)[k1] * 10
         c(i) = a(i)[k2] * 100
 300     continue

      return
      end subroutine
!!! end of file !!!
```

図 1 サンプルプログラム 1
Fig. 1 Sample program 1

- トークンを先読みして文のキーワードを見つけ，文の種別を判定する（代入文と DO 文の識別など）．
- 文の種別をヒントに字句解析の修正を行う（キーワードと名前を識別するなど）．
- ラベル付き DO 構文（古い書式）について，構文の終りの位置に疑似トークンを挿入する．

2.1.3 構文ルールへのパッチ

conflict 回避の最後の手段として，構文ルールの手修正を行った．将来の手間を最小にするため，元の構文規則に対する変更点（削除，修正，追加等）だけを記述するパッチファイルとした．

2.2 生成される構文木

パーサは，**図 1** のプログラムを，**図 2** に示すような構文木（python のリスト）に変換する．

構文木の要素となるデータ構造は以下の 4 種類であり，それぞれ，Block, Stmt, Expr, Token クラスのインスタンスである．

- block は構文に対応し，block と stmt をもつ．
- stmt は文に対応し，expr と token をもつ（図 2 では省略されている）．
- expr は式に対応し，expr と token をもつ．
- token は終端記号であるキーワード，名前，定数表現，演算子，括弧などに対応する．

これらのクラスは，Fortran 文法の言語要素の包含

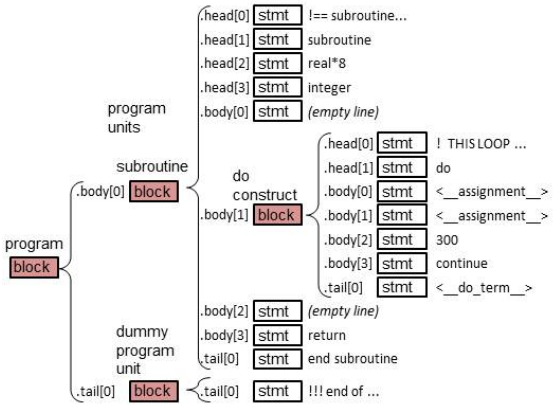

図 2 ブロックの内部表現 (サンプルプログラム 1)
Fig. 2 Internal representation (IR) of a block (for sample program 1)

関係に沿ったサブクラスをもっている．例えば，Stmt のサブクラスの一つは ExecutableStmt（実行文）であり，そのサブクラスの一つは IfStmt（IF 文）である．サブクラスを定義する python プログラムは，Fortran の構文ルールの記述から自動生成している．

3. プログラマブルなインタフェース

3.1 利用者インタフェース

利用者インタフェースは python ベースの Domain Specific Language(DSL) として提供される．インタプリタモードで使用するなら，環境変数 `PYTHONPATH` に本システムの command モジュールへのパスを追加し，python を立ち上げて最初に

```
from command import *
```

とタイプすれば，以下のような関数やメソッドが使用できる．

`accept(`*filename*`)` ファイルからプログラムを読み込み，構文解析を行い，内部表現（構文木）を返す．

`str(`*obj*`)` / `print` *obj* / *obj*`.write(`*filename*`)`
obj を Fortran プログラムのテキストに変換しその文字列を返す/ディスプレイに表示する/ファイルに書き出す（コード生成）．

obj`.deepcopy()` *obj* の深いコピーを作って返す．浅いコピーには普通の代入文を使う．

block`.allstmts([`*only*`])` / *block*`.allblocks([`*only*`])`
block に含まれる全ての文/ブロックを返すジェネレータ．`only` 引数で絞り込みができる．例えば，

```
for x in b.allblocks(only=DoConstruct)
    ...
```

と書くと，ブロック `b` 内のすべての DO 構文を変数 `x` に順に代入しながら反復するループとなる．

```
1
2  from command import *
3
4  program = accept('sample/himeno.f90')
5  jacobi = program[10]
6
7  loop0 = jacobi.body[1]
8  do_k  = loop0.body[1]
9  do_j  = do_k.body[0]
10 do_i  = do_j.body[0]
11 core  = do_i.body[:]
12
13 do_i.body = []
14 do_j.body = []
15 do_k.body = []
16
17 do_ijk = [do_i, do_j, do_k]
18
19 for x,y,z in [(0,1,2), (0,2,1), (1,0,2),
                 (1,2,0), (2,0,1), (2,1,0)]:
20     newLoops = [do_ijk[x], do_ijk[y], do_ijk[z]]
21     loop0.body[1] = newLoops[0]
22     newLoops[0].body = newLoops[1]
23     newLoops[1].body = newLoops[2]
24     newLoops[2].body = core
25
26     program.write("himeno{0}{1}{2}.f90".format(x,y,z))
```

図 3 構文木を操作するプログラムの例
Fig. 3 An example of program handling IR

構文木の部分木の削除，挿入，修正などは，python がもつリストの基本操作を使う．

3.2 プログラムの例

Himeno ベンチマークプログラムを入力とし，Jacobi ルーチンの 3 重ループについて 6 通りの loop interchange を行ってファイルに出力するプログラムの例を，**図 3** に示す．生成された 6 つのプログラムはどれも正しく実行することができ，ループの順序は元の (k,j,i) が最も高速であることを確認した．

ここではその位置（何番目のブロックか）を直接指定することで変換対象のループやサブルーチンを特定したが，条件にマッチするものをプログラム中から検索することもできる．他に，プログラムの解析や式の変形に使える関数やメソッドを今後増やしていく．

4. まとめと今後

Fortran の最新仕様への対応は大きな課題である．本研究では，Fortran の文法書から抜粋した構文ルールから BNF ルールを自動生成し，さらに字句解析に後処理を加えるなどの工夫により，Fortran2008 仕様に対応するパーサを作成した．引き続き，Fortran2008 仕様のフルサポートを目指す．

パーサが生成した構文木へのインタフェースは，python をベースにしたプログラム可能なもの（DSL）とした．今後は，これを具体的なトランスレータの作成に使いながら，機能を増やしていきたい．