3D-10

ＵＮＩＸ上でのソース移植システム

富田智良，湯川　謙，渡辺秀次，小川義高

三菱電機東部コンピュータシステム（株）

１．はじめに

Ｆｏｒｔｒａｎの言語仕様は、ＪＩＳで規格化され各種Ｆｏｒｔｒａｎプログラム間の互換性が考慮されている。しかし、各種Ｆｏｒｔｒａｎで、それぞれ独自の機能拡張を有するため、別のＦｏｒｔｒａｎシステム間のプログラム移植時に問題が生じる。今回この問題に対する一つの例として、当社の異る機種間でのプログラム移植に際し、Ｆｏｒｔｒａｎプログラム変換ツール（ＡＦＣＯＮＶ）を開発した。本論文では、この変換ツールに関する紹介と、ＵＮＩＸを用いたツールの開発手法について報告する。

２．変換ツールの概要

変換ツールは移植元のＦｏｒｔｒａｎソース・プログラムを移植先で動作可能な様にソース・プログラムの文法を変換する機能を持つ。図２－１に変換後の出力ソース・プログラムを示す。

```
      DO 10 I=1,10
C         INT1(I)=INT2(I)=I-1                                          ―――― 拡張代入文
          INT2(I)=I-1                                                  ********
          INT1(I)=INT2(I)                                              ********
   10 CONTINUE
      CALL INIT(L1,L2,RTC)
C     IF(L1.EQR.L2)IF(RTC.EQ.0) GOTO 900                               ―――― 論理ＩＦの連続と
      IF(L1.NEQV.L2) THEN                                              ******** 論理演算子. EOR.
      IF(RTC.EQ.0) GOTO 900                                            ********
      ENDIF                                                            ********
      DO 20 I=1,10
C         CALL CHK(INT1(I),INT2(2)) ; IF(INT1(I).EQ.INT2(I)) GOTO 100  ―――― 複合文
          CALL CHK(INT1(I),INT2(2))                                    ********
          IF(INT1(I).EQ.INT2(I)) GOTO 100                              ********
   20 CONTINUE
```

図2－1　変換ツールの出力例

この変換ツールの機能構成は、図２－２に示すように５つにわかれている。ここで構文解析部は、ＵＮＩＸ上で動作するコンパイラ・コンパイラｙａｃｃにより記述している。解析木作成部はｙａｃｃのアクションとして呼び出され、入力したＦｏｒｔｒａｎプログラムの解析木を作成する。

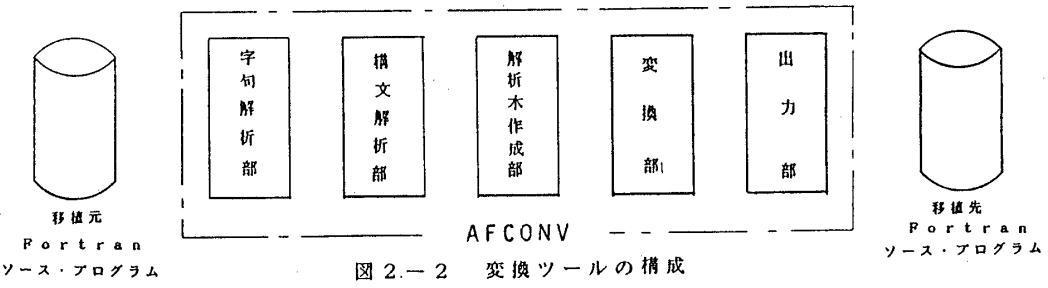

図２－２　変換ツールの構成

FORTRAN SOURCE PROGRAM CONVERSION SYSTEM ON UNIX
T.TOMITA, K.YUKAWA, S.WATANABE, Y.OGAWA
MITSUBISHI ELECTRIC TOHBU COMPUTER SYSTEM CORP.

３．ｙａｃｃを使った解析木の作成

ｙａｃｃを使って解析木を作成する場合、ｙａｃｃのアクション部に２つの処理を記述する必要がある。それは「ノードをつくる処理」と「枝をつなぐ処理」である。
この２つの処理をｙａｃｃのアクション部へ記述することにより、解析木は簡単に作成される。図３－１は構文解析部の一部で、関数ＭＫ＿ｎｏｄｅ（　），ＭＫ＿ｂｒａｎｃｈ（　）はそれぞれ「ノードをつくる処理」，「枝をつなぐ処理」に対応する。アクション部の記述はルール部の右辺のシンボルの数に依存する。そこで、図３－３の様にアクション部をマクロ化し、保守性、拡張性を向上した。マクロ展開は、ＵＮＩＸツールａｗｋ，ｔｒ等を組み合わせて実現している。

```
multiexec: exec                    {                    ①
                                   $$=Mk_node(MULTIEXEC);
                                   Mk_branch($$,$1);
                                   }
         | multiexec _ssemicol exec {                    ②
                                   $$=Mk_node(MULTIEXEC);
                                   Mk_branch($$,$1);
                                   Mk_branch($$,$2);
                                   Mk_branch($$,$3);
                                   }
```

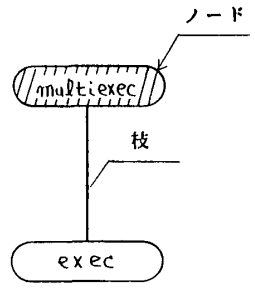

図３－１ｙａｃｃを使った解析木の作成

```
multiexec: exec                    {
                                   NON_TERM
                                   }
         | multiexec _ssemicol exec {
                                   NON_TERM
                                   }
```

図３－３ マクロによるｙａｃｃソース

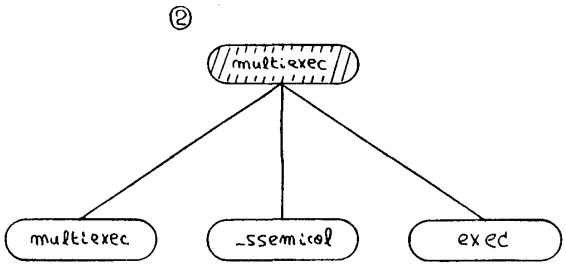

図３－２解析木

４．まとめ

ｙａｃｃを使う場合、複雑なロジックを記述する必要がないため、構文解析においては非常に強力である。ＵＮＩＸ上で動くコンパイラ・コンパイラとして、もう１つｌｅｘがある。ｌｅｘは字句を解析するプログラムを出力するツールであるが、この変換ツールでは使用していない。それはｌｅｘが、Ｆｏｒｔｒａｎの字句を解析することができないためである。Ｆｏｒｔｒａｎの字句は予約語がなく、たとえば、ＩＦ（・・・と入力された場合、ＩＦ文なのか、ＩＦという配列なのか、又は関数なのかｌｅｘのレベルでは判断できないためである。予約語のない文法はｌｅｘでは処理できないのである。

解析木の作成では、前章で記したようにｙａｃｃのアクション部をマクロ化することができた。マクロ化したことによりルール部を修正してもアクション部は修正する必要がなくなり拡張性、保守性が向上している。ＵＮＩＸはこのマクロ化という一寸したアイディアがすぐに実現できる手軽で強力なＯＳである。