実用性を重視したＰＬ／Ｍ－Ｃトランスレータの実現

4Y-4

小椋　覚　，　吉田正和　，　小山孝司

ファームウェアシステム株式会社

１．はじめに

ＰＬ／Ｍ８６ソースをＣ言語に変換するトランスレータについては既に事例報告[1]があり、製品[2]も存在する。

今回、コンパイラと同等の解析処理を行い変換効率を高めると共に、注釈やマクロ定義、インクルード・ファイルなどのプログラムの形態を極力保存する、実用性を重視したＰＬ／Ｍ－Ｃトランスレータを開発した。これはΣＷＳ（ΣＯＳ）上で動作するΣツールで、情報処理振興事業協会シグマシステム開発本部と弊社で開発したものである。

以下では、このトランスレータの概要について報告する。

２．ＰＬ／ＭとＣ言語の差異

ＰＬ／ＭとＣ言語の差異は次のとおり：

（１）ブロック構造によるもの
- ・関数（手続き）の入れ子
- ・入れ子になった関数のローカル変数
- ・関数外部へのＧＯＴＯ

（２）ハードウェア依存の仕様によるもの
- ・Ｈ／Ｗに密着したメモリ割付の機能
  （例：ＡＴ、ＤＡＴＡ、ＩＮＩＴＩＡＬなど）
- ・データの連続性の保証
- ・Ｈ／Ｗを意識したメモリ参照機能
  （例：ＳＥＬＥＣＴＯＲ、ＭＥＭＯＲＹなど）
- ・Ｈ／Ｗを直接操作する組込み機能

（３）演算子の順位が若干異なる

（４）表現式が持つ値が異なる
- ・算術演算の暗黙の型変換の規則
- ・真偽の値

（１）の関数の入れ子は、独立した関数として変換する。入れ子になった関数のローカル変数は、関数外の静的変数に変換する。関数外部へのＧＯＴＯ及び（２）は、変換が困難なので、警告メッセージを出力しその部分を注釈にする。（３）は、括弧を用いて明示的に演算の順位を変える。（４）は、厳密に変換しようとすると、生成されるＣプログラムは読みづらく冗長になるので、オプションにより、どのように変換するかを指定するようにする。

３．トランスレータの特徴

ＰＬ／Ｍ－Ｃトランスレータの特徴は次のとおり：
- ・コンパイラと同等の解析処理
- ・注釈、数値定数の基数などの形態情報を保存
- ・インクルードファイルの復元変換
- ・マクロ定義の復元変換
- ・警告メッセージの出力とマーカーの挿入
- ・変換方法のオプション指定

４．変換アルゴリズム

トランスレータは、次の３つのパスから構成されており、ドライバにより各パスの制御が行われる。

（１）プリプロセッサパス

ＰＬ／Ｍのソースファイルよりインクルード文を展開する。コンパイラ疑似命令は、変換出来るもの（条件付きコンパイル）であれば、Ｃ言語プリプロセッサの疑似命令に変換し、変換できないもの（メモリモデルの指定等）であれば、注釈にする。

（２）構文解析パス

プリプロセッサパスにより展開されたＰＬ／Ｍソースを構文解析して構文木を作る。ＬＩＴＥＲＡＬＬＹ、注釈などの形態情報は構文木に付加する。変換に必要なので、宣言、式はコンパイラと同等の意味解析をする。また、関数引数の型チェック、変数の前方参照処理も行う必要がある。

（３）変換出力パス

構文解析パスより得られた構文木より、Ｃ言語ソースを出力する。ＰＬ／Ｍソースの注釈、インクルード・ファイル、マクロ定義（ＬＩＴＥＲＡＬＬＹ宣言）を保存復元し、トランスレータが識別子を自動生成するようなこともなるべく避ける。変換不可能な場合や結果が疑わしい場合には、警告メッセージを出力し、後処理のためにＣソースにマーカーを挿入する。まれにしかない特殊ケースの対応はオプションにし、必要な時にのみ指定するようにする。変換の既定値は、最も一般的で読みやすい出力になる。

An Implementation of Practical PL/M-C Translator

Satoru OGURA, Masakazu YOSHIDA, Takashi KOYAMA

FIRMWARE SYSTEMS, Inc.

5．インプリメントと変換例

トランスレータのインプリメントにｙａｃｃを使用する事例が多いが、形態情報の保持等の例外的な処理を柔軟に行うために、使用しなかった。また、主記憶上にソースプログラムの全てを構文木として展開するため、大きなプログラムを変換する場合には、大量にメモリを必要とする。今回は、ΣＷＳ上で使用することを前提にしているので、これは特に問題とならない。

以下に変換例を示す：

ＰＬ／Ｍ－８６（入力）

```
$LARGE
$INCLUDE(:F2:SAMPLE.EXT)
EXAMPLE:DO;
      DECLARE TRUE LITERALLY '0FFH';
      DECLATER FALSE LITERALLY '0';
      DECLARE (A,B) BYTE;
      PRINT$DIAG: PROCEDURE(X,Y) EXTERNAL
            DECLARE (X,Y) BYTE;
      END PRINT$DIAG;
      MUL: PROCEDURE (A,B) BYTE PUBLIC;
            DECLARE (A,B) BYTE;
            RETURN A * B;
      END MUL;
$IF   DEBUG = 1
      CALL PRINT$DIAG(TRUE, FALSE);
$ENDIF
      A = MUL(A, B);
END EXAMPLE;
```

Ｃ言語（出力）

```
#include        <plm.h>
/*### not supported memory model %%%*/
/*### $LARGE %%%*/
#include        "hexample.ext"
extern VOID printdiag();
BYTE    mul();

#define TRUE    0xff
#define FALSE   0

main()
{
      static BYTE a;
      static BYTE b;
#if   DEBUG == 1
      printdiag(TRUE, FALSE);
#endif
      a = mul(a, b);
} /* example */

BYTE
mul(a, b)
  BYTE a;
  BYTE b;
{
      return(a * b);
} /* mul */
```

6．おわりに

トランスレータを使用すれば、ＰＬ／Ｍ８６プログラムの８割から９割は自動的にＣ言語へ変換することができる。変換できない残りの１割ないし２割は、どうしても人手で後処理をしなければならない。変換作業の負担を軽減するには、変換効率を高めて後処理の量を減らすことと、後処理が楽になるように工夫をすることの２つの道がある。変換効率については自ら限界がある。
本ツールでは、形態情報を保存し読みやすい変換を行うこと（＝美観上の修正のための後処理を減らす）や、後処理が必要な箇所にマーカーを挿入するなど、実用性を重視したＰＬ／Ｍ－Ｃトランスレータの実現を行い、十分な結果が得られた。
今回の実現では、複数のファイルから参照されているインクルード・ファイル及び複雑なマクロの復元が完全には対応できていない。これを今後の課題としたい。

参考文献：

[1] 木下　他　"ＰＬ／ＭｔｏＣトランスレータの実現と課題"　情報処理学会第３４回全国大会予稿集　ｐ８１５～８１６
[2] 松浦　他　"ＰＬ／Ｍトランスレータ"　Ｃｏｍｐｕｔｅｒ　Ｔｏｄａｙ　１９８７／４別冊　ｐｐ８７～９４