51-1

システム記述言語Modula-2の移植性の高い処理系の作成

隅田 英一郎・武市 正人
電気通信大学 計算機科学科

1 言語と処理系の概要

Modula-2は可視性を制御するmodule構造を導入し、データ抽象化、情報隠蔽、局所性などのプログラミング方法論上の原理を支援し、特に機械依存部分は低水準のmoduleに封じ込めた。さらに、non-localなmoduleは、定義部と実現部に分けられ、他のmoduleを参照しているmoduleは、必要なmoduleの定義部を利用して、完全な検査のもとに分割翻訳される。これは、大規模なプログラムの開発・保守に役立つ機能である(詳細は参考文献にゆずる)。

本稿の処理系はBCPLで記述した。構文解析部(YACCで生成したもの)は、構文木(的分木)を出力する。意味解析部は構文木を変換する。一度BCPLの中間コードのOCODEに落とし、さらにNATIVEなコードに変換する。故に、OCODEの変換系さえ作れば容易に移植できる。

複雑な可視性の解析と分割翻訳の実現法を以下に詳述する。

2 スコープ・アナリシス

伝統的なALGOL-60やPASCALでは外側のscope(blockやprocedure)で宣言された名前とそれが指す対象は、内側のscopeで(内側で同じ綴りの名前が宣言されない限り自動的に)利用可能である。ところが、最近の言語Euclid, Mesa, Modula-2などは、内側のscopeでの利用を制限し、programmerが明示(Modula-2の用語では、import)した時だけ許すscopeを導入した。前者をopen scope、後者をclosed scopeと呼ぶ。又、closed scopeでは、データ抽象化や情報隠蔽などを支援するために、あるscopeにlocalな対象をprogrammerが明示(export)した時だけ利用可能となる(図-1参照)。

図-1 open or closed scopeの例

```
MODULE m1;
  IMPORT a;
  EXPORT b,c;
  VAR d:INTEGER;
  MODULE m2;
    IMPORT d;
    EXPORT e;
    VAR e,f: BOOLEAN;
    { c,d,e,f are accessible }
  END m2;
  PROCEDURE b;
    VAR f:REAL;
    { a,b,c,d,e,f are accessible }
  END b;
  { a,b,c,d,e are accessible }
END m1;
```

本稿のアルゴリズムは、名前表を作らずに、構文木の上で等価なものを実現する。ハッシュ法によって綴りの同じ名前は同じノードを共有している。各ノードには一語スペースがあって、名前の場合、現在(import, export, declarationのノードの場合;外側で)有効なdeclaration等を指す。又、各ノードはscopeのlevelを持っている(=重定義のチェックに使う)。木を二度なめる。moduleの例を示す。

フェーズ1

i) import, declaration, 子供のmoduleのexportをなめる。オープン。(図-a)
ii) 子供のmoduleに対して、再帰呼出し
iii) 自分のexportと対応するdeclarationの

統合（図－c）。
(v) i)の逆。クローズ。

フェーズ2

i) import（図－b），declaration（図－a），子供のmoduleのexport（図－d）を集める。
ii) declarationの右辺のapplied occurrenceの処理と子供のmoduleに対し再帰呼出し
iii) statementの中のapplied occurrenceの処理
iv) iの逆

3　分割翻訳

Moduleの定義部からexportされた文章は，module名を明示してimportすれば他のmoduleで利用できる。類似した分割翻訳の方式がMesaやAdaにある。

例えば，図－2のStorageの実現部を翻訳する時に，SYSTEMの定義部からimportするには，前者を中断して，SYSTEMの定義部ソースを用いてscope analyzerを使って parseすれば良い。

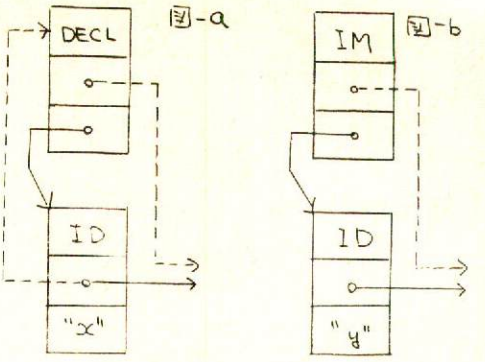
関係する構文規則を次に示す。

definition_module ::= DEFINITION
　MODULE IDENT ";" {import}
　[export] {definition} END IDENT "."
import ::= FROM IDENT IMPORT ident_list

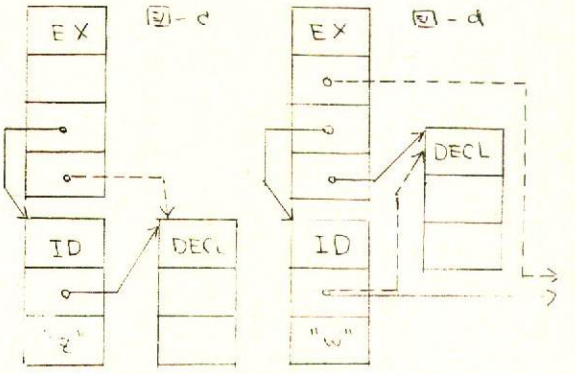
この構文規則を次の様に拡張すれば，上記の目的を容易に達成できる。

import ::= FROM module_ident definition_module IMPORT ident_list
module_ident ::= IDENT

module_identを認識した時の処理系のactionは現在の入力ファイルをスタックし，IDENTによって指定されたファイルをopenする事。ファイルを読み切った時scannerが，そのファイルをcloseし，以後スタックされていたファイルの続きを読む。



図－a　図－b



図－c　図－d

図－2

```
DEFINITION MODULE Storage;
  FROM SYSTEM IMPORT ADDRESS;
  EXPORT QUALIFIED Allocate;
  PROCEDURE Allocate(VAR a:ADDRESS;n:CARDINAL);
END Storage.

IMPLEMENTATION MODULE Storage;
  FROM SYSTEM IMPORT ADDRESS;
  VAR lastused:ADDRESS;
  PROCEDURE Allocate(VAR a:ADDRESS;n:CARDINAL);
  BEGIN a:=lastused; INC(lastused,n)
  END Allocate;
BEGIN lastused:=∅
END Storage.
```


参考文献

Wirth, N.: MODULA-2 Berichte des Instituts für Informatik Nr. 36, ETH Zürich (1980)