1．PASCALコンパイラの論理的な分割について

電気通信大学 情報数理工学科　木村　友則・小林　光夫
計算機科学科　武市　正人

要約

プログラミング言語Pascalは、1971年にN.Wirthによって設計されて以来、現在までに70種類以上の計算機上で動いており、日本においても数種の移植がなされているが、これらはいずれも中型以上の機械が中心になっている。これは従来のPascalコンパイラが大型機の上で動くことを前提として、1パスコンパイラとして設計されていたためである。また、主として大学・研究所等で少人数でコンパイラを保守していく際に、従来の1パスコンパイラは必しも好ましい形であるとはいえない。

われわれは、Pascalコンパイラの処理を論理的に分割し、小型計算機の上で、standard Pascal（full Pascal）のコンパイラが動くようにしようとするものである。分割された処理の機能は、互いに独立であり、それぞれの機能を果たすプログラムは同じ構造をもつものである。こうしていくつかのパスに分割されたコンパイラは、処理内容が容易に理解でき、移植・保守に際しても扱いが簡単である。従来の1パスコンパイラは、Pascalプログラムで　5000～6000行であるが、ここでは、各パスが　1500行以下であり、プログラムの見通しがよくなっている。

コンパイラの移植に際しては、機械に依存するパスだけを書き直すことにより、従来よりも簡単に移植が可能である。

§1．Pascalの移植と保守の現状

Pascal News［1,2,3］によると、大部分のPascalコンパイラは、Pascal-P［4］をもとにして移植が行なわれている。Pascal-Pは、E.T.H.（チューリッヒ連邦工科大学）が提供しているPascalコンパイラのキットで、P-codeと呼ばれるコードを出力し、それを解釈することにより実行が行なわれる。キットを受け取って、

① P-codeのインタプリタを適当な言語で書いて、Pascal-Pコンパイラを動くようにする。
② Pascalで記述されているPascal-Pコンパイラを変更して、目的とする機械のコードを出力するように作り直す。

必要がある。また、②の部分でPascal-Pのかわりに、Trunkコンパイラと呼ばれるものを使うことも可能である。Trunkコンパイラは、H.H.Nägeliが作成したPascalコンパイラのキットで、各コンパイラのハードウェアに依存する部分がコメントによって記述されたものである。日本では、東京大学大型計算機センター

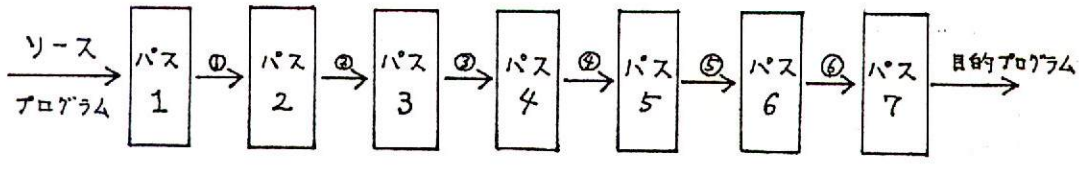
のPascal-8000[5,6]や、東京大学工学部情報工学のFACOM 230/38で、動いているPascal[7]が、Trunkを用いている。しかしながら、外国では、どのようなわけかTrunkを用いた例はみあたらない[1,2,3]。

Pascalの移植は、まず、①を行なうわけであるが、Pascal-Pは1パスコンパイラなのでかなりのメモリを必要とする。従って、中型以上の計算機でないと実現は困難である。①ができたとして、さらに、②を行なおうとすると、Pascal-Pの4000行以上にもわたるソースプログラム全体を読む必要がある。しかも、直接機械語を出力するようにすると、プログラムの行数は、5000～6000行になり、大学・研究所等で少人数でコンパイラを保守するのは大変である。さらには、こうして出来たコンパイラは、1000行程度のプログラムを処理するのに、150KB程度のメモリを必要とする（Pascal 8000の場合）。

これらのことから小型計算機でPascalのコンパイラを実現するには、

(i) fullのPascalはあきらめて、Pascalの部分言語を扱う。
(ii) コンパイラを、オーバーレイ構造にする。
(iii) コンパイラを、多走査型にする。
(iv) アセンブリ言語で作成する。

が考えられる。このうち、(i)は、Pascal-S[8]が、候補として考えられるが、これは機能が小さすぎるという問題が残ると思われる。(ii)は、東京大学の教育用計算機センターのCOSMO 700のPascal[7]で実際に行なわれている方法であるが、宣言の部分の処理と、手続き・関数の本体の処理とを、オーバーレイして処理を行なっている。しかし、この方法では、手続き・関数が宣言されるたびに、オーバーレイが行なわれ、処理効率が低下する。しかも記憶場所を節約する上では、あまり効果的でない。(iii)のものとしては、Solo[9, 10]システムで使われている、Sequential-PascalとConcurrent-Pascalがある。これらは7つのパスでできていて、6つの中間言語を設定し、それぞれのパスを、1つの独立なコンパイラとして作成している。



①～⑥：中間言語

ところが、各パスは独立にできていて、それぞれの構造は、すべて異っている。しかも、各パスのプログラムの行数の合計は約6800行であり、1パスのコンパイラより大きくなっていて保守が難しい。この問題は、各パスを独立なコンパイラとして、独立に中間言語を設定したために、各パスの処理が中間言語に左右されることになって生じたものと思われる。

これらのことから、われわれは、同一の形式の中間言語をもち、しかも各パスの処理の構造が同じであるように、多走査型コンパイラを作成することができれば、これらの問題をうまく解決できるのではないかと考えた。

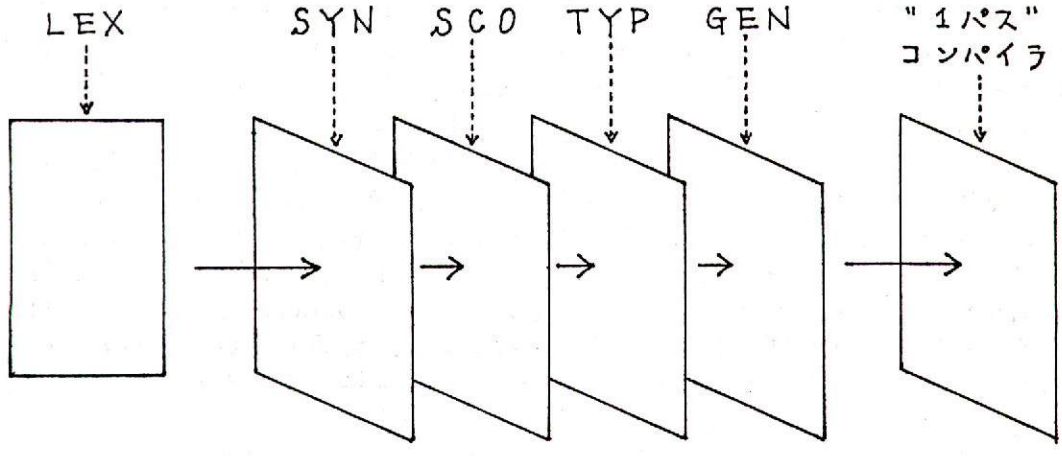
§2. コンパイラの機能分割と構成

コンパイラの仕事はふつう、
語彙解析 (lexical analysis)
構文解析 (syntax analysis)
意味解析 (semantic analysis)
目的プログラム生成 (code generation)
に分類される。

Pascalは、名前の有効範囲の規則 (scope rule) をもつので、意味解析は、
- 名前の有効範囲の解析 (identifier scope analysis)
- 対象の適合性の検査 (compatibility check)

に分けて考えられる。従って、
① 語彙解析
② 構文解析
③ 名前の有効範囲の解析
④ 対象の適合性検査
⑤ 目的プログラム生成
のように、コンパイラの処理を分けることを考える。

これらのうち、構文解析、名前の有効範囲の解析、対象の適合性検査、目的プログラム生成の処理は、構文に基づいて行なうのが自然であろう。また、①～⑤の機能を、一つにまとめた"1パス"のPascalコンパイラ [5, 6, 7] が、存在するのであるから、"1パス"のコンパイラのプログラムを論理的にこれらの5つ及び共通部分 (構文に関係がある部分) に分類することができると思われる。しかも、こうして分類したものを、共通部分を中心にして、5つのプログラムとして再構成すれば、各パスの処理の構造が同じであるような、多走査型コンパイラを実現することができるものと思われる。



先の①～⑤に相当する機能を、われわれは次のように定めた。

LEX ――― 語彙解析（lexical analysis）
Pascalのプログラムの金物表現（文字列）を、Pascalの基本記号および、名前、定数を識別して、それぞれを記号（symbol, 細胞 syllable）に変換する。記号は、扱いやすい整数値（と、補助的な情報）で表わす。名前に対しては、有効範囲とは無関係に、1つの綴りに対して、1つの番号を与える。

SYN ――― 構文解析（syntax analysis）
LEXで出力された記号の列を入力し、Pascalの構文の検査を行なう。ここでは構文解析した結果を、別の形式の中間言語として出力することはしない。誤りが発見されたときには、回復を試みる。

SCO ――― 有効範囲の解析（scope analysis）
構文上の誤りのないプログラムの記号の列を入力として、名前などの有効範囲の解析を行ない、有効範囲にたじて名前の番号を付けかえる。番号は以後の処理で参照しやすいようにつける。

TYP ――― 型の解析（type analysis）
対象（変数、定数、手続き、関数）の型の適合性を検査して、Pascalの文に相当する記号列の部分だけを出力する。

GEN ――― 目的プログラム生成（code generation）
目的プログラムを生成する。

これらの機能を定めるにあたって、機械に依存する部分は、できるだけうしろのパスで処理するよう配慮をした。

これまでコンパイラの機能を論理的に分割することについて、述べてきたが、実際にわれわれが、コンパイラを構成するにあたっては、1パスのコンパイラのプログラムそのものの分割は行なわなかった。1パスコンパイラでは、プログラムを解析して得られたすべての情報を、表に貯えておいて、目的プログラムを、生成する。従来のPascal のコンパイラでは、動的に表の要素を生成して、木構造を作っている。しかし、機能を分割して解析する際には、パス間で情報を伝達する必要が生じる。これには、動的なデータ構造は適切であるとはいえない。あるパスで作成した表の要素の番号によって、次のパスにおいても同じ対象を、参照することができるように表を作っておくと、情報の伝達を容易にすることができる。この観点から、各パスで用いる表は、配列の形式で実現することにした。コンパイラで動的なデータ構造を用いると、コンパイラを移植する際に、記憶場所の割り付けで考慮すべきことが多くなる。従って、われわれの実際の作業は、1パスのコンパイラのプログラムから、各パスの共通部分（これをskeleton＝骨格と呼ぼう）を編集して抜き出し、このskeletonに各パスでの処理の内容を肉付けしていくことで、各パスの作成を行なっている。

各パスは機能的には独立なので、効率を考慮して各パスを取り替えることや、最適化のパスを挿入するなどが容易にできる。また各パスで受渡しを行なうデータは、一応、整数値の列（Pascalの file of integer）の形式をとるが、このファイルは必ずしも、外部記憶である必要はない。

```
PROCEDURE CONSTDECLARATION;
  VAR LCP:CTP; LSP:STP; LVALU:VALU; EXPR:CEP;
  BEGIN
    IF SY <> IDENT THEN
      BEGIN ERROR(2); SKIP(FSYS+(.IDENT.)) END;
    WHILE SY = IDENT DO
      BEGIN NEW(LCP,KONST);
        WITH LCP@ DO
          BEGIN NAME := ID; IDTYPE := NIL; NEXT := NIL;
          END;
        INSYMBOL;
        IF OP = EQOP THEN INSYMBOL ELSE ERROR(16);
        IF SY=LCBRACK THEN
          BEGIN CONSTEXPRESSION(FSYS+(.COLON,SEMICOLON.),EXPR);
            IF SY=COLON THEN INSYMBOL ELSE ERROR(5);
            TYP(FSYS+(.SEMICOLON.)+TYPEDELS,LSP,FALSE);
            CONSTIMAGE(LSP,EXPR,LVALU);
          END
        ELSE CONSTANT(FSYS+(.SEMICOLON.),LSP,LVALU);
        ENTERID(LCP);
        LCP@.IDTYPE := LSP; LCP@.VALUES := LVALU;
        IF SY = SEMICOLON THEN
          BEGIN INSYMBOL;
            IF NOT (SY IN FSYS+(.IDENT.)) THEN
              BEGIN ERROR(6); SKIP(FSYS+(.IDENT.)) END
          END
        ELSE ERROR(14)
      END
  END (*CONSTDECLARATION*) ;
```

1パスコンパイラのプログラム（Pascal 8000）

下線を引いた部分は、ＳＣＯの処理と対応する部分である。

```
PROCEDURE CONSTDEFINITION;

 BEGIN
  WHILE SY=IDENT DO
   BEGIN
    (*IDENT*) NEXTSY;
    (*EQUAL*) NEXTSY;
    CONSTANT;


    (*SEMICOLON*) NEXTSY
   END
 END;
```

Skeleton

```
PROCEDURE CONSTDEFINITION;
 VAR I: IDINDEX;
 BEGIN
  WHILE SY=IDENT DO
   BEGIN NEWID(CONSTID, I);  IDNO:= I;
    (*IDENT*) NEXTSY;
    (*EQUAL*) NEXTSY;
    CONSTANT;
    ENTERID(I);
    IDTABLE(.I.).FORM:= SCALARSTYLE;
    (*SEMICOLON*) NEXTSY
   END
 END;
```

SCO

このようにして出来たPascalコンパイラの特徴をまとめる。

- 各パスが同じ構造をしているので読みやすく保守が容易である。
- 機械に依存する部分と依存しない部分が、明確に分離されているので、移植が簡単である。

§3. 移植について

われわれの作成したPascalコンパイラの移植について述べよう。移植を希望する人は、まず、

1. TYPの一部の書き替え（整数型、実数型などの語長の指定を行なう）を行なう。
2. GENの目的語生成の部分を目的とする計算機用に書く。

を行なう。2は、現存するものを参考にしながら書いていけば、かなり容易に書くことができる。TYPで出力される中間言語の内容さえ知っていれば、LEX、SYN、SCO、TYPの中身に関して、深い知識の無い段階でも、GENのプログラムを書くことは可能である。1、2の作業が完了したあとは、つぎのうちのいずれかの方法をえらぶことにより、目的とする計算機へのPascalコンパイラの移植ができる。

A. 現在動いているPascalコンパイラが利用できるときには、それを用いてLEX、SYN、SCO、TYP、GENを、それぞれクロスコンパイルすればよい。

B. 身近に適当なPascalコンパイラが無いときには、TYPで出力したコードを、インタプリタで解釈実行する。この場合、インタプリタは、ブートストラップするだけのものであり、限られた機能をはたせばよいものであるので、あらかじめ作った（たとえばFORTRANで書かれた）ものを、コンパイラのプログラムとともに、提供することができる。

C. Pascalで書かれた各パスを、人手で、アセンブリ言語などで書くことにより、コンパイラは小さく作ることができる。その際に各パスの大きさが1パスのものに比べて小さいので、作成は容易になる。

また、十分大きな計算機で使用する場合には、各パスのプログラムをつなげて1つのプログラムとしてコンパイルすることもできる。

この他の準備として、Pascalモニタの作成が必要である。Pascalモニタは、Pascalコンパイラとオペレーティングシステムの間の連絡および、コンパイラの管理を行なう。Pascalモニタの主要な処理は、

ファイルの入出力、
標準手続き、関数、
記憶場所の動的割当て

である。このうち、コンパイラの移植に際して必要な機能は、ファイルの入出力だけでよい。Pascalモニタの大きさは、すべての機能を含めると、10～20KBであるが、コンパイラの移植だけならば、その半分程度のPascalモニタをアセンブリ言語などで書けばよい。

§4．おわりに

Pascalコンパイラの機能を論理的に分割したことにより、文法書で明確でない事項についての、コンパイラでの処理が明白にされた。たとえば、名前の有効範囲の取扱いに対しては、ＳＣＯのパスのみが関与している。

コンパイラは、現在開発中（発表の際には完成する予定）であるが、ＬＥＸ、ＳＹＮ、ＳＣＯが完成している。これらのプログラムの行数と大きさは、

ＬＥＸ	約３８０行	２３ｋＢ
ＳＹＮ	約８５０行	２３ｋＢ
ＳＣＯ	約９８０行	３０ｋＢ

である。大きさには、Pascalモニタの大きさは含まれていないので、この大きさにモニタの大きさを加えたものが、プログラムを実行するための大きさとなる。現在使用しているPascalコンパイラ（Pascal 8000 をHITAC8350に移植したもの）は、整数型、論理型の変数を、４バイトで扱っているが、それらを２バイトで扱うようにすることにより、もうすこし小さくなると思われるので、あとのＴＹＰ、ＧＥＮをうまく構成すれば、32 kB 程度の記憶場所で、fullのPascalコンパイラが動くことも夢ではないであろう。

参考文献

1. Pascal News Nos. 9,10 PUG, Univ. of Minesota, Sept. 1977.
2. Pascal News No. 11 PUG, Univ. of Minesota, Feb. 1978.
3. Pascal News No. 12 PUG, Univ. of Minesota, Jun. 1978.
4. Nori, K.V., Ammann, U., Jensen, K., Nägeli, H.H.: The Pascal 'p' Compiler: Implementation Notes, Nr. 10, Berichte des Institutus für Informatik, E.T.H. (Dec. 1974).
5. Hikita, T., Ishikata, K.: PASCAL 8000 Reference Manual, Technical report 76-02, Dept. of Information Science, University of Tokyo, 1976.
6. Ishikata, K, Hikita, T.: Bootstrapping PASCAL Using a Trunk, Technical report 76-04, Dept. of Information Science, University of Tokyo, 1976.
7. Takeichi, M.: PASCAL Implementation and Experience, Journal of the Faculty of Engeneering, University of Tokyo, Vol. 34, No. 1. (1977), pp. 129-136.
8. Wirth, N.: Pascal-S: A Subset and its Implementation, Nr. 12, Berichte des Instituts für Informatik.
9. Hartmann, A.C.: A Concurrent Pascal Compiler for Minicomputers, Lecture Notes in Computer Science, No. 50, Springer Verlag, 1977.
10. Hansen, P.B.: "The Architecture of Concurrent Programs," Prentice-Hall, 1977.
11. 和田英一：プログラム言語 Pascal, bit, 1978年 1月号～10月号.