北海道大学工学部研究報告
第119号（昭和59年）

Bulletin of the Faculty of Engineering,
Hokkaido University, No. 119(1984)

プログラムスキーマに基づくプログラミング環境システム

宮本　衛市　北山　泰英*　桃内　佳雄　竹村　伸一

（昭和58年9月30日受理）

A Programming Environment System Based on Program Schemata

Eiichi MIYAMOTO, Yasuhide KITAYAMA, Yoshio MOMOUCHI and Shin-ichi TAKEMURA

(Received September 30, 1983)

Abstract

This paper describes an initial implementation of an advanced programming environment system which supports programmers through interactive editing based on program schemata. The motivation is that experienced programmers possess a large quantity of program schemata which they have used before, and the technique of how such schemata should be applied to the problem confronting them. As a first step, the system presents program schemata available to the programmer's retrieval, while he applies to his program. Program schemata are provided with necessary program components to implement some specific job, so that the programmer can apply them to his program without the possibility of the invasion of 'bugs' such as missing initialization or wrong looping.

1. はじめに

われわれがプログラムの作成を行う場合，以前に使用したことのあるプログラム例や，典型的なアルゴリズムを参考にして作業を進めることが多い。すなわち，プログラムは種々のプログラム例をパターン化して持ち，解決すべき問題にこのパターンを適用してプログラムを作成しているのであって，活用できるプログラム例をどのくらい所有しているか，そしてそれらのプログラム例を直面する問題にどのくらい活用できるかが，プログラミングの能率，ひいてはプログラマとしての能力を決定するといっても過言ではない。

プログラミングを支援するツールとして，まずエディタがあげられる。通常のエディタは文字列を対象としたテキスト処理を行うためのものであるが，構造化エディタ[1),2)]といわれるエディタは，編集対象とするプログラミング言語の構文規則に従ってテキスト編集を行う。それゆえ，構造化エディタのもとでは，プログラマは使用する言語のレベルでの対話でプログラムを作成することができる。そのため，構造化エディタは単に作業の能率化や，少なくとも構文的には正しいプログラムの作成を支援するばかりでなく，トップダウンプログラミングや段階的詳細化法[3)]などのプログラミング方法論に従ってプログラムの作成を行うときには，きわめて有効なプログラミング環境をプログラマに提供する[4)]。

情報工学専攻　情報システム工学講座
*現在は三菱電機株式会社勤務

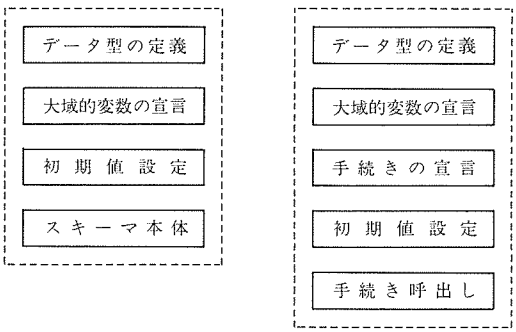
そこでわれわれは，構造化エディタの機能を一層強化して，典型的なプログラム例をプログラムスキーマとして蓄積したデータベースに基づき，プログラムスキーマも編集対象に加えたエディタ SCORE (Schema ORiented Editor) の開発を行っている。前述したように，経験を積んだプログラマは多数のプログラム例と，それらの活用技術を習得している。われわれは経験を積んだプログラマの能力をエディタに持たせ，より高度なプログラミング環境を作るための第1段階として，SCORE に多数のプログラムスキーマを与えておき，その活用はプログラマに任せるような環境を設定した。したがって，プログラマは自分の必要とするプログラムスキーマを SCORE から検索しなければならない。そのために，SCORE はデータ構造に基づいてプログラムスキーマを木構造に分類して蓄積する。これは SCORE がプログラミング言語 Pascal を前提としており，アルゴリズムとデータ構造を一体として考えるのは抽象データ型[5]の考えであり，データ構造を基本にした方がアルゴリズムの蓄積・検索が容易であると考えたからである。このようにして蓄積されるプログラムスキーマの正当性が証明されていれば，プログラムスキーマの指示に従って作成すべきプログラムに組み込み，アルゴリズム上正しいプログラムを作成してゆくことができる[6]。

SCORE が目指しているようなプログラミング環境として PA (Programmer's Apprentice)[7] が有名である。PA はその名が示すように，プログラマがプログラムを作成するのを有能な助手として手助けするシステムであり，プログラマとは限られた範囲内ではあるが，英語で会話を行う。このシステムの特徴はアルゴリズム例をプランと称する，プログラミング言語とは独立した内部表現形式で所有しており，プログラマとはこのプランを通して会話的に編集を行い，プログラマが指示するプログラミング言語でプランを具体化する。SCORE ではプログラマが自分の必要とするアルゴリズムは自分で探すのに対し，PA では会話を通してシステム側が探索する。PA のような方式ではシステムが所有するアルゴリズムが多くなるにつれ，プログラマの要求に応じたアルゴリズムを取り出すことが一つの大きな問題点となろう。

以下，2．では SCORE の設定するプログラミング環境と支援機能について述べ，3．では SCORE の処理方式を概説し，4．で SCORE の問題点をふまえての今後の展開について考察する。

2．システムの設定する環境と支援機能

1．で述べたように，SCORE はプログラムスキーマに基づくプログラミング環境を設定するので，プログラムスキーマをどのように構成し，蓄積し，そしてプログラマの利用に供するかが最大の眼目となる。SCORE は Pascal をプログラミングの対象とする言語として選び，したがってプログラムスキーマも Pascal で記述されたプログラムの，いわゆる断片であるが，プログラマがプログラムスキーマの不完全部分を補完するのであって，プログラムスキーマはちょうどプログラムのテンプレートに対応させることができる。



(a) インライン型スキーマ　(b) 手続き型スキーマの例

図—1 プログラムスキーマの一般形式

```
(* DATA TYPE *)
  TYPE
   LISTP=@LISTREC;
   LISTREC=RECORD
             KEY: <<1>>;
             NEXT:LISTP
           END;
(* GLOBAL VARIABLES *)
  VAR
   P,Q:LISTP;
(* INITIALIZATION *)
  P:=NIL;
(* BODY *)
  NEW(Q);
  Q@.NEXT:=P;
  P:=Q;
  Q@.KEY:= <<2>>;
```

(a) インライン型スキーマの例

```
(* DATA TYPE *)
  TYPE
   TREEP=@TREEREC;
   TREEREC=RECORD
             KEY: <<1>>;
             LEFT,RIGHT:TREEP
           END;
(* GLOBAL VARIABLES *)
  VAR
   ROOT,P:TREEP;
(* PROCEDURE DECLARATION *)
  PROCEDURE SEARCH(P:TREEP);
   BEGIN
    IF P=NIL THEN
      <<2>>  (* FAILED TO FIND *)
    ELSE
     IF P@.KEY> <<3>> THEN
      SEARCH(P@.LEFT)
     ELSE
      IF P@.KEY< <<3>> THEN
       SEARCH(P@.RIGHT)
      ELSE
        <<4>>  (* FOUND *)
   END;
(* INITIALIZATION *)
  P:=ROOT;
(* PROCEDURE CALL *)
  SEARCH(P);
```

(b) 手続き型スキーマの例

図—2 プログラムスキーマの例

2．1 プログラムスキーマの構成

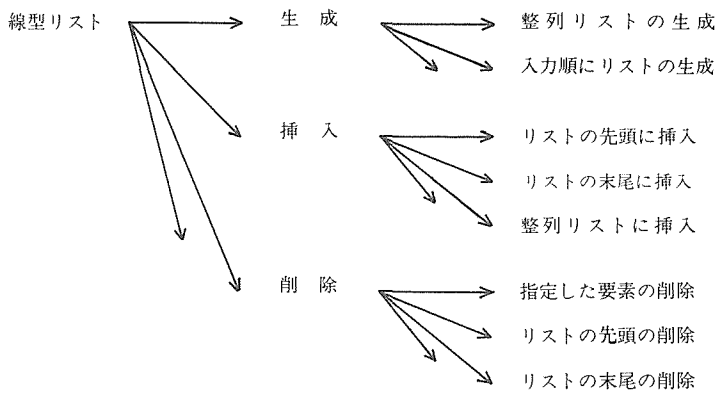
プログラムスキーマには，作成するプログラム中にプログラムスキーマを直接埋め込む，いわゆるインライン展開を前提としたスキーマと，手続き宣言を先行させてそれを呼び出すことを前提としたスキーマの2つの形式がある。前者をインライン型のスキーマ，そして後者を手続き型のスキーマと呼ぶことにする。図1に両スキーマの一般的な構成を示す。

プログラムスキーマは，あるアルゴリズムの具体的な記述であり，Pascalプログラムの一部分を構成する。アルゴリズムを記述するためには変数が必要であり，したがってプログラムスキーマにはそこで使用する変数を宣言する部分が必要である。ただし，プログラムスキーマは2．2でも述べるように，データ構造を基本とした木構造に分類されているので，そのプログラムスキーマが所属するデータ型は定義ずみのものとして使用することができる。ここでいう変数は，一般的に大域的変数の取扱いを受けるものとする。

変数の初期値設定の部分では，アルゴリズムを記述するプログラムスキーマ本体で使われている変数で，本体が駆動される以前に初期設定されている必要があるものについて，その初期化を記述する。ただし，この初期化は変数宣言の部分に列挙されている大域的変数に関するものである。プログラムスキーマの本体で，そのスキーマが目的とするアルゴリズムを記述する。この場合，記述量やアルゴリズムの完結性，あるいは再帰的な呼出しを伴うアルゴリズムであるかなどにより，図1に示すいずれかの型のスキーマを選ぶ。図2に両スキーマの例をあげるが，スキーマの各構成部分は作成するプログラム中に一体として埋め込む必要はなく，各部分をプログラマが必要な個所に挿入してゆく。ただし，各部分がすべて埋め込まれたかどうかはシステムが監視している。なお，図2の型定義は木構造の上位で定義されていれば不要である。

2．2 プログラムスキーマの分類と蓄積

一般に，ある一つのデータ構造を設定すると，それを使ったいくつかのアルゴリズムが考えられる。図3に線形リストを想定したデータ構造の例を示すが，プログラムスキーマとしてはリス



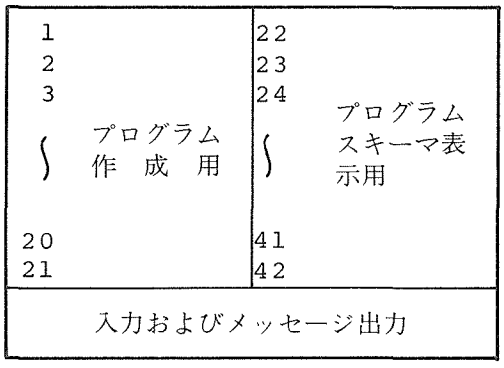
図—3 アルゴリズムの階層構造

表1 プログラムスキーマの編集に関するコマンドの一覧

コ マ ン ド	機 能
N	プログラムの新規作成
OB［データ構造名］	データ構造の提示要求
OP［スキーマ名］	スキーマの提示要求
I ＊ *lnum*	スキーマの挿入
I *rnum* [*rnum*] *lnum*	スキーマの構成要素の挿入
L ［手続き名］	指定した手続きの表示
T [*tnum*] 〈ソーステキスト〉	補完部分の埋込み
IP [*lnum*]	手続き宣言の挿入
LF, LB	左画面の上方，下方へのスクロール
RF, RB	右画面の上方，下方へのスクロール

lnum: 左画面の行番号，*rnum*: 右画面の行番号，*tnum*: 補完部分の番号

トの生成，挿入，削除などの抽象レベルでまず分類し，さらにこの抽象レベルでの分類を詳細化してゆくと，最終的に個々のアルゴリズムに到達するような木構造に展開する。すなわち，木構造の根では基本的なデータ構造を，節では分類項目を，そして葉の部分で具体的なアルゴリズムを保持させるものとする。

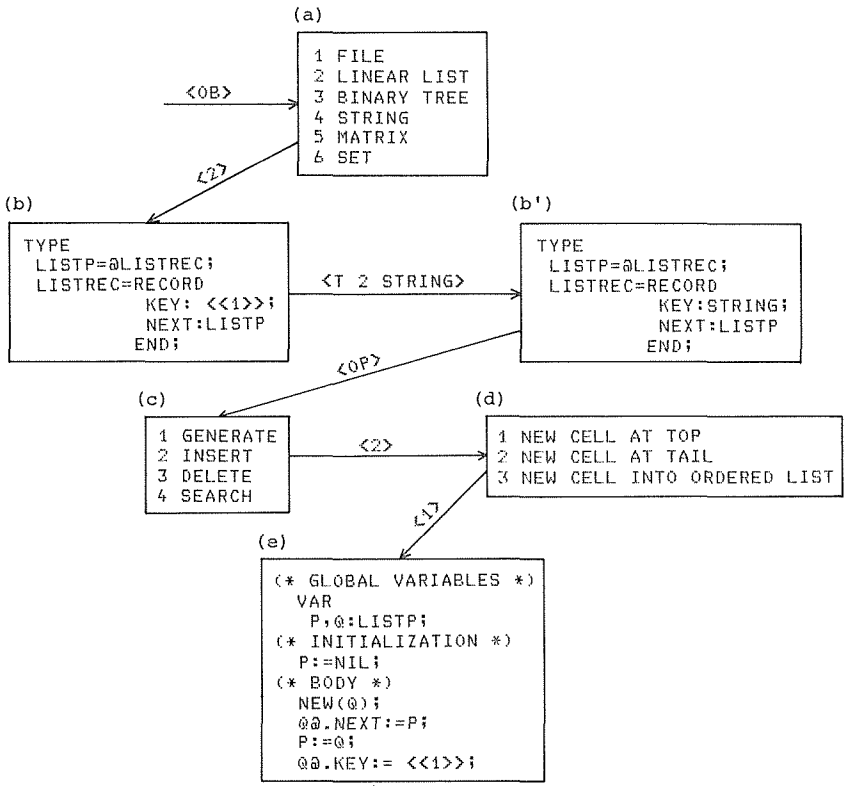


図—4 CRT 画面の分割

2．3 プログラムスキーマに基づく支援機能

SCORE はプログラマと CRT ディスプレイを介して対話する。そのため，CRT 画面を図4に示すように分割して使用する。画面上部 21 行の左半分を作成中のプログラムの表示用として使い，右半分はプログラムスキーマの表示用として使う。下辺3行は入力行およびシステムからのメッセージ表示のために使用する。表1は SCORE のもつ，スキーマの編集に関するコマンドを一覧表にしたものである。SCORE のもとでの編集作業を例により説明する。

図5は線形リストのデータ構造に対して，新しい要素を頭部に追加するプログラムスキーマの



図—5 プログラムスキーマの検索例

```
1 PROGRAM PASCAL(INPUT,OUTPUT);
2  BEGIN
3  END.
```

図—6 プログラムの初期設定

```
1 PROGRAM PASCAL(INPUT,OUTPUT);
2  TYPE
3   STRING=PACKED ARRAY[1..8] OF CHAR;
4  BEGIN
5  END.
```

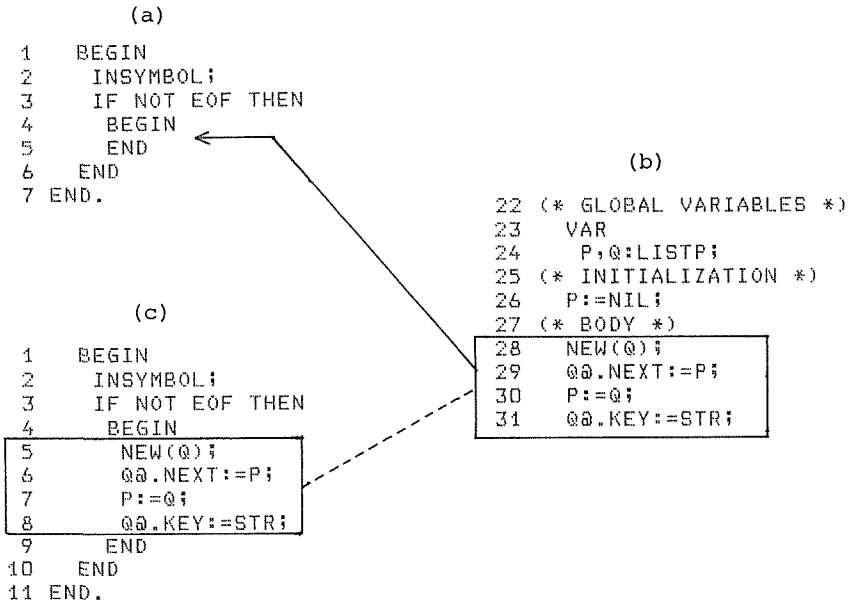
(a) 挿入前のプログラム

```
1 PROGRAM PASCAL(INPUT,OUTPUT);
2  TYPE
3   STRING=PACKED ARRAY[1..8] OF CHAR;
4   LISTP=@LISTREC;
5   LISTREC=RECORD
6             KEY:STRING;
7             NEXT:LISTP
8           END;
9  BEGIN
10 END.
```

(b) 挿入後のプログラム

図—7 データ構造の挿入

検索例である。まず，コマンド〈OB〉で同図(a)で示すようなデータ構造の一覧表が右画面に提示される。(ただし，コマンドの指示で〝〈〟と〝〉〟はコマンド入力を明示するためのものであって，実際には入力されない) そこで〈2〉を入力して線形リストを選択すると，そのデータ構造（b）が表示される。このデータ構造はポインタ型とレコード型で表現されており，レコード型のフィールドKEYのデータ型が未定となっているので，コマンド〈T 1 STRING〉を入力するとフィールドKEYのデータ型としてSTRINGが与えられ，画面は(b')のように修正される。ただし，型STRINGは定義ずみとする。次にコマンド〈OP〉で線形リストに関するアルゴリズムの提示を求めると（c）が提示され，〈2〉を入力して2番目の項目を選択すると（d）が提示され，さらに〈1〉を入力すると（e）のプログラムスキーマが表示され



図—8 プログラムスキーマの挿入

る。これを必要に応じて左画面に挿入してゆけばよい。図5の表示はすべて右画面を用いて行われる。

一方，画面の左半分でプログラムの作成を行うが，プログラムの作成を新たに行う場合，コマンド〈N〉の投入から始まる。これにより左画面には図6が表示されて，プログラム作成の初期化の処置がとられる。いま図5で示す検索中，(b′)で示すデータ構造を取り込みたければ，コマンド〈I ＊ 3〉により，図7（a）の3行目の下に，右画面に表示されていた内容が挿入されて，左画面は同図（b）のように更新される。また，図8（a）に示す段階のプログラムに対し，右画面に表示されたプログラムスキーマの本体を，コマンド〈I 28 31 4〉で左画面に挿入すると，左画面は同図（c）のように更新される。

表1にあげたコマンドの中で，前述の説明に現われなかったもののなかで，〈L 手続き名〉は左画面上にすでに作成ずみの手続きを指定して表示させ，編集対象とする。このとき，指定した手続きを入れ子に含む手続きを上部に表示して，手続き宣言の入れ子関係を通知する。また，コマンド〈IP〉は手続き型のスキーマの宣言部を左画面に表示されているプログラム中に挿入するものであり，手続き呼出しはコマンド〈I〉で挿入する。

なお，左画面で作成するプログラムの修正等の編集作業は，構造化エディタ POESY[2] の支援機能を準用する。

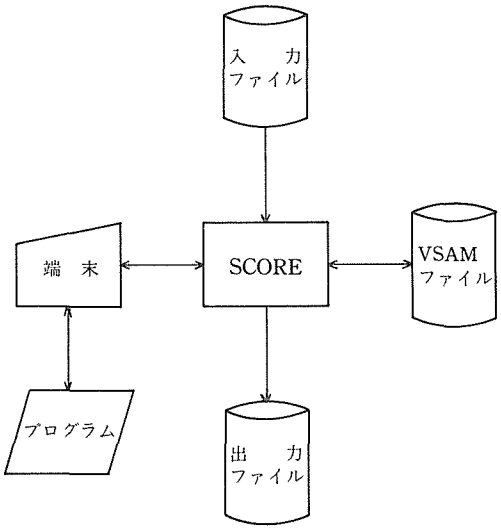
3．SCORE の内部処理方式

SCORE の内部処理方式は，基本的には構造化エディタ POESY と同じであり，作成中のプログラムは木構造からなる内部表現のもとで管理され，構造的な編集に備えている。SCORE の場合には，これにプログラムスキーマの管理が追加される。図9に SCORE のシステム構成を示すが，SCORE の使用するファイルのうち，プログラムスキーマ用のファイルは VSAM ファイルである。以下に主な処理方式について述べる。

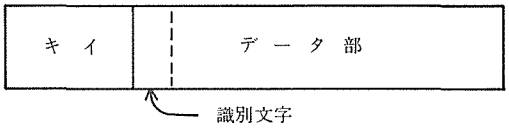
3．1 プログラムスキーマの蓄積と参照

プログラムスキーマはVSAMファイル上で木構造に展開する。ここで用いたVSAMファイルの形式は図10に示すように，各レコードごとに20文字のキィをもった形式である。データ部には木構造上の節の位置に応じて，データ構造，分類項目あるいはプログラムスキーマが与えられる。これらのレコードを区別するため，データ部の先頭の文字を識別用に用いている。

レコードのキィを木構造上の節の名前として使うので，節のデータ部には，その節からさらに分類される節や葉のレコードキィが格納されている。葉に対応するレコードには，プログラムスキーマのほかに編集時に備えて，スキーマの形式，他の手続き型のスキーマを呼び出しているかどうか，大域的変数を使用しているか，初期化の部分を含んでいるかなどについての情報もあわせて格納されている。



図—9　SCOREのシステム構成

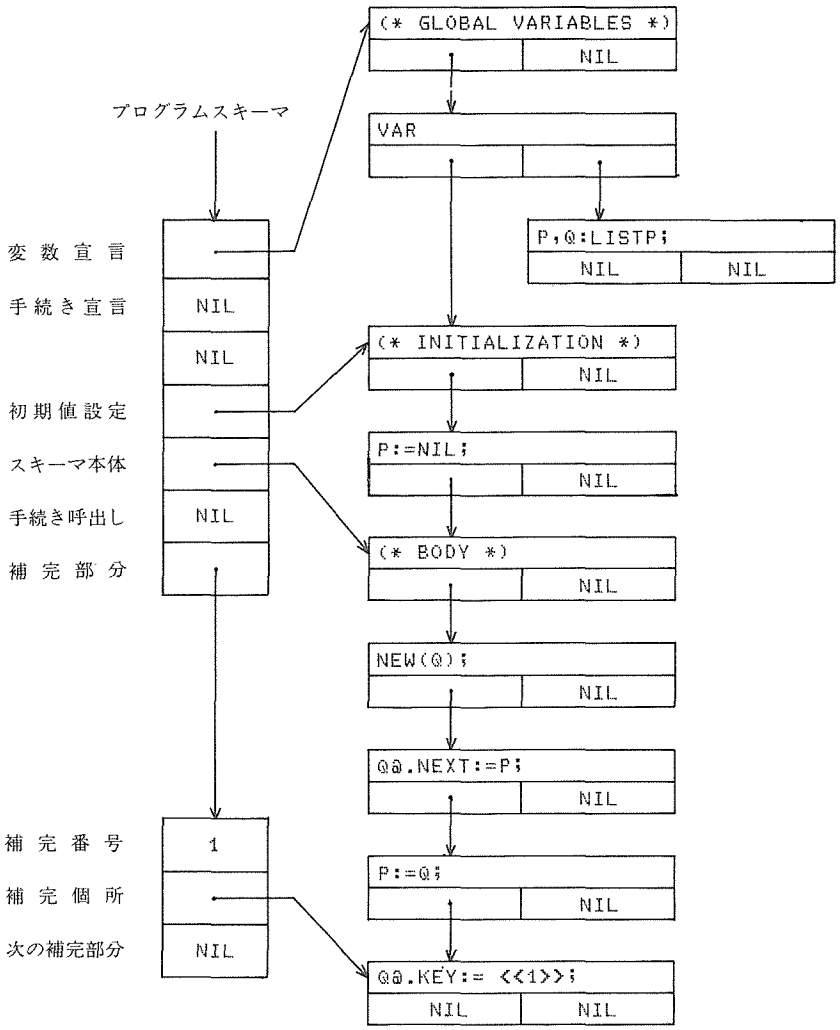


図—10　VSAMレコード

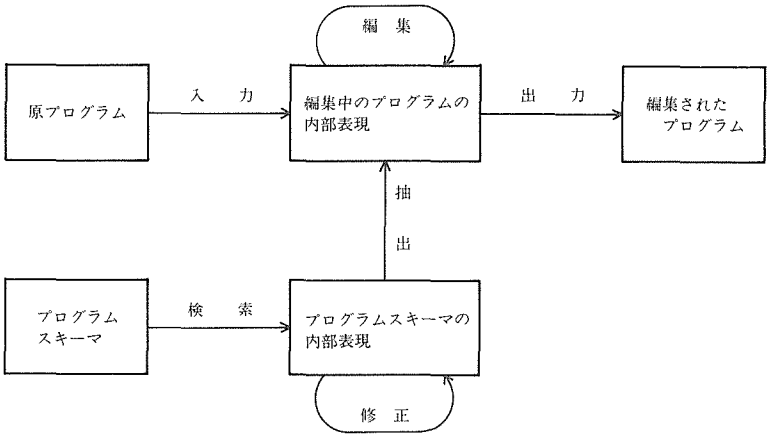
3．2 プログラムの内部表現と編集処理

作成中のプログラムおよび呼び出されたプログラムスキーマは，手続きおよび文を単位として木構造に基づく内部表現に変換される。これにより，構造的な編集処理を容易にするためである。特に，プログラムスキーマはプログラマに提供されるプログラムの部品であるので，スキーマからの構成要素の抽出や管理が容易になるように表現されている。そのため，図11の例で示すように，プログラムスキーマを木構造に変換し，各構成要素の先頭やまだ未定になっている個所なども同時に掌握している。

図12はSCOREの処理手順の概要を示したものである。プログラマの指示に基づき，外部ファイルからプログラムあるいはプログラムスキーマを読み込み，それらを内部表現に変換した上で編集処理を施こす。編集終了後あるいは編集中断時には，プログラムを内部表現から再びソースイメージに直して外部ファイルへ出力する。編集処理は手続きまたは文単位で行うので，内部表現上では主としてポインタのつなぎ替えですむ。プログラマが埋め込むプログラムスキーマの未定部分は，構文上の文法単位に基づいて部分解析を行った後に取り込む。



図—11 プログラムスキーマの内部表現



図—12 SCORE の処理手順

4. あとがき

プログラムスキーマは，そこで必要とする最小限の情報で組み立てられている。すなわち，データ型の定義，使用する変数の宣言，初期値設定あるいは手続き宣言などを必要に応じて網羅してある。したがって，これらを適所に挿入してプログラムを組み立てていけば，プログラムスキーマを正しく働かせることができる。しかし，それらの編集指示，さらに既製のプログラムスキーマを修正して自分の目的とする問題に応用するのも，すべてプログラマに任せられている。また，プログラムスキーマの検索も，プログラマが木構造をトップダウンに探索することを前提としている。SCOREは構造化エディタからプログラムの問題領域へ一歩踏み込むことを目指したエディタであるが，もちろんまだ模索の域を出たものではない。SCOREの作成および使用実績に基づいて，現在のSCOREの主な問題点とそれらに対する対応策についてまとめると次のようになる。

（1）プログラムスキーマを探索するのに，メニュー表示による探索のみでは時には繁雑である。特にプログラマが既知のプログラムスキーマを呼び出すときにはなおさらである。そのためには，直接必要なプログラムスキーマを呼び出すことができるほか，キィワードなどで必要とするプログラムスキーマを指定して，エディタ側に検索させることも望ましい。また，木構造の中をトップダウンばかりでなく，自由に動いて探索できることも必要である。

（2）プログラムスキーマ自体としてはアルゴリズム上完結しているが，これにプログラマが修正を加えると，一般にはエディタはその意図を理解することはできず，修正に伴うプログラムの正しさの検証は放棄せざるをえない。これに対し，エディタにプログラム作成上の知識を与えておいて，エディタの理解できる範囲内で修正に応じることも考えられるが，その範囲は一般にプログラマにとって窮屈な足枷になるし，またそれを感じさせないようにするためには膨大な知識をエディタに付与する必要があろう。もちろん，これは究極的なエディタの姿であろうが，そこに到達するまではやはりプログラマに修正を委ねることになる。そこで次善の策ではあるが，プログラムスキーマのデータベースに，スキーマ自体のほかにプログラムスキーマに関する種々の情報も蓄えておき，プログラマの要求に応じて適切な情報を提示し，プログラマの便宜を図ることが考えられる。

（3）現在のSCOREはデータ構造を基本としてプログラムスキーマを体系づけているが，さらにアルゴリズムの観点からも体系づけるなどして，いくつかの検索経路があることが望ましい。これはデータ構造に重きを置かないアルゴリズム，例えば最大公約数を求める問題のように，データ構造の観点からは分類しにくいものもあるからであり，またアルゴリズムの方がプログラムスキーマの目的を端的に表わしていることが多いからである。

われわれは上記のような考察に基づき，現在のSCOREの改良を計画しており，よりよきプログラミング環境の探究を進める予定である。

参考文献

1）Donzeau-Gouge,V. et al.: Rapport de Recherche IRIA, **144** (1975)
2）宮本衛市，浅見可津志：情報処理学会論文誌, **20** (1979), 6, pp.474～480
3）Wirth,N.: Commun. ACM, **14** (1971), 4, pp.221～227
4）Shapiro,E. et al.: SIGPLAN NOTICES, **16** (1981), 8, pp.50～57
5）Liskov,B.H. et al.: SIGPLAN NOTICES, **9** (1974), 4, pp.50～59
6）Dershowitz,N.: 5th Inter. Conf. Software Engineering, (1981), pp.79～88
7）Waters,R.C.: IEEE Software Engineering, **SE-8** (1982), 1, pp.1～12