データベース・システム 58－1
（１９８７．３．１６）

言語依存型構造エディタにおける構文データの管理

松田　孝子
東北大学大型計算機センター

プログラミング言語の構造に依存して操作できるエディタを実用性の高いＦＯＲＴＲＡＮについて実現した。ＦＯＲＴＲＡＮはボトムアップに成長してきた言語であるため他の言語と比べて形式的扱いが難しい。また，他との互換性や慣用的用法を許すためにＪＩＳ規格を遵守する処理系は現実にはサービスされず頻繁に機能の拡張が行われる。この実情に柔軟に対処するために，構文規則をプログラム本体から独立させた構造エディタＦＥＤＩＴを開発した。ここではこの構文データの管理を中心に述べた。構文を表現するためにグラフ記号を導入し，グラフ記述言語ＧＲＡＤＬを設けた。構文規則を一続きのメイングラフとその中から呼出すサブグラフ群からなる構文グラフとして表現した。構文解析は入力ステートメントが構文グラフ上に過不足なく張れる枝が存在するかどうかによって行う。エディタが編集対象とする構文要素はサブグラフとし，構文解析時にこのサブグラフを通過した時点で検出する。本システムでは構文データとプログラムの独立を図ったので，構文の管理が容易でエディタの改訂を迅速に行える。また，複数の構文データを動的に選択すれば多目的エディタとして発展できる。

Storage and Use of Syntax Data in a Language Oriented Editor

Takako MATSUDA
Computer Center, Tohoku University
2-1-1 Katahira, Sendai 980 Japan

Storage and use of syntax data of a syntax directed editor, which is available for editing of programs written in FORTRAN, is described. The language specifications of FORTRAN, a bottom-up growing language, are often revised and enlarged. And so, we intended to develop a flexible editor, named FEDIT.

The FEDIT has the syntax data independent of the system program. The syntax is represented with directed graphs which consist of a main graph and several subgraphs. The syntax analysis of this editor is performed by fitting a FORTRAN statement to a branch of the syntax graph exactly. Each of the editing elements is stored as a subgraph and is detected in time of search through it. Consequently, it becomes easy to manage the syntax data, and it is possible to grow this editor into a multi-purpose structure editor.

1．はじめに

オンラインの利用形態が普及する中でエディタは最も頻繁に使用されるツールとなっている。エディタを用いてプログラムやデータを編集する場合には，テキストの論理構造や文字列の意味に従う操作が必要になることがある。しかし，現在実用化されているエディタは文字列処理用のものであるため，このような操作を行うことはできない。これを解決するには，データ構造を扱うデータベースやプログラミング言語に依存する構造エディタが有効である 1),2)。

我々は，科学技術計算の分野で最も利用度の高いプログラミング言語ＦＯＲＴＲＡＮについて，構造エディタＦＥＤＩＴ（Fortran syntax directed editor）を開発した 3)~5)。ＦＯＲＴＲＡＮはボトムアップに成長してきた言語であるため，文法はＪＩＳに準拠するだけでなく他の処理系や慣用的用法などを包含するのが殆どで，仕様の拡張も頻繁に行われ，多種のものが実用になっている。また，ユーザ利用環境の中では複数種類のコンパイラが同時にサービスされている場合も多い。そこで，ＦＥＤＩＴでは構文データをプログラム本体から独立にし管理しやすいようにした。ここでは，構文データの格納と利用を中心に述べる。

2．ＦＥＤＩＴのシステム概要

ＦＥＤＩＴは，関連する５つのファイル，編集処理を遂行する３つのプログラムおよび構文を記述するための言語ＧＲＡＤＬ（Graph description language）の処理系からなる（図１）。

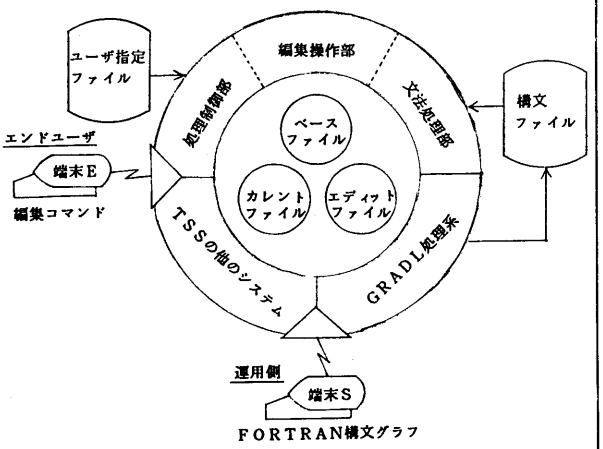

図１　ＦＥＤＩＴのシステム構成

各ファイルおよびプログラムはつぎのものである。

- ユーザ指定ファイル……ソースプログラムファイル
- ベースファイル……ユーザ指定ファイルの内容を複製し，定型に整列した作業用ファイル
- エディットファイル……ベースファイルの中からつぎのいずれかにより抽出した編集モジュールを格納する作業用ファイル

プログラム名	（指定の名前のプログラム単位）
*ＭＡＩＮ	（主プログラム）
*ＢＬＫＤＴ	（初期値設定副プログラム）
*ＡＬＬ	（プログラム全体。省略時解釈）

- カレントファイル……ＴＳＳの他のシステム（他のエディタやＦＯＲＴＲＡＮコンパイラなど）と連係をとるための一時的な仲介ファイル
- 構文ファイル……対象とする言語の構文規則を格納したファイル
- 処理制御部……ＦＥＤＩＴの主ルーチンで，ユーザとの応答，作業用ファイルの管理を行うプログラム
- 編集操作部……ユーザの指示に従ってエディットファイルを編集するプログラム
- 文法処理部……構文ファイルを用いて文を検査，解析するプログラム
- ＧＲＡＤＬ処理系……ＧＲＡＤＬで記述された構文データを翻訳し構文ファイルに格納するプログラム

ＦＥＤＩＴの編集コマンドを表１に示す。

表１　ＦＥＤＩＴの編集コマンドの形式

ｖｘ／ｅ１／ｅ２／ｎ

ｖ　：動詞。つぎの操作種別を指示する（下線部分を入力）。

Find	前進探索	Back up	後退探索
Print	印刷表示	Replace	置換
Insert	挿入	Delete	削除
Verify	照合モードの設定		
Noverify	照合モードの解除		
Merge	編集モジュールをベースファイルに併合		

ｘ　：要素属性。つぎの操作対象の文法項目を指定する（下線部分を入力）。

Name	英字名	Label	文番号
Statement	文または文のクラス名		

ｅ１：検索要素。ｘに応じて英字名，文番号または文の名称かつぎの文のクラス名を指定する（下線部分を入力）。

*Control	制御文
*Assignment	代入関係
*Input/output	入出力関係
*Declaration	宣言関係
*Procedure	手続き文

ｅ２：置換要素。ｖがReplace の時，ｘに応じて置換する新しい英字名か文番号の綴りを指定する。

ｎ　：操作の繰り返し回数。整数か「最後まで」を表す＊。省略するとｎ＝１。

ＦＥＤＩＴは，編集コマンドを用いて文（または注釈行）を単位として進行するサーチポインタを移動しながらテキストの編集を行う。文が入力または変更された時に，構文ファイルに格納されている構文データを用いて検査・解析を行う。

3．構文データの表現と記述言語

3.1 構文グラフ

ＦＯＲＴＲＡＮは他の言語と異なり，予約語を持たずまた空白を区切り文字として使用できないので，構文解析の際にレコードの先頭部分だけで文を確定することができない。そのため，最終的にはつぎの例のように代入文である可能性を考慮する必要がある。

例） DO 10 I = M + N
WRITE(6,200) = X + A

このような言語の特性からＪＩＳでは構文の図式表現を用いている 6)。しかし，ＪＩＳの構文図は人間には理解し易いものであるが，機械処理には適さないので，ＦＥＤＩＴでは，以前ミニコン上に実現した構文チェッカ 7) で用いたグラフ記号を改訂した表２の記

表2　構文グラフの記号とＧＲＡＤＬステートメント

種　類	グラフ記号	ＧＲＡＤＬステートメント	備　考
atom	(atom) (atomの列)	*ATOM「$c_1c_2\cdots c_n$」;*	atomは文字 ci
atom any	(any)	*ATOM「」;*	任意の一文字
atom nil	▽t	*NIL t;*	tは文のクラスコード（D,C,P,M,Aのいずれか）
statement class (upward-stopper)	●t	*STCLASS t;*	
look up atom table	g	*LOOK g;*	gはテーブル名
call subgraph	(g)	*CALL g;*	gはサブグラフ名
entry	▽g	*ENTRY g;*	
return	▽	*RETURN;*	
branch		*BRANCH l;*	左の枝優先で分岐 lは右側分岐先のラベル
link	——→	*LINK l;*	lはリンク先ラベル
link if counter+ ≦ limit	——≦→	*IFLINK #c l;*	カウンタcがリミット値以下ならラベルlにリンクする. ここを通過する毎にカウンタを1増す.
loop while s- ≠ 0	——s=→	*LOOP #s; ～～ LEND;*	スタックsの値による回数だけ繰り返す.
set limit	◇n	*SET #c=n;*	カウンタcのリミット値をnにする.
stack atom	▼	*STACK #s;*	この間に現われるatomの列をスタックsにセットする.
stack end	▲	*SEND;*	
connection	▽i		グラフ上の連結記号 iは数字
comment		*COMMENT <comment>;*	<comment>は注釈

号を用いて書き改めた。

各グラフ記号は以下のように用いる。

(1) atom はソーステキスト中に出現する文字と照合することを表す。 atom nil はグラフの終端記号で，ソーステキスト中に出現する文字がこれ以上存在しないことを表す。

(2) look up はソーステキスト中に出現する文字を予め宣言された文字集合のテーブルの中の文字と照合することを表す。

(3) atom nil と statement class に付したｔは文のクラス名（C,A,I,D,P）で，この記号によって文を識別し，文のクラスを認識する。また，それ以降の探索で構文グラフとの不一致が発生しても他の文の枝へ探索を進めないようにした。

(4) call subgraph は定義済のサブグラフを呼出して探索することを表す。サブグラフは entry を入口とし，return を出口として記述する。entry に付けた名前を call subgraph で引用する。

(5) branch は左優先で分岐するためのもので，それ以降で構文の不一致が生じた時に残された他の枝へ探索を進める。分岐は最近通過した枝から用いる。

(6) link if はカウンタがリミット値ｎ以下なら１ずつ増しながら指定の接続先に飛び越すもので，このループは，文番号（５字以下），英字名（たとえば６字以下）および次元数（７次元以下）をチェックするために用いている。ｎは set limit で設定する。

(7) loop while はスタックの値ｓによってｓ回繰り返すことを表すもので，文字定数と書式の編集記述子に使用されるホラリス型の文字列ｎＨｈ1ｈ2……ｈｎの文字数ｎをカウントするために用いている。ｓには stack atom と stack end で挟まれた atom の列が表す数値を与える。

図２にＡＣＯＳ-６ ＦＯＲＴＲＡＮ77 8) の構文グラフの一部分を示す。参考のために対応するＪＩＳの構文図を一部添付した。メイングラフは入口から一続きになったグラフである。サブグラフとしてはつぎの３５個のものを用意した。テーブルはつぎの３種類のものである。

- テーブル

Ｓａ（英字）　Ｄｚ（数字）　Ｄｏ（８進数字）

- サブグラフ

Ｎａ（英字名）　Ｌｂ（文番号）
Ｃａ（定数）　Ｃｉ（符号なし定数）
Ｃｒ（実数,整数）　Ｃｃ（複素数）
Ｃｐ（符号なしの単/倍/４倍精度実定数，整定数）
Ｃｌ（論理定数）　Ｃｈ（文字定数）
Ｈｓ（'ｈ1ｈ2……ｈｎ' または "ｈ1ｈ2……ｈｎ"）
Ｃｏ（８進定数）　Ｃｂ（２進定数）
Ｃｅ（指数部）　Ｒｌ（関係演算子）
Ｒａ（先頭の・を除いた関係演算子）
Ｅｘ（式）　Ｅａ（算術式）
Ｅｌ（論理式）　Ｅｃ（文字式）
Ｐｃ（変数,配列,配列要素,関数の引用）
Ｓｄ（宣言子添字）　Ｓｖ（添字）
Ｌｅｎ（長さ指定）　Ｓｃ（文字位置）
Ｔｐ（型）　Ｉｖ（初期値のデータ並び）
Ｉｄｏ（DOの制御並び）　Ｄｄｏ（DATA文のDO型並び）
Ｌｉｏ（入出力並び）　Ｌｄｏ（入出力のDO型並び）
Ｆｓ（書式仕様）　Ｆｔ（書式識別子）
Ｉｃ（入出力文の制御並び）
Ｏｃ（ OPEN, CLOSE, BACKSPACE, REWIND, ENDFILE 文の制御情報並び）
Ｑｃ（ INQUIRE 文の制御情報並び）

３.２ グラフ記述言語ＧＲＡＤＬ

構文グラフを記述するためにグラフ記述言語ＧＲＡＤＬを設けた。ＧＲＡＤＬによる記述構造はつぎの形式で与える。

```
DEFINE SYNTAX GRAPH <グラフ名>;
  DEFINE SYMBOL;
    STRING=h1h2;
  END;
  DEFINE TABLE <テーブル名>;
    <atom の列>
  END;
    .....
  DEFINE MAINGRAPH;
    <element description>
      .....
  END;
  DEFINE SUBGRAPH <サブグラフ名>;
    <element description>
      .....
  END;
    .....
END SYNTAX GRAPH;
```

記述の冒頭で定義するグラフ名は，ＦＥＤＩＴがこの名前を用いて複数の構文グラフを実行時に動的に選択して扱えるようにするために設けたものである。

ＳＹＭＢＯＬの定義は atom の文字列を囲む左右の括弧を与えるものである。これは，対象とするＦＯＲＴＲＡＮ処理系の使用する文字集合が変更されても，

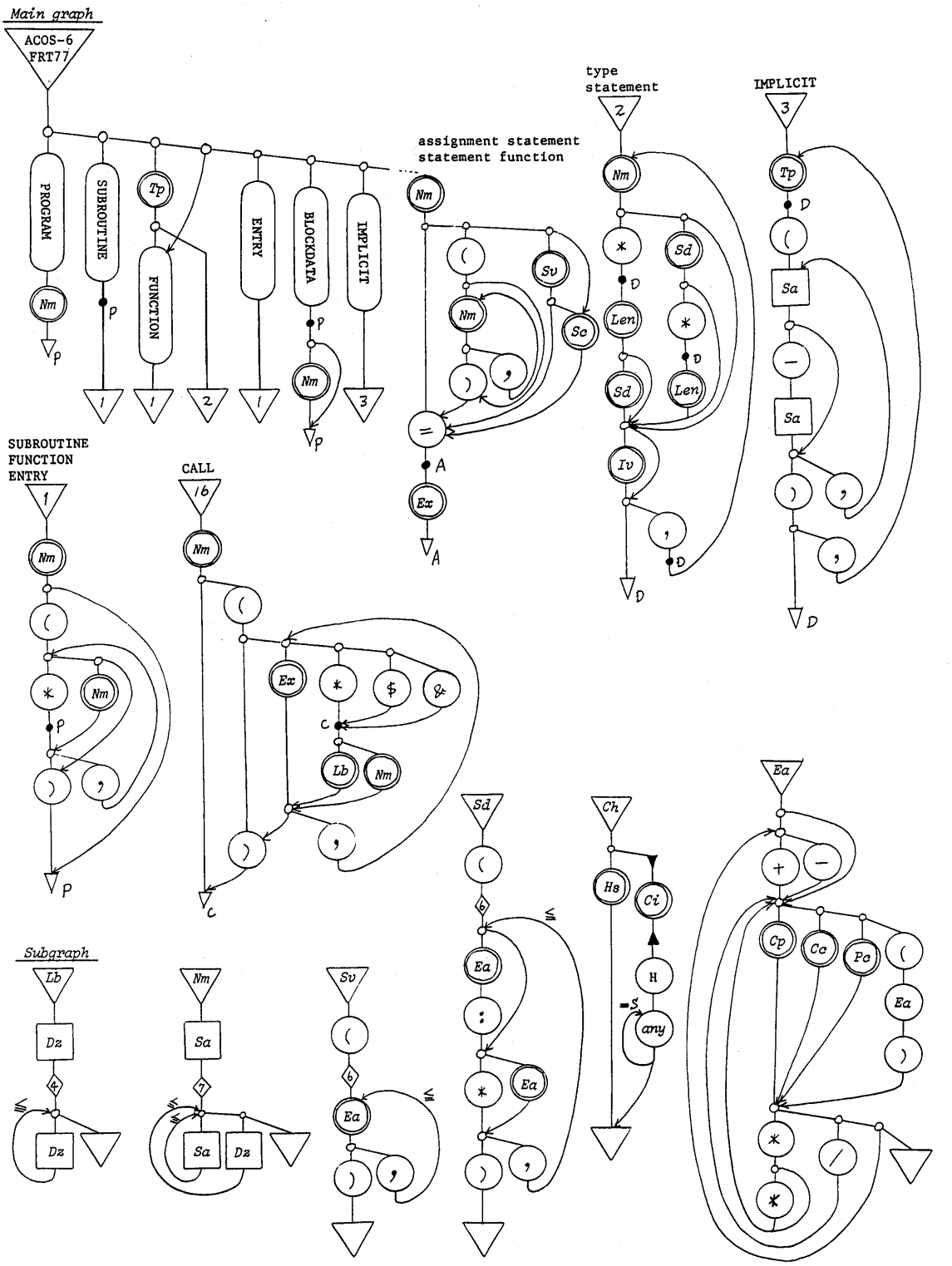

図2　構文グラフの例（ACOS-6 FORTRAN77）

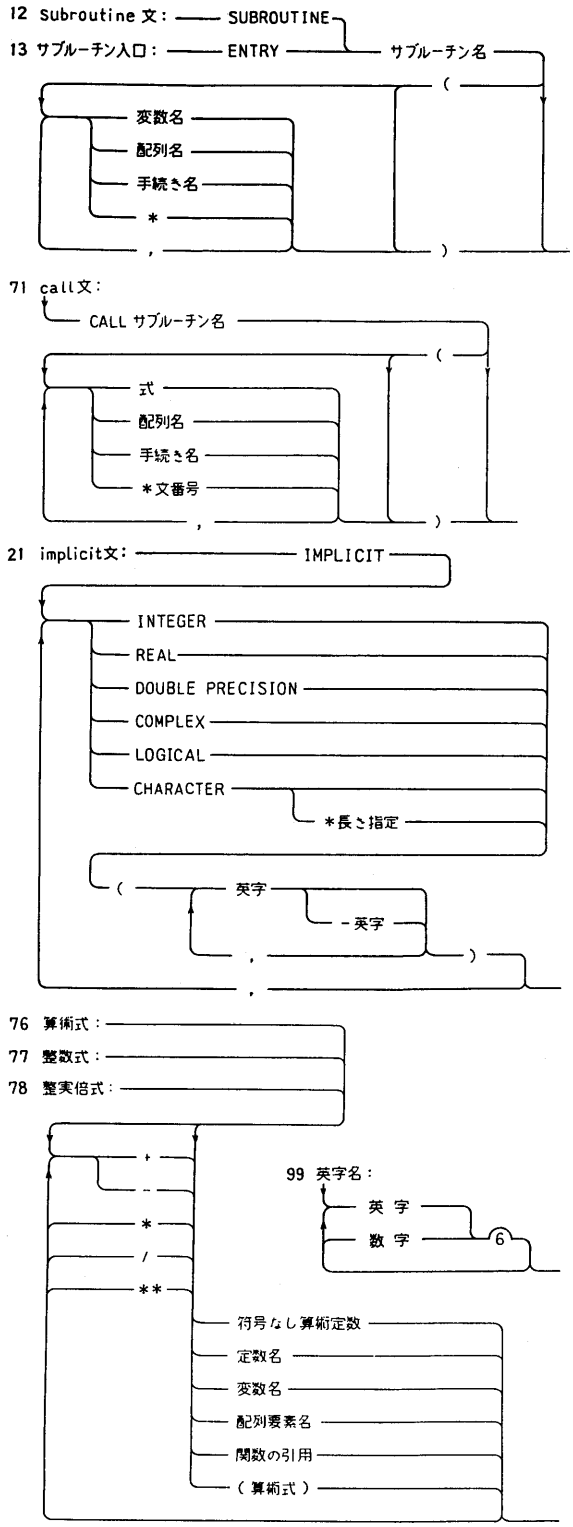

図2（つづき）　ＪＩＳの構文規則図の例

ＧＲＡＤＬ処理系に影響を及ぼすことがないようにするために設けたものである。

ＴＡＢＬＥの定義は look up で用いる文字集合のテーブルを定義するもので，必要なテーブルをここに並べる。

ＭＡＩＮＧＲＡＰＨの定義はメイングラフをＧＲＡＤＬで記述したものである。

ＳＵＢＧＲＡＰＨの定義はサブグラフをＧＲＡＤＬで記述したもので，必要なだけここに並べる。

メイングラフやサブグラフの中に書く <element description>は，つぎの形式のものとした。

＜label＞：＜statement＞

ここで，＜label＞は このステートメントに付けるラベルで８文字以内の英数字とし，メイングラフおよびサブグラフ毎に一意な名前を与える。＜statement＞には表２に示したＧＲＡＤＬのステートメントを構文グラフの記号に従って書く。ステートメントの中の記述事項の間は空白で区切り，ステートメントの終わりは；とする。ＡＴＯＭに書く「ｃ１ｃ２----------ｃｎ」は，文字ｃｉ（ｉ＝１，２，.....，ｎ）を定義済の括弧ｈ１とｈ２で囲んだもので，任意の文字「any」の場合には単に「」と書くことにした。

図３にＧＲＡＤＬによる構文グラフの記述例を示す。ＳＹＭＢＯＬの定義で，ＡＴＯＭの文字を囲む左右の括弧として@@を定義して用いている。

４．構文ファイルの生成と利用

４．１ 構文ファイル

ＧＲＡＤＬで記述した構文グラフの各要素は，ＧＲＡＤＬ処理系によってつぎの形式のセルに変換する。

［ｐ，ｑ］

ｐ：グラフ要素の種別

ｑ：つぎのようにｐに応じて設定されるオペランド

ATOM　：文字列へのポインタ
CALL　：サブグラフの入口のアドレス
LOOK　：テーブルの格納アドレス
BRANCH：右側への分岐先ラベルのアドレス
LINK　：接続先ラベルのアドレス
IFLINK：カウンタ番号と接続先ラベルのアドレス
LOOP　：スタック番号と対応するLENDのアドレス
LEND　：対応する LOOP のアドレス
STACK ：スタック番号
SET　：カウンタ番号と設定値ｎ
NIL　：文のクラスを表すコード
STCLASS：文のクラスを表すコード

```
DEFINE SYNTAX GRAPH  FRT77;
COMMENT SYNTAX GRAPH OF ACOS-6 FORTRAN77 BY GRADL;

DEFINE SYMBOL;  STRING=@@;  END;

DEFINE TABLE SA;
  ABCDEFGHIJKLMNOPQRSTUVWXYZ$abcdefghijklmnopqrstuvwxyz
  END;
DEFINE TABLE DZ;  0123456789  END;
DEFINE TABLE DO;  01234567  END;

DEFINE MAINGRAPH;
COMMENT PROGRAM;
 L01:  BRANCH L02;  ATOM @PROGRAM@;  CALL NM;  NIL P;
COMMENT SUBROUTINE;
 L02:  BRANCH L03;  ATOM @SUBROUTINE@;  STCLASS P;
       CALL NM;  BRANCH M014;  ATOM @(@;
 M011:  BRANCH M015;  ATOM @*@;
 M012:  BRANCH M016;
 M013:  ATOM @)@;
 M014:  NIL P;
 M015:  BRANCH M013;  CALL NM;  LINK M012;
 M016:  ATOM @,@;  LINK M011;
COMMENT FUNCTION OR TYPE STATEMENT;
 L03:  BRANCH L04;  CALL TP;  BRANCH M021;
 L031:  ATOM @FUNCTION@;  LINK M011;
COMMENT TYPE STATEMENT;
 M021:  CALL NM;
 M022:  BRANCH M026;  ATOM @*@;  STCLASS D;  CALL LEN;
        BRANCH M024;  CALL SD;
 M024:  BRANCH M025;  CALL IV;
 M025:  BRANCH M028;  NIL D;
 M026:  BRANCH M024;  CALL SD;  BRANCH M024;
        ATOM @*@;  STCLASS D;  CALL LEN;  LINK M024;
 M028:  ATOM @,@;  STCLASS D;  LINK M021;

COMMENT ASSIGNMENT STATEMENT OR STATEMENT FUNCTION;
       CALL NM;  BRANCH M212;
 M211:  ATOM @=@;  STCLASS A;  CALL EX;  NIL A;
 M212:  BRANCH M215;  ATOM @(@;  BRANCH M214;
 M213:  CALL NM;  BRANCH M217;
 M214:  ATOM @)@;  LINK M211;
 M215:  BRANCH M216;  CALL SV;  BRANCH M216;  LINK M211;
 M216:  CALL SC;  LINK M211;
 M217:  ATOM @,@;  LINK M213;
END;
DEFINE SUBGRAPH  LB;
      LOOK DZ;  SET #1=4;
 N1:  BRANCH N2;  LOOK DZ;  IFLINK #1 N1;
 N2:  RETURN;
END;
DEFINE SUBGRAPH  NM;
      LOOK SA;  SET #2=7;
 N1:  BRANCH N2;  LOOK SA;  IFLINK #2 N1;
 N2:  BRANCH N3;  LOOK DZ;  IFLINK #2 N1;
 N3:  RETURN;
END;
DEFINE SUBGRAPH  SD;
     ATOM @(@;  SET #3=6;
 N1:  BRANCH N2;  CALL EA;  ATOM @:@;
 N2:  BRANCH N4;  ATOM @*@;
 N3:  BRANCH N5;  ATOM @)@;  RETURN;
 N4:  CALL EA;  LINK N3;
 N5:  ATOM @,@;  IFLINK #3 N1;
END;
DEFINE SUBGRAPH  SV;
     ATOM @(@;  SET #4=6;
 N1:  CALL EA;
 N2:  BRANCH N3;  ATOM @)@;  RETURN;
 N3:  ATOM @,@;  IFLINK #4 N1;
END;
DEFINE SUBGRAPH  CH;
      BRANCH N1;  CALL HS;  RETURN;
 N1:  STACK #1;  CALL CI;  SEND;  ATOM @H@;
      LOOP #1;  ATOM @@;  LEND;  RETURN;
END;

END SYNTAX GRAPH;
```

図3　GRADLによる構文グラフの記述例
（図2のグラフを記述したもの）

変換後の構文データは，図4に示すようにグラフ領域とデータ領域に分かれた構文ファイルに格納する。前者にはメイングラフ，続いてサブグラフ群を定義順に格納した。後者にはＴＡＢＬＥに定義された文字集合を定義順に並べ，続いてＡＴＯＭに指定された文字列を格納し，グラフ領域の各要素からポインタで指して参照するようにした。

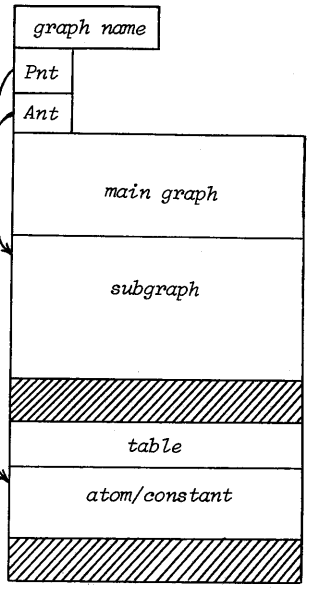

図4　構文ファイルの構成

4.2　構文解析

編集操作部はつぎの時点で文法処理部に依頼して構文解析を行う。

① 編集モジュールを抽出してエディットファイルが作成された時

② 挿入，置換コマンドが実行された時

③ ＦＥＤＩＴの中からＴＳＳの他のシステムを呼び出し，再びＦＥＤＩＴに戻った時（カレントファイルを経由してエディットファイルが変更されている可能性がある）

構文解析は，編集操作部が切出して渡す入力レコード（一つのＦＯＲＴＲＡＮの文）が構文ファイルに格納されている構文グラフ上のどの枝に過不足なく一致するかを調べることによって行う。この結果，グラフ上に正しく張り渡せる枝が見つかれば，文のクラスの識別およびその中に含まれる構文要素（英字名，文番号，文）の情報を抽出して編集処理部に返す。一致するものがなければエラーを表示し，編集操作部はユーザにこの文の修正を促す。

構文グラフの探索はつぎのように行う。

(1) メイングラフの入口から要素を順に辿り，各要素の指示する処理を遂行する。

(2) atom または look up に出会った時にソーステキストの文字列と照合する。ソーステキストの中の空白は，意味をもつ特別なものを除いてこの時無視する。

(3) 文字列の照合が一致すればグラフの前進方向の枝

を，一致しなければ後退方向の枝を選ぶものとし，これが branch，link，link if，loop while のいずれであるかによってグラフ上の探索経路を決定してつぎのノードを選ぶ。

(4) call subgraph によって指定のサブグラフを引用して探索し，return によって元のグラフに戻る。

(5) atom nil に到達し，入力レコードに残りの文字がなければ，ここで正常な文と確定する。残りの文字があれば，他の枝に探索を移すが，statement class の現われた場所より上方へは進まず，探索する枝がなくなったらエラーとして表示する。

サブグラフは再帰的に呼び出されるので，プッシュダウンスタックを用いて，call と return を管理した。また，グラフの分岐先アドレスの管理にもプッシュダウンスタックを用い，先入れ後出しによって分岐先を選ぶようにした。

構文要素の識別は該当するサブグラフの入口と出口で検出するようにした。すなわち，編集コマンドの要素属性ｘの英字名に対してはサブグラフＮｍで，文番号に対してはＬｂで検出している。また，文の名称とクラス名はグラフの探索時に statement class を通過した時点で設定する。したがって，ｘの指定項目を拡張するには検出するサブグラフを追加すればよい。

構文解析の結果をつぎのエントリをもつ３種類の構文要素テーブルとして作成し，編集時に用いている。

- ステートメントテーブル

  [V，ｒ＃，Ｌ＃，ｐ＃，ｎ＃]

- 英字名テーブルおよび文番号テーブル

  [V，ｒ＃，Ｌ＃，Ｃｆ，Ｃ１，ｐ＃，ｎ＃]

ここで，

V　：構文要素の値（文の名称，英字名，文番号）
ｒ＃：対応するエディットファイルのレコード番号
Ｌ＃：ソーステキストに付いている行番号
Ｃｆ：ソーステキスト中のこの要素の開始カラム
Ｃ１：ソーステキスト中のこの要素の終了カラム
ｐ＃：構文要素テーブル中の直前要素の位置
ｎ＃：構文要素テーブル中の直後要素の位置

である。ｐ＃とｎ＃によってＸ構文テーブルの要素間のリンクをとっている。

５．おわりに

実用性の最も高いプログラミング言語の一つであるＦＯＲＴＲＡＮについて，言語依存型の構造エディタＦＥＤＩＴを開発し，東北大学大型計算機センターの主システムであるＡＣＯＳ 1000 で利用可能にした。

ＦＥＤＩＴでは，構文データをプログラムから独立させ機械処理可能なグラフ表現を用いたために，つぎのような利点をもつシステムを構築することができた。

- 一般に実用システムでは頻繁に実施されるＦＯＲＴＲＡＮの言語仕様の追加に容易に対処できるシステムとなった。
- ユーザでも理解し易い図式表現であるため，フルセットで提供される構文データの中から，ユーザの状況に合わせたサブセットの構文データを作成してサービスすることが可能である。
- ＦＥＤＩＴのシステムプログラムから複数種類の構文データを実行時に選択できるので，対象言語の拡大が容易に行える。

ＦＥＤＩＴは単純な設計をとった上にＣ言語を用いてインプリメントしたので，機種の移行に容易に適応できるばかりでなく，今後の利用環境の中で重要な役割を担うワークステーションのソフトウェアにも適用可能である。また，簡便な構文解析の手法は，ＦＯＲＴＲＡＮだけでなく他の言語，特に第４世代言語の解析に有効である。

本開発では実現できなかったが，構文グラフを直接操作できるグラフエディタおよびグラフから内部形式に直接変換するシステムを開発すれば，ＧＲＡＤＬを用いる必要がなくなり一層便利なツールになる。これらは今後の課題として取り組みたいと考えている。

最後に，ＦＥＤＩＴは日本電気㈱ との共同開発により実現したものである。関係各位に感謝する。

【参　考　文　献】

1) 武田，益田，石川（編）：データベース特集号，情報処理，Vol.17，No.10 (1976).

2) 永田，黒川（編）：エディタ特集号，情報処理，Vol.25，No.8 (1984).

3) 松田孝子，嶋田理恵，嶋田真一郎，表　俊夫：ＦＯＲＴＲＡＮ構造エディタＦＥＤＩＴ，日本ソフトウェア科学会構造エディタに関するワークショップ (1986).

4) 松田孝子，嶋田理恵，嶋田真一郎，表 俊夫：ユーザに構文設定機能を提供するＦＯＲＴＲＡＮ構造エディタの設計と実現，情報処理学会論文誌（印刷中，Vol.28，No.5).

5) ＡＣＯＳ-6　ＦＯＲＴＲＡＮ構造エディタＦＥＤＩＴ使用法説明書，東北大学大型計算機センター (1985).

6) 電子計算機プログラム言語ＦＯＲＴＲＡＮ，JIS C 6201-1982，日本規格協会 (1982).

7) 松田孝子，高橋理：ミニコンによる高水準ＦＯＲＴＲＡＮの文法チェックの一手法，情報処理学会第１４回全国大会講演論文集 (1973).

8) ＡＣＯＳ　ＦＯＲＴＲＡＮ７７言語説明書，日本電気，AGB01-4 (1985).