プログラミング言語 17－1
（１９８８．７．１５）

集合指向言語ＳＯＬと その言語処理系について

吉見康一　重松保弘
（九州工業大学工学部）

吉田　将
（九州工業大学情報工学部）

アルゴリズムの記述には集合が利用されることが多い．しかし、従来の手続き型言語は、Pascalを除き、集合型のデータ構造を持たない．また、Pascalの集合型も、実際にアルゴリズムを記述するには制限が強すぎる．そこで、筆者らは、Pascalの集合型の機能を強化し、さらに写像の概念を導入したプログラミング言語ＳＯＬを設計し、その言語処理系を開発した．ＳＯＬの主な特徴は、集合と写像の動的な定義が可能であること、数学的な記法がそのまま使用できること、等である．本稿では、ＳＯＬの言語仕様およびその言語処理系について述べる．

Set Oriented Language SOL And Its Language Processor

Koichi YOSHIMI* Yasuhiro SHIGEMATSU* Sho YOSHIDA**
* Department of Computer Science, Faculty of Engineering, Kyusyu Institute of Technology, Kitakyusyu 804 Japan
** Department of Artificial Intelligence, Faculty of Computer Science and Systems Engineering, kyusyu Institute of Technology 820 Japan

Mathmatical notation of set theory is useful in describing computer algorithms. However, it is impossible, except Pascal, to use the notation in computer languages, at present. Even Pascal, there are strict restrictions in using the notation. Thus, we have been designed a new algorithm description language SOL(Set Oriented Language) and developed its language processor. SOL is based on set theory and predicate logic. Therefore, universal and existential quantifiers are introduced. Statements for dynamic definition of set data and mapping relation are also introduced. In this paper, features and the syntax of SOL and its compiler are described.

1．序

現在、我々が使っているデータは、なにかある物のうちの1つ　つまりある集合の1つの要素であることが多い．例えば、生徒の数学の成績の集まりも集合である．このような物の集まりは、集合の概念を持った言語であれば、非常に記述しやすい．論文等におけるアルゴリズムの記述にも、しばしば集合・写像など数学的な記述を用いたものが多数見受けられる．

現在、広く使用されている言語で、集合の概念を持っているのはPascalだけである[7]．Pascalは、データ構造が豊富であることからアルゴリズムの記述にもよく使われている．しかし、Pascalの集合型には制限が多く、アルゴリズムの記述に抽象的なデータ及びその集合が使われている場合には、その記述は困難である．制限としては、①集合の総要素数に制限がある，②集合の要素を直接読み書きできない，等があげられる．

そこで、この集合型の拡張・強化を行うと共に、写像と述語論理の記法を導入することにより、アルゴリズムの記述性の向上を狙いとして開発したのが、ＳＯＬ（Set Oriented Language）である[3][5][6]．以下に、ＳＯＬの主な特徴をあげる．

①Pascal風の手続き型言語であり、プログラムの構造、さまざまな定義の方法、変数のスコープ等は、Pascalと同じである．
②集合および写像の入出力が可能である．
③高階の集合（集合族）が取り扱える．
④集合演算子（∩，∪，－：集合間の積、和、差の演算子）が使用できる．また、集合用の関係演算子として、集合の等値関係（＝≠）、包含関係（⊂）、要素関係（∈∉）が使用できる．ＳＯＬでは、これら数学的記号がそのまま使用できる．
⑤論理式に、集合に関する全称論理記号（∀）と存在論理記号（∃）を使用することができる．
⑥動的な集合の生成が可能である．これには、外延記法と内包記法がある．
⑦集合間に1対1ないし多対1の写像を定義できる．（ＳＯＬでは集合族が扱えるので、像を集合とすることにより、実質的には1対多ないし多対多の対応関係が定義できる．
⑧写像の動的な再定義が可能である．

ＳＯＬと同じ数学的な概念と記述を取り入れた多目的なシステム記述手法として、ＶＤＭ（Vienna Development Method）が1970年代に、ＩＢＭ Vienna Research Lab.で開発されている[14]．このＶＤＭは、集合，写像，述語論理などの数学的な概念を用いてシステムの記述を行う．ただしＶＤＭは、その記述によるデータ構造，手続き，関数などを段階的に詳細化していき、最終的に実在する言語でシステムを動作させるという点でＳＯＬとは異なっている．

本論文では、ＳＯＬの言語仕様とその処理系について報告する．

2．ＳＯＬの言語仕様

ＳＯＬはPASCALを基本とし、その集合型について大幅な拡張を行っている．以下、拡張（及び変更）された部分を重点的に説明する．

2．1　データ型

データ型は大きく分けて、次の3種類に分けられる．
①基本型　②集合型　③添数付き集合型

①基本型
- 整数型（int）　：整数
- 実数型（real）　：実数
- 文字型（char）　：$00～$FFで表される1文字
- 文字列型（str）：文字の集まり（終端は$00）
- 論理型（bool）　：true,false
- 組型　（tuple）：
  いくつかのデータを組にして1つのデータとする（Pascalのレコード型から可変部を除いた物と同等、ただし要素として配列構造は不可）
- ファイル型（file）：ファイル識別子

②集合型
集合型は、次の定義によって再帰的に定義される．

　setof　基本型または集合型

このうち、”setof　集合型”は、集合族を定義する．例えば、”setof setof int”は、”整数の集合の集合”型を表す．

③添数付き集合型
添数付き集合型は、他の手続き型言語における配列と同様の型である．なお、添数集合は自然数である．添数付き集合型は、次の形式で定義される．

```
indexedset ［範囲］ of
                基本型または集合型
```

”範囲”は、添数の動く範囲を規定するものでる．例えば、１から５の範囲を動く１次元の添数なら”範囲”は”１～５”となるし、２次元で、１から５、１０から１５の範囲を動くなら、”範囲”は、”１～５，１０～１５”と記述する．

２．２　写像

ＳＯＬは、集合間に写像を宣言し定義することができる．集合ＸとＹの間にｆという写像を宣言したい場合は、map文によって次のように宣言する．

```
map f:X→Y
```

ｆの値を定義する方法には２種類ある．第一は代入によるものであり、第２は入力関数（readなど）によってファイルから入力するものである．次に代入の例を挙げる．

```
X←{1, 2};  Y←{"one", "two"};
f←{ (1,"one"), (2,"two") };
```

この例では、まず、集合ＸとＹの値を代入によって定義し、続いて写像ｆを代入によって定義している．この結果、図１に示す写像関係が定義される．

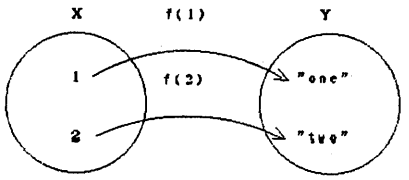

図１．写像の定義例

２．３　動的な集合の生成

ＳＯＬでは、集合をプログラムの実行時に動的に生成することができる．動的な生成の方法としては、外延的記法と内法的記法の２種類の記法がある．

外延的記法は、集合要素を列挙することによって集合を定義するものであり、次の形式をとる．

{ 式$_1$, 式$_2$, 式$_3$, ……, 式$_n$ }

この場合、式$_1$～式$_n$を計算し、その値を要素とする集合が生成される．

内包的記法は、集合の性質を論理式で表し、その論理式を満たすものによって集合を定義する．

{ 変数∈式 | 論理式 }

２．４　述語論理記法の導入

ＳＯＬでは、全称論理記号∀と存在論理記号∃を論理式において用いることができる．これらの論理記号を用いて以下の形式の作用素を導入した．

全称作用素：　∀（束縛変数∈集合式）
存在作用素：　∃（束縛変数∈集合式）

作用素をともなった論理式は、束縛変数によって束縛される．作用素を伴った論理式もまた論理式であるので、評価の結果、真または偽の論理値が返される．ただし、評価が確定した時点で、束縛変数には次のような値が残される．

全称作用素：論理式を偽とする要素の１つ．
存在作用素：論理式を真とする要素の１つ．

これらは、作用素を含む論理式の評価にともなう副作用である．この副作用は、集合要素の探索などの処理に応用することができる．
つぎに例を挙げる．

［例］集合classを学生番号（整数）の集合とし、集合pointを生徒の点数（整数）の集合とする．ｆをclassからpointへの写像とし、class,point,ｆの値が定義されているものとする．このとき、点数が５０未満の生徒の学生番号をclassから削除するには次の文を実行すればよい．

```
while ∃(x∈st)(f(x)<50) do
   st=st-{x}
od
```

２．５　文

ＳＯＬの特徴的な文を以下に説明する．

①forall文

```
forall  <束縛変数>∈<集合式>  do
   <文1>；<文2>…；<文n>
```

od

forall文は、＜集合式＞で得られる集合の要素を順に束縛変数に代入し、その束縛された文１～文ｎを要素の数だけ実行する．

②defmap文

defmap文は集合間に写像を動的に定義するための文である．defmap文の形式は、次のようになっている．

defmap　写像変数（＜式$_1$＞）＝＜式$_2$＞

＜式$_1$＞は定義域の要素、＜式$_2$＞は値域の要素でなければならない．このdefmap文によって、式$_1$の写像による像が式$_2$に再定義される．また、定義されている写像を取り消すには、空集合定数φを用いて次のように記述する．

defmap　写像変数（＜式＞）＝φ

２．６　標準手続きと標準関数

①集合用関数

・ｇｅｔｅｌ（集合変数）
引数から要素を１つ取り出して返す．その要素は、集合から取り除かれてしまう．（破壊的関数なので注意が必要）．

・ｃａｒｄ（集合変数）
引数の集合の要素数を返す．

②入出力関数

・ｅｏｆ（ファイル変数）
ファイルの終わりを調べる．

・ｅｏｌｎ（ファイル変数）
ファイルの行の終わりを調べる．

③入出力手続き

・ｒｅａｄ［ｌｎ］（ファイル変数，変数$_1$，変数$_2$，…，変数$_n$）
変数１から変数ｎに、ファイル変数で指定されたファイルから入力した値を渡す．変数には、写像も指定することができる．

・ｗｒｉｔｅ［ｌｎ］（ファイル変数，変数$_1$，変数$_2$，…，変数$_n$）
変数１から変数ｎを、ファイル変数で指定されたファイルに出力する．変数には、写像も指定することができる．

・ｏｐｅｎ（ファイル変数、ファイル名、入出力指定）
ファイル名で指定されたファイルをオープンする．

・ｃｌｏｓｅ（ファイル変数）
指令されたファイルをクローズする．

３　ＳＯＬの言語処理系

３．１　言語処理系と中間コード

ＳＯＬはPascal（正確には、Pascal-S）を基本とし、それをＬＬ（１）文法の形式を崩さず拡張している．したがって、ＳＯＬコンパイラはPascalと同じく再帰下降型のコンパイラとなっている．コンパイラは、ＳＯＬ言語で書かれたプログラムをＳＯＬ用の仮想機械の疑似コードであるＳコードに変換し、実行する．（Ｓコードは、Ｐコードを拡張したものである．）

プログラムは、日本語ワードプロセッサか、システム立ち上げ時に文字フォント登録済みの日本語フロントエンドプロセッサ（日本語ＦＥＰ）を組み込んだエディタを用いて作成する．ＳＯＬのプログラムの作成から実行までの手順を図２に示す．

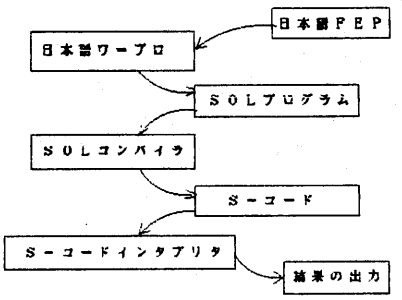

図２．ＳＯＬプログラムの作成から実行まで

Ｓコードとその意味を表１に示す．（拡張部分のみをあげる．）Ｓコードは、ＳＯＬの仮想スタックマシン上で実行される．それぞれのコードはスタックに置かれたデータを処理する．

表１．　Ｓコード

ｺｰﾄﾞ	意味
70	スタック上に積まれた２つの集合データの集合積をとる
71	スタック上に積まれた２つの集合データの集合和をとる
72	スタック上に積まれた２つの集合データの集合差をとる

73 集合と要素データ（スタック上に存在）について、その要素が集合に存在しているどうかをテストする
74 73の否定を返す
75 スタック上に積まれた２つの集合データ間に包含関係があるかどうかをテストする
76 スタック上に積まれた２つの集合データが等しいかどうかをテストする
87 76の否定を返す
77 スタックに積まれた要素を、要素１つの集合にして返す
78 ファイルから写像データを入力し、対応する集合間に写像を定義する
79 内包式のみに使用する集合和のコード
81 写像によって定義される像を取り出す
82 スタック上の集合データから要素を１つ取り出す．
83 ∃∀専用のデータ格納コード
スタック上にデータとアドレスを積み、データをそのアドレスに格納する
84 スタックからｎデータｐｏｐする
85 現在定義されている写像の値域の要素を再定義する
86 getel関数用コード
90 tupleのメンバのアクセス
指定された番号のデータを返す．
91 tupleのメンバのアドレスを返す．
92 指定されたtupleメンバのデータを書き換える
100 写像代入文に使用するコードで、リンク構造で作成している写像データを実際の写像構造に作り替える
207 tupleのデータが同じかどうかテストする
208 207の否定をとる

３．２　集合と写像のデータ構造

３．２．１　集合のデータ構造

集合を表現する方法には、大きく分けて、配列を使う方法とリンク構造を使う方法がある．配列を使う場合、要素の最大数が固定されてしまうという欠点があるが、要素のアクセスはリンク構造のそれよりも多少早い．リンク構造を使う場合、要素はメモリの許す限り取ることができるが、アクセスはポインタ変数があるぶんだけ遅くなり、また、リンクの構造上、ガベージコレクションが必要になる．

今回作成したコンパイラでは、集合データをより柔軟に扱えるという観点から、集合のデータ構造に、リンク構造を採用した．（組型および、写像のデータ構造についても同様．）集合の要素（セル）は、図３の構造をしている．

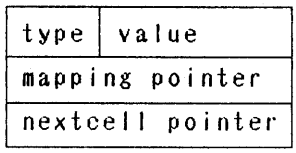

図３．　セル

以下、それぞれの欄を説明する．

・ｔｙｐｅ
要素の型を示すタイプタグである．要素の型には、整数型、実数型、文字型、文字列型、組型、集合型がある．

・ｖａｌｕｅ
要素の実際の値が入っている．
ｔｙｐｅで指定される型により、次のように値が入る．
整数型、文字型：値そのものが入る．
文字列型　　：文字列プールのインデックスが入る．
集合型、組型　：それぞれのデータを構成しているセルを指すポインタの値が入る．

・ｍａｐｐｉｎｇ　ｐｏｉｎｔｅｒ
集合間に写像を定義するときに使用されるポインタである．このポインタは、像を指すためのリンク構造化された先頭を指している．

・ｎｅｘｔ　ｃｅｌｌ　ｐｏｉｎｔｅｒ
このポインタをつなぐことによって、集合を構成している．
データ構造から見た最後のセルのこの欄には、０が入れられる．

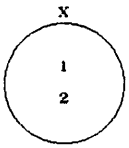

図４．集合Ｘ

図４に示す集合ＸはＳＯＬでは図５のような構造になる．

X←{ 1, 2}

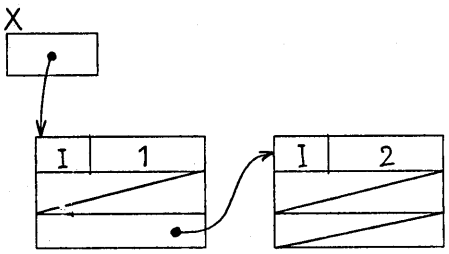

図５．集合Ｘの内部表現

３．２．２　写像のデータ構造

写像のデータ構造も集合と同じ構造のセルを用いて実現している．しかし、各フィールドの意味は異なっている．

・ｔｙｐｅ
　写像を表す定数が入れられる．

・ｖａｌｕｅ
　値域集合の要素を指すポインタである．

・ｍａｐｐｉｎｇ　ｐｏｉｎｔｅｒ
　同じ定義域集合上に、違う写像を定義する場合に使用されるポインタで、次の写像セルを指している．データ構造から見た最後のセルのこの欄には、０が入れられる．

・ｎｅｘｔ　ｃｅｌｌ　ｐｏｉｎｔｅｒ
　この欄には、写像の定義番号が入れられる．これにより、それぞれの写像を区別する．

以下に、例を示す．

図１のように、集合ＸとＹとの間に写像ｆが定義されているとすると、写像ｆの内部構造は、次図のようになる．

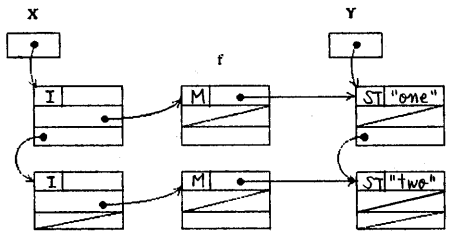

図６．写像ｆの内部構造

３．２．３　データのアクセス方法

集合・組　及び、写像のアクセスは、基本的にはリンクリストをたどることで行われる．集合の場合は、単にリンクリストをたどるだけである．

添数付き集合は、実際には配列とおなじ構造となる．例として、整数３つの要素を持つ添数付き集合は図７のようになる．

```
var x:indexedset [1..3] of int;
```

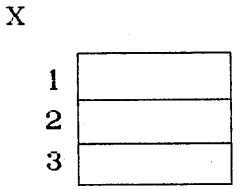

図７．添数付き集合の例

組データの場合、宣言されたデータの順番と作成された組データのセルの順番とを対応させることで、一致を取っている．つまり、欄$_n$ に対するデータは、リンクリストの先頭から数えてｎ番目のセルに存在することになる．整数と文字の組を要素とする集合の例をあげる．宣言は図８のようになる．

```
type tt=tupleof [a:int; b:char];
var x:setof tt;
```

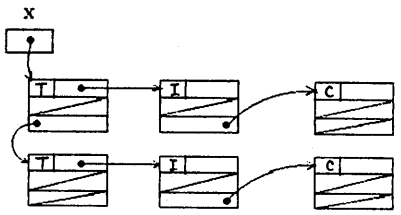

図８．組型の例

写像の場合は、写像セルの定義番号により対応する値域のデータをアクセスする．１つの集合に２つの写像が定義されている場合は、図９のようになる．

```
var X,Y,Z:setof int;
map f:X→Y; g:X→Z;
```

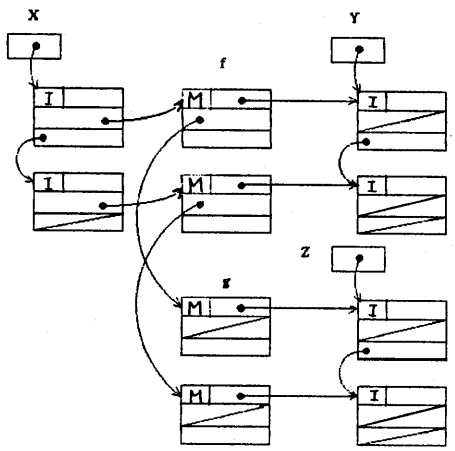

図９．写像の例

３．２．４　セルの管理

ＳＯＬで実際に使用しているセルは、整数型のポインタを使用した配列で作っている．これは、Pascalのnew, dispose等では、ガベージコレクション等のセルの自己管理が困難だからである．セルは次のように宣言されている．

```
type cell=record
            gb   : char;
            ctyp : char;
            val,mp,np : integer
          end;

var cb : array [1..cellmax] of cell;
```

ＳＯＬプログラムの起動時に、ｃｂ（セルの配列）をフリーリストとしてリンクしておき、これからセルを切り取って使用する．

ｇｂは、ガベージコレクション用の変数である．ガベージコレクションの方法としては、処理がしやすいこともあり、印付けの方法を用いている．

３．３　作用素のオブジェクト

ここでは、作用素を伴う論理式のオブジェクト生成について説明する．作用素の処理で使用されるＳコードのうち主なものは、82,83,84及び、jt（スタックトップが真のときジャンプ）、jf（偽のときジャンプ）、push等である．
作用素を伴う論理式の構文は、次のようになっている．

```
(∀ | ∃) '('<変数>' ∈' <集合式>')'
                '('<論理式>')'
```

上記の構文に対して生成されるコードは、一般に次のようになる．（∃の欄で空白の部分は、∀の欄と同じ）

		∀の場合	∃の場合
		<集合式>	
82	L1:	getatom L2	
		loadadr <変数>	
83		store2	
		<論理式>	
		jt L1	jf L1
84		pop 1	
		push FALSE	push TRUE
		jump L3	
84	L2:	pop 1	
		push TRUE	push FALSE
	L3:		

図１０．∀，∃のオブジェクトコード

まず、コード８２で、集合から要素を１つ取り出す．その要素をコード８３で変数に代入し、論理式の処理を行う．∀の場合、論理式の結果が真であれば、次の要素を処理するために、Ｌ１へジャンプする．偽であれば処理を中断し、論理値FALSEを返す．逆に∃の場合は、偽のときにジャンプする．真であれば処理を中断し、論理値TRUEを返す．

要素すべてについて演算し終わったらＬ２へジャンプする．∀の場合、要素すべてが条件にあったとして、論理値TRUEを返す．∃の場合、条件にあう要素が存在しなかったとして、論理値FALSEを返す．

4．まとめ

集合指向言語ＳＯＬとその言語処理系の開発を行った．ＳＯＬは、集合と写像が使えることからさまざまな分野への応用が考えられる．現在は、ＳＯＬで有向グラフの世界を記述しているが、今後はプログラムグラフの問題、そしてデータベースの処理などへの応用が考えられる．

現在、処理系はPascalで記述しており、行数は、3416行である．ＳＯＬの処理系は、NEC PC-9801では、MS-DOSのTurbo Pascal、及び、Apollo DOMAIN ワークステーションではDOMAIN PASCALで各々稼働している．現行の処理系には次のような問題が残っている．

①処理速度

現在の処理系は、Ｓコードというソル用の中間言語のインタプリタであるため、実行が遅い．

②ガベージコレクション

現在の処理系ではガベージコレクションをユーザに任せている．システムで管理するのがいちばん良い方法なのであるが、コンパイラであることがそれを難しくしている．集合に関する式を処理している場合、集合演算の途中結果はスタック上に積まれているが、システムにはそれがほんとうに集合データかどうかを判断できない．これはタグ付きのデータを使うことで解決できる．

③データ構造

現在の処理系ではリンク構造を用いているが、これを他の構造（配列，２分木，多分木等）に変えて処理系の評価を行う必要がある．

謝辞　ＳＯＬの言語仕様の検討と処理系の作成に協力いただいた九州工業大学情報工学科の與那覇誠君と橘亜由美さんに感謝します．また、御討論いただいた九州工業大学工学部の安在弘幸教授と山之上卓助手に感謝します．

参考文献

[1] Wirth,N.: Algorithms + Data Structure = Programs, Prentice-Hall,1976.

[2] Aho,A.V., Hopcroft,J.E., Ullman,J.D. : The Design and Analysis of Computer Algorithms, Addison-Wesley.

[3] 重松，吉見、吉田：集合向き言語ＳＯＬの言語仕様について，昭和６２年度電気関係学会九州支部連合大会論文集,pp.510.

[4] Cohen,J.:Garbage Collection of Linked Data Structure, Computing Surveys, Vol 13,No.3,pp341-367,September 1981.

[5] 與那覇：集合向き言語ＳＯＬの開発，九州工業大学，昭和６２年度　卒業論文.

[6] 重松，吉見，吉田：集合と写像に基づくアルゴリズム記述言語ＳＯＬの言語仕様について，昭和６３年度情報処理学会九州支部研究会報告，pp40-49.

[7] Jensen,K.,Wirth,N.:Pascal User Manual and Report 2nd edition, Springer-Verlag,1975.

[8] Lipschutz,S.,金井,他訳：集合論，マグロウヒル好学社,1982.

[9] 松坂：集合・位相入門，岩波書店,1985.

[10]Liu,C.L., 成嶋，他訳：組合せ構造とグラフ理論入門，マグロウヒル好学社,1978.

[11]Yonezawa,A.:A Method for Synthesizing Data Retrieving Programs,Jounal of Information Processing,Vol.5,No.2, pp.94-101,1982.

[12]前原：数学基礎論入門,朝倉書店,1977.

[13]Date,C.J.,藤原訳：データベース・システム概論,丸善,1984.

[14]Cohen,B.,Harwood,W.T.,Jackson,M.I.: The Specification of Complex Systems, Addison-Wesley, 1986.

付録　ＳＯＬの構文規則

以下の記法において〈、〉、{、}、[、]、=、| はメタ記号である。[ ]は、選択してもしなくてもよい。{ } は、１回選択する。[ ]* は、０回以上の繰り返し、[ ]+ は、１回以上の繰り返し、を各々意味する。

〈プログラム〉= proc〈プログラム名〉;〈ブロック〉.

〈ブロック〉= [define [〈変数名〉=〈定数〉;]+]
[type [〈型名〉=〈型〉]+]
[var [〈変数名〉[,〈変数名〉]* :〈型〉;]+]
[map [〈写像名〉[,〈写像名〉]* :〈変数〉→〈変数〉;]+]
[[proc〈手続き名〉[(〈引き数〉)] :〈型〉;〈ブロック〉;]* |
[func〈関数名〉[(〈引き数〉)] :〈型〉;〈ブロック〉;]*]*
begin〈文〉[;〈文〉]* end

〈型〉= [indexedset [〈定数〉～〈定数〉[,〈定数〉～〈定数〉]*] of]
[setof]*
〈基本型〉

〈基本型〉= int | real | char | str | bool | file | tupleof (〈欄の並び〉)

〈欄の並び〉=〈名前〉[,〈名前〉]* :〈型〉[;〈名前〉[,〈名前〉]* :〈型〉]*

〈引き数〉= [var]〈変数〉[,〈変数〉]* :〈型〉
[; [var]〈変数〉[,〈変数〉]* :〈型〉]*

〈文〉= {〈変数〉|〈関数名〉|〈写像名〉}←〈式〉|
〈手続き名〉[(〈式〉[,〈式〉]*)] |
begin〈文〉[;〈文〉]* end |
case〈式〉of〈定数〉[,〈定数〉]* :〈文〉
[〈定数〉[,〈定数〉]* :〈文〉]* esac |
for〈変数〉←〈式〉to〈式〉do〈文〉[;〈文〉]* od |
forall〈束縛式〉do〈文〉[;〈文〉]* od |
break |
repeat〈文〉[;〈文〉]* until〈論理式〉|
if〈論理式〉then〈文〉[;〈文〉]* [else〈文〉[;〈文〉]*] fi |
while〈論理式〉do〈文〉[;〈文〉]* od |
defmap〈写像名〉(〈式〉)=〈式〉

〈論理式〉=〈式〉|{ ∀ | ∃ }(〈束縛式〉)(〈論理式〉)

〈束縛式〉=〈変数〉∈〈式〉

〈式〉＝〈単純式〉［｛＞｜＜｜＝｜≠｜≧｜≦｜∈｜∉｜⊂｝〈単純式〉］

〈単純式〉＝［＋｜－］〈項〉［｛＋｜－｜∪｜or｝〈項〉］*

〈項〉＝〈因子〉［｛＊｜／｜∩｜and｜div｜mod｝〈因子〉］

〈因子〉＝〈変数〉｜〈内包集合式〉｜〈外延集合式〉｜
〈符号のない定数〉｜
〈関数名〉［（〈式〉［，〈式〉］）］｜
〈写像名〉（〈式〉）｜
（〈式〉）｜
not〈因子〉

〈変数〉＝〈変数名〉［［〈式〉［，〈式〉］*］｜．〈欄の名〉］*

〈内包集合式〉＝｛〈束縛式〉｜〈論理式〉｝

〈外延集合式〉＝｛〈式〉［，〈式〉］*｝

〈定数〉＝［＋｜－］｛〈符号のない定数〉｜〈定数名〉｝

〈符号のない定数〉＝〈定数名〉｜〈符号のない数〉｜”〈文字〉+ ”｜
［〈定数〉［，〈定数〉］*］｜
｛〈定数〉［，〈定数〉］*｝｜Φ｜
｛［（〈定数〉，〈定数〉）］+｝｜
true｜false

〈符号のない数〉＝〈符号のない整数〉
［［．〈符号のない整数〉］e［＋｜－］〈符号のない整数〉］

〈符号のない整数〉＝〈数字〉+

〈型名〉,〈定数名〉,〈変数名〉,〈関数名〉,
〈写像名〉,〈欄の名〉,〈手続き名〉,〈プログラム名〉＝〈英字〉［〈英字〉｜〈数字〉］*