集合指向言語 SOL とその言語処理系の開発†

重松保弘†† 吉見康一†† 吉田 将†††

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，さらに写像の概念と述語論理の記法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，述語論理式が使用できること，集合族や添数付き集合が使用できること，数学的な記法がそのまま使用できること，などである．SOL の言語処理系はコンパイラ・インタプリタ方式として設計され，移植性を考慮して PASCAL で記述されている．言語処理系では，集合や写像はリンク構造のセルで表現され処理される．コンパイラは SOL プログラムを仮想スタックマシンの中間コードであるSコードに変換し，インタプリタでシミュレートする．本稿では，SOL の言語仕様，言語処理系およびフローグラフへの応用例について述べる．

1. まえがき

アルゴリズムは，当初（特に FORTRAN が普及した時期には），主として文章や流れ図で記述されていた．しかし，構造化プログラミング言語 PASCAL が普及し始めてからは，アルゴリズムの検証が容易なこともあり，アルゴリズムの記述に PASCAL の記法が使用されることが多くなった．一方，計算機の利用分野の拡大に伴って，アルゴリズムの記述対象も，数値という具体的なデータのみならず，有向グラフにおける節や矢のような，より抽象的なものとしてのデータにまで拡張されるようになった．特に，データ構造の分野では，アルゴリズムの記述に抽象的なデータとその集合の記法が用いられることが多い．

アルゴリズムの記述対象が数値などの具体的なデータであれば，そのアルゴリズムをプログラムとして直接実行することもできる．しかし，その対象が抽象的なデータとその集合である場合，これをプログラムとして直接実行することはできない．それは，アルゴリズム記述言語としての PASCAL の集合記述能力が弱いことに1つの原因がある．実際，PASCAL では，集合データは，あらかじめ要素を列挙するか部分範囲の指定により確定しておかなければならず，取り扱える要素数にも厳しい制限 (PASCAL 6000-3.4 の場合，58 個[1]) がある．また，PASCAL には集合演算子や関係演算子が用意されているが，集合間の要素の対応関係を定義し，参照するような手段は用意されていない．もし，この手段があれば，データ構造の表現や操作にポインタ等を用いなくてもよい場合がでてくるので，アルゴリズムの記述性が大幅に向上することが期待できる．

そこで，筆者らは，PASCAL の集合型の機能を大幅に拡張し，さらに写像の概念を導入することによって，新たにアルゴリズム記述のためのプログラミング言語 SOL (Set Oriented Language) を設計し，その言語処理系を開発した[10),11)]．

SOL の主な特徴を以下に列挙する．

（1） PASCAL 風な手続き型言語である．したがって，プログラムの構造，手続きや関数の定義，定数，型および変数の定義，変数の有効範囲などは PASCAL と同じである．ただし，制御文についてはなるべく複合文を使わなくてよいよう構文規則を変更した．

（2） 集合および写像（後述）の入出力が可能である．この機能により，集合要素をあらかじめプログラム中で定義しておく必要がなくなる．

（3） 集合族が取り扱える．

（4） 集合演算子（∩，∪，－：集合間の積，和，差を作る演算子）が使用できる．また関係演算子として，集合の等値関係（＝，≠），包含関係（⊂），集合と要素の関係（∈，∉）が使用できる．PASCAL では，これらの演算子を数値演算子などで代用しているが，SOL では解読性を重視し，数学上の記法をそのまま

† Design and Development of A Set Oriented Language SOL and Its Language Processor by YASUHIRO SHIGEMATSU, KOICHI YOSHIMI (Department of Electrical, Electronic and Computer Engineering, Faculty of Engineering, Kyushu Institute of Technology) and SHO YOSHIDA (Department of Artificial Intelligence, Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology).

†† 九州工業大学工学部電気工学科
††† 九州工業大学情報工学部知能情報工学科

採用している．

（5） 論理式に全称論理記号（∀）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，外延記法によるものと内包記法によるものがある．

（7） 集合間に1対1ないし多対1の写像を定義できる．ただし，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） 写像の動的な変更が可能である．

集合を扱う言語としては，ニューヨーク大学でSETL[6]が開発され，日本ではLorel-2[7]が開発されている．SETLは有限集合と組（tuple）を基本とした集合指向言語である．また，Lorel-2も有限集合と組を基本としているが，集合は実質的には線形リストで表現されている．ただし，SETLもLorel-2も，述語論理の記法や写像の記法は導入されていないので，アルゴリズムの記述性には問題が残っている．

SOLと類似した言語に，IBMで開発されたVDM仕様記述言語（VDM Specification Language）がある[8]．VDM（Vienna Development Method）は，集合，写像，述語論理に基づくVDM仕様記述言語を用いて複雑なシステム（complex system）を記述し，記述された手続き，関数，データ構造等を人手によって段階的に詳細化し，最終的には実在するプログラミング言語でシステムを実現しようとする手法である．したがって，VDM仕様記述言語とSOLとはその開発目的が基本的に異なっている．

本論文では，SOLの主な言語仕様，言語処理系の概要，およびSOLの応用例について述べる．また，SOLの構文規則を付録に示す．

2. SOLの主な言語仕様

2.1 データ型

SOLで扱うデータ型は大別すると3種類に分かれる．すなわち① 基本型，② 集合型，③ 添数付き集合型，である．

基本型は，さらに7種類に分けられる．すなわち，整数型，実数型，文字型，文字列型，論理型，ファイル型，組型である．このうち組型はいくつかのデータを組にして1つのデータとするもので，PASCALのレコード型から可変部を除いたものと同等である．

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．たとえば，“**setof setof int**”は“整数の集合の集合”型を表す．

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

ここで，“範囲”は添数の動く範囲を規定するものであり，PASCALの配列の添字の記法に類似した記法を採用している．集合を要素とする添数付き集合は，自然数を添数集合とする集合族を構成する．

2.2 動的な集合の生成

SOLでは集合をプログラム実行時に動的に生成することができる．これには2種類の記法がある．第1は外延による記法であり，第2は内包による記法である．

外延による記法は，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 n}

この場合，式1～式 n は計算され，その値を要素とする濃度 n の集合が生成される．外延による記法において，式がすべて定数である形式は，集合の入出力におけるデータの記法である．

内包による記法は，集合の性質を論理式で表し，その論理式を満足するものによって集合を定義するものであり，次の形式をとる．

{変数∈式|論理式}

この形式の意味は次のようになる．すなわち，“式”が表す集合の要素である“変数”について，“論理式”の値が真になる場合を探し，“論理式”を真とする“変数”の値から成る集合を生成する．次に例を挙げる．

【例】 X を整数の集合型変数とし，i を整数型変数とする．X の値が {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12} のとき，$\{i \in X | i \bmod 3 = 1\}$ の返す値は {1, 4, 7, 10} となる．ただし，**mod** は剰余演算子である．

内包記法に相当するものにZF（Zermelo-Frenkel）式と呼ばれる表記法がある．この記法は，関数型言語のKRC[9]やMiranda[9]に導入されている．ただし，これらの言語は実質的にはリストである多重集合しか表現できないので，ZF式はリスト処理の一手段として導入されているにすぎない．

2.3 集合と写像の宣言と定義

PASCAL では，集合型を定義する場合，その要素のとりうる範囲を基底型として，要素の列挙または部分範囲の指定によってあらかじめ宣言しておかなければならない．列挙された要素は順序を持つので，PASCAL で取り扱う対象は順序集合に限定される．SOL でも，集合型変数は，その要素の型を宣言するが，要素の取り得る範囲を宣言する必要はない．また，要素に順序がついている必要はない．

次に例を示す．

var X: **setof int**;

この例では，集合型変数 X が宣言され，その要素は任意の整数である（整数の取り得る範囲は処理系によって制限される）．なお，変数 X の値は，宣言時には不定である．X に整数の集合を代入する例を次に示す．

$X \leftarrow \{1, 2, 3, 4, 5\}$;

この代入文によって X の値が動的に定義されることになる．

集合変数が変数宣言されると，SOL では集合変数間に写像を宣言することができる．次に例を挙げる．

var X: **setof int**;
Y: **setof str**;
map f: $X \rightarrow Y$;

この例では，X（ないし，その部分集合）を定義域（始集合），Y を終集合とする写像 f が宣言されたことになる．ただし，この時点では，X, Y および f は未定義である．X と Y を**図1**のように定義し，X と Y の要素間に図1のような写像 f を定義したいとする．X と Y および f を代入によって定義する例を次に示す．

$X \leftarrow \{5, 7, 10\}$;
$Y \leftarrow$ {"one", "five", "seven", "ten", "eleven"};
$f \leftarrow \{(5, \text{"five"}), (7, \text{"seven"}), (10, \text{"ten"})\}$;

この例では，終集合 Y が値域（$f(X)$）と同一でないので“中への写像（into-mapping）”になっている．

集合が動的に定義できたように，SOL では写像も動的に定義できる．このために写像定義文（defmap 文）が使用される．次に例を挙げる．

$X \leftarrow X \cup \{1\}$;
defmap $f(1) =$ "one";

この例では，1行目の文で f の定義域 X に1を追加し，2行目の文で，X の元1の f による像 $f(1)$ を“one”に定義している．

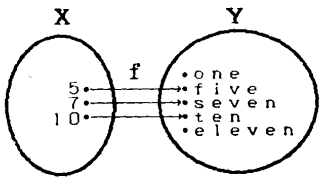


図 1 集合と写像の例
Fig. 1 An example of set and mapping.

また，10 と “ten” の写像関係を取り消すには，空集合定数 Φ を用いて，次の文を実行すればよい．この文の実行後，$f(10)$ は Φ を返すことになる．

defmap $f(10) = \Phi$;

SOL では，Φ は，空集合または値が未定義であることを示す．

2.4 作 用 素

集合の性質を調べるために，述語論理の作用素の概念を導入することは有益であると考えられる．そこで，SOL では，全称論理記号 $\forall$ と存在論理記号 $\exists$ を導入し，これを用いて以下の形式の作用素を許すことにした．

全称作用素の形式：$\forall(x \in X)$
存在作用素の形式：$\exists(x \in X)$

これらの作用素の記法において，X は x の変域を規定する集合である．これらの作用素は次の形式で再帰的に論理式を構成する．

$\forall(x \in X)$（論理式）
$\exists(x \in X)$（論理式）

この場合，論理式は，x によって束縛される．

作用素を伴う論理式もまた論理式なので，評価されて真または偽の値を返す．ただし，評価の結果が確定した時点で，束縛変数には次のような値が残る．

（1） 全称作用素の場合：論理式を偽とする値の1つ
（2） 存在作用素の場合：論理式を真とする値の1つ

これは論理式の評価に伴う副作用である．この副作用は集合要素の探索などに用いることができる．次に例を挙げる．

if $\exists(x \in X)(x < 5)$ **then** write$(x, f(x))$ **fi**;

ここで，X と Y を整数の集合とし，f を X から Y への写像とする．するとこの文によって，SOL は X の要素に5より小さいものがあるかどうかを調べ，あればその値 x と Y の要素 $f(x)$ を出力する．

作用素と類似した動作をする文に forall 文がある．次に例を示す．

forall $x \in X$ **do** write($f(x)$) **od**;

この例では，X のすべての要素 x に対して像 $f(x)$ が出力される．

3. SOL の言語処理系

3.1 言語処理系と中間コード

SOL の言語処理系はコンパイラ・インタプリタ方式になっている．SOL の構文規則は，付録に示すように PASCAL（正確には PASCAL-S[5]）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 のコンパイラは，PASCAL と同じく再帰下降型コンパイラである．コンパイラは，SOL 言語で書かれたソースプログラムを，仮想スタックマシンの中間コード（S コードと呼ぶ）に変換する．仮想スタックマシンのシミュレータは，S コードのインタプリタとして動作し，S コードを実行する．なお，S コードは，PASCAL の P コード[5]を SOL 用に拡張したものである．P コードにない S コードの主なものとその意味を**表 1** に示す．

SOL 言語は ASCII コードに規定されていない文字を使用している．そこで，SOL プログラムの入力には日本語ワードプロセッサか日本語エディタを使用する．

3.2 集合と写像のデータ構造

PASCAL のように普遍集合があらかじめ定まっており，しかも，その要素数が少ない場合には，ビットマトリクスを用いて効率良く集合を表現できる．しかし，SOL のように普遍集合が定まっていない場合には，リンク構造が適していると考えられる[7]．そこで，集合や写像を表現するためのデータ構造としてリンク構造を採用することにした．リンク構造を構成する要素をセルと呼ぶ．セルは，**図 2** に示すように 4 つの欄から構成される．SOL のすべてのデータは，各々 1 つのセルで代表される．セルの各欄は，データの場合と写像の場合では異なった役割を持っている．

データの場合，各欄は以下の意味を持つ．

（1） type 欄

要素の型を示すタイプタグが入る．要素の型には，

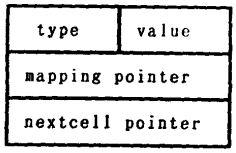


図 2 セルの構造
Fig. 2 The structure of cell.

表 1 S コードの拡張部
Table 1 Extension part of S-code.

コード番号	コードの意味
70	スタック上の 2 つの集合の集合積をスタックに積む．
71	スタック上の 2 つの集合の集合和をスタックに積む．
72	スタック上の 2 つの集合の集合差をスタックに積む．
73	スタック上の集合と要素について，要素が集合に含まれていれば真を，さもなければ偽をスタックに積む．
74	73 の否定をスタックに積む．
75	スタック上の 2 つの集合間に包含関係があれば真を，さもなければ偽をスタックに積む．
76	スタック上の 2 つの集合が等しければ真を，さもなければ偽をスタックに積む．
77	スタック上の要素を，その要素 1 つの集合にしてスタックに積む．
78	ファイルから写像を入力し，指定された集合間に写像を定義する．
79	内包記法の論理式を真とする束縛変数値を集合に加える．
81	スタック上の原像とパラメータの写像番号をもとに像を求めスタックに積む．
82	スタック上の集合から要素を 1 つ取り出し，スタックに積む．
83	束縛変数に値を格納するために使用される．
84	スタックからパラメータの個数だけ pop する．
85	スタック上の集合等とパラメータの写像番号から写像を動的に定義する．
86	スタック上の集合から要素を 1 つ取り出してスタックに積む．この要素は集合から取り除かれる．
87	76 の否定をスタックに積む．
91	tuple 型のメンバの番地をスタックに積む．
100	代入によって与えられた写像データから写像の構造を作る．
207	スタック上の tuple 型のデータが等しければ真を，さもなければ偽をスタックに積む．
208	207 の否定をスタックに積む．

整数型，実数型，文字型，文字列型，組型，集合型がある．

（2） value 欄

要素の値が入っている．type 欄の型により，次のように値が入る．整数型，実数型および文字型では，値そのものが入る．文字列型では，文字列プールのインデックスが入る．集合型と組型では，それぞれのデータを構成しているセルへのポインタが入る．

（3） mapping pointer 欄

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．写像のリンク構造を指している．

（4） next cell pointer 欄

集合を構成するポインタである．

写像の場合，各欄は以下の意味を持つ．

(1) type 欄

"写像" を示す定数が入る．

(2) value 欄

値域の要素を指すポインタである．

(3) mapping pointer 欄

同じ定義域集合上に，異なる写像を定義する場合に使用されるポインタであり，次の写像セルを指している．

(4) next cell pointer 欄

写像の識別番号が入る．

例として，以下のように集合と写像が宣言，定義されているとする．このとき，**図3**のようなデータ構造が構成される．(図3中の int, str, char, map, f, g は実際には整数値に置換される．)

```
var   X: setof int;
      Y: setof str;
      Z: setof char;
map   f: X→Y;
      g: X→Z;
X←{1, 2, 3};
Y←{"one", "two"};
Z←{"a", "b"};
f←{(1, "one"), (2, "two")};
g←{(1, "a"), (3, "b")};
```

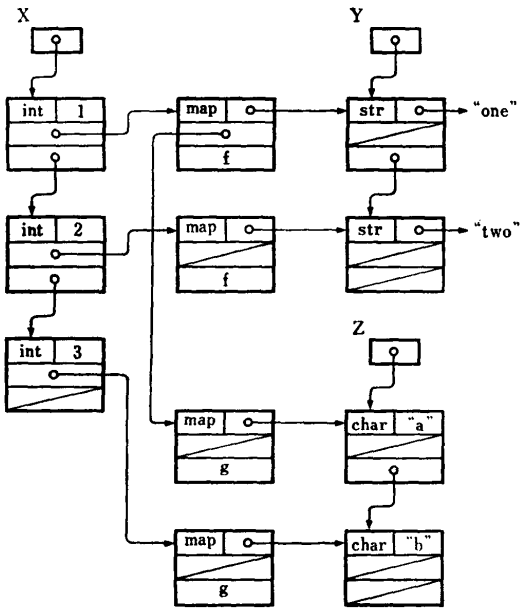


図 3 集合と写像のデータ構造例
Fig. 3 An example of data structure representation of set and mapping.

3.3 セルの管理

SOL の言語処理系では，各セルは，整数型のポインタを用いたレコード型として以下のように宣言されている．

```
type  cell=record
            gb: char;
          ctype: char;
          val, mp, np: integer
      end;
var cb: array [1.. cellmax] of cell;
```

ここで変数 *cb* はセルの配列であり，システム起動時にはすべて自由リスト (free list) としてリンクされており，必要に応じて使用される．また，cell の *gb* 欄は，ちり集め (garbage collection) のために使用する (図2からは除外してある)．セルをこのように配列で静的に用意した理由は，PASCAL のようにヒープ領域を用意し，これを標準手続き new と dispose で動的に管理する方法をとると，ちり集め等のセル管理が困難になるからである．

3.4 作用素の目的コード

ここでは，作用素を伴う論理式の目的コードの生成方法について述べる．作用素を伴う論理式は，次のような構文規則になっている．

∀(〈変数〉∈〈集合式〉)(〈論理式〉)

∃(〈変数〉∈〈集合式〉)(〈論理式〉)

この構文に対して生成される目的コードは，一般に**図4**のようになる．

図4では，まず，Sコード82で集合から要素を1つ取り出しスタッフに積む (もし，取り出すべき要素がなければ処理は終わりなので L2 に飛ぶ)．その要素をSコード83で〈変数〉に代入し，〈論理式〉の処理

全称作用素の目的コード		存在作用素の目的コード	
〈集合式〉の目的コード		〈集合式〉の目的コード	
L1: getatom(82)	L2	L1: getatom(82)	L2
loadadr(0)	〈変数〉	loadadr(0)	〈変数〉
store2(83)		store2(83)	
〈論理式〉の目的コード		〈論理式〉の目的コード	
jt(12)	L1	jf(11)	L1
pop(84)	1	pop(84)	1
push(24)	FALSE	push(24)	TRUE
jump(10)	L3	jump(10)	L3
L2: pop(84)	1	L2: pop(84)	1
push(24)	TRUE	push(24)	FALSE
L3:		L3:	

図 4 作用素の目的コード．( )内はSコード番号を示す．
Fig. 4 Object codes of quantifier forms.

を行う．全称作用素の場合，〈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に飛び，偽であれば処理を中断し論理値 FALSE をスタックに積む．存在作用素の場合は，この逆の処理を行う．L2 以降では，全称作用素の場合，要素のすべてが〈論理式〉を満足したとして TRUE をスタックに積み，存在作用素の場合，〈論理式〉を満足する要素が存在しなかったとして，FALSE をスタックに積む．

4. SOL の応用例

本章では，SOL のフローグラフ (flow graph) への応用例について述べる．フローグラフは，プログラムを制御の流れに着目してグラフ化した制御フローグラフ (control flow graph) の抽象モデルである[2]．フローグラフ G は節 (node) 集合 N，矢 (arc) 集合 A および入口節 (initial node) s の3つ組 (N, A, s) で表現される．ここで，N と A は1つの有向グラフを表し，s $(\in N)$ はその有向グラフの唯一の入口節である．**図5**にフローグラフの例を示す．ここで，$N=\{1, 2, 3, 4\}$，$A=\{(1,2), (2,3), (3,2), (2,4)\}$ および $s=1$ である．

矢 $(i,j)\in A$ が存在するとき，i は j の直接先行節 (immediate predecessor)，j は i の直接後行節 (immediate successor) と呼ばれる．節 i から節 j へ矢をたどって到達できるとき，その途中の節の列は経路 (path) と呼ばれる．経路の最初の節と最後の節が同じ物であるとき，その経路は閉路 (circuit) と呼ばれる．$(i,j)\in A$ なる節 i の数は，節 j の入次数 (incoming degree) と呼ばれる．

有向木 (directed tree) とは，閉路を持たない有向グラフである．入次数0の節が1つで，他のすべての節の入次数が1の有向木は根付き木 (rooted tree) と呼ばれる．各辺が正整数でラベル付けされている根付き木は，順序木 (ordered tree) と呼ばれる．

フローグラフ G の DFST (Depth First Spanning Tree) とは，G のすべての節を含む順序木であり，次のアルゴリズムで生成される[2]．

[rPOSTORDER 計算アルゴリズム]

入力：フローグラフ $G=(N, A, s)$ である．N の濃度を n とすると，G の節には，最初，1から n までの任意の番号が与えられている．

出力：配列 rPOSTORDER に格納された節の新しい番号付け．この番号付けは，G を深さ優先で探索した時に最後に通った節の順序を逆にしたもの (reverse postorder) である．

方法：最初，すべての節に "unvisited" とマークする．広域的整数型配列 rPOSTORDER [1: n] を用意する．広域的整数型変数 i を用意し，初期値を n とする．**図6**のアルゴリズム DFS(s)[2] を実行する．

図8は，SOL で記述した rPOSTORDER 計算プログラムとその実行結果の例である．このプログラムでは，**図7**(a)のフローグラフを例として使用した．フローグラフにおいて，N, A および s は，図8のプログラム中では集合型変数 node，写像 succ および整数型変数 s で各々表現されている．succ は node から nodes への写像であり，succ(n) は節 n の直接後行節集合を返す．このプログラムの特徴は，動的に集合 tnode，tnodes および写像 tsucc を生成している点で

```
recursive procedure DFS(integer x)
  Mark x "visited".
  while SUC(x)≠∅ do
    Select and delete a node y from SUC(x).
    if y is marked "unvisited" then
      Add arc (x,y) to the DFST diagram.
      call DFS(y)
    endif
  endwhile
  rPOSTORDER[x] := i
  i := i-1
return
```

図6 DFS アルゴリズム
Fig. 6 The DFS algorithm.

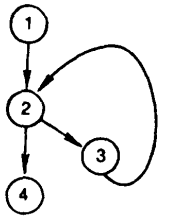
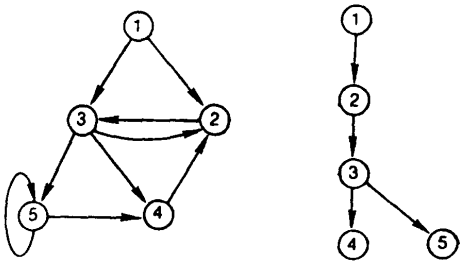


図5 フローグラフの例
Fig. 5 An example of flow graph.



(a) フローグラフの例 (b) (a) の DFST の例
(a) An example of flow graph. (b) A DFST of (a).

図7 フローグラフとその DFST の例
Fig. 7 An example flow graph and its DFST.

ある（これらは DFST を表現している）．このために defmap 文が使用されている．実行結果から，図7(b)の DFST が生成されていることがわかる．

5. あとがき

本論文では，集合指向言語 SOL の言語仕様，その言語処理系の概要および SOL のフローグラフへの応用例について述べた．SOL の応用例からもわかるように，フローグラフのアルゴリズムは，その解読性 (readability) を損なわない程度に SOL プログラムとして記述できる．これは，SOL が集合，写像および述語論理の記法を基礎としていることに起因する．SOL が有向グラフのアルゴリズムを記述するのに適していることはすでに確認しており[10]，今後，SOL をプログラムグラフのデータフロー解析に応用できれば，プログラムの最適化処理系を SOL を用いて開発することも可能になると考えられる．

SOL の言語処理系は，現在，PASCAL で記述されており（約 3500 行），apollo 社の DN 4000，DN 3000 ワークステーションと NEC の PC-9801 シリーズのパソコン上で稼働している．図8のプログラムと実行結果は apollo 社の DN 4000 によって出力されたものである．また，図8のプログラム行の先頭の整数は，Sコードの番地を示している．

現在，Sコード・インタプリタでは，ちり集めを行っていない．このため，ちり集め用の手続きを用意し，ユーザにちり集めを任せている．これは，セルにデータ種別を示すタグがないことに起因するので，今後，セルにタグ付きデータ構造を採用すればインタプリタによるちり集めも可能になると思われる．

謝辞 SOL の言語仕様の検討に協力していただいた九州工業大学工学部の與那覇誠君と橘亜由美さんに感謝します．また，貴重なコメントをいただいた東京工業大学工学部の徳田雄洋助教授，および，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．

```
 0 proc rpostorder; /* reverse-postorder computation
 0                     by depth-first search */
 0
 0 define max = 5;
 0
 0 type ints = setof int;
 0
 0 var node, tnode, unvisited : ints;
 0     nodes, tnodes : setof ints;
 0     s, i : int;
 0     rpostorder : indexedset [1~max] of int;
 0
 0 map succ : node -> nodes;
 0     tsucc : tnode -> tnodes;
 0
 0   proc dfs( x : int );
 0   var suc : ints;
 0       y : int;
 0   begin
 0     unvisited ← unvisited - { x };
 6     suc ← succ( x );
12     while suc≠φ do
16       y ← getel( suc );
20       if y∈unvisited then
25         defmap tsucc( x )=tsucc(x) ∪ { y };
38         tnode ← tnode ∪ { y };
44         dfs( y )
46       fi
48     od;
49     rpostorder[ x ] ← i;
54     i ← i - 1
57   end;
59
59 begin
60   node ← { 1, 2, 3, 4, 5 };
63   nodes ← { { 2, 3 }, { 3 }, { 2, 4, 5 }, { 2 }, { 4, 5 } };
66   succ ← { ( 1, { 2, 3 } ), ( 2, { 3 } ), ( 3, { 2, 4, 5 } ),
69            ( 4, { 2 } ), ( 5, { 4, 5 } ) };
71   s ← 1;
74   i ← max;
77   unvisited ← node;
80   tnode ← tnode ∪ { s };
86   dfs( s );
89   writeln("tnode = ", tnode);
94   writeln("tnodes = ", tnodes);
99   write("tsucc = ");
101  forall i∈tnode do
105    if tsucc(i)≠φ then
112      write(i,"→", tsucc(i))
121    fi
121  od;
123  writeln;
124  for i←1 to max do
128    writeln("rpostorder[", i,"]=",rpostorder[i])
139  od
141 end.
```

```
tnode = {    1    2    3    4    5 }
tnodes = { { 2 } { 3 } { 2  4  5 } { 2 } { 4  5 } }
tsucc =      1→{  2 }      2→{  3 }      3→{  4  5 }
rpostorder[  1]=  1
rpostorder[  2]=  2
rpostorder[  3]=  3
rpostorder[  4]=  5
rpostorder[  5]=  4
```

図 8 rPOSTORDER 計算プログラムと実行結果
Fig. 8 rPOSTORDER computation program written in SOL and its execution result.

参考文献

1) Jensen, K. and Wirth, N.: *Pascal User Manual and Report 2nd edition*, p. 167, Springer-Verlag, New York (1975).
2) Hecht, M. S.: *Flow Analysis of Computer Programs*, p. 232, North-Holland, New York (1977).
3) Lin, C. L., 成嶋ほか訳：組合せ構造とグラフ理論，p. 342，マグロウヒル好学社，東京 (1978).
4) Lipschutz, S., 金井ほか訳：集合論，p. 281，マグロウヒル好学社，東京 (1982).
5) Berry, P. E., 竹市訳：プログラム言語の処理系，p. 211，近代科学社，東京 (1983).

6) Schwartz, J. T.: Automatic Data Structure Choice in a Language of Very High Level, *Comm. ACM*, Vol. 18, No. 12, pp. 722-728 (1975).
7) 榎本 (進), 宮地, 片山, 榎本 (肇): Lorel-2 言語について, 情報処理, Vol. 19, No. 6, pp. 522-530 (1978).
8) Cohen, B., Harwood, W. T. and Jackson, M. I.: *The Specification of Complex Systems*, p. 139, Addison-Wesley Pub., England (1986).
9) 井田哲雄: プログラミング言語の新潮流, p. 262, 共立出版, 東京 (1988).
10) 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資料, pp. 40-49 (1988).
11) 吉見, 重松, 吉田: 集合指向言語 SOL とその言語処理系について, 情報処理学会プログラミング言語研究会資料, 17-1 (1988).

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〉 )
                        [ , ( 〈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〉 ]*
```

(昭和63年 7 月 25 日受付)
(昭和63年12月 12 日採録)

重松 保弘（正会員）

昭和22年生．昭和45年九州工業大学工学部電子工学科卒業．昭和47年九州工業大学工学部情報工学科助手．昭和61年同講師．現在，同電気工学科助教授．工学博士．マイクロプログラミング，計算機ネットワーク，プログラミング言語などの研究に従事．著書「基礎 BASIC プログラミング」，「ネットワークアーキテクチャの基礎」，「CASL によるアセンブリ言語入門」．訳書「ネットワークと分散処理」．電子情報通信学会会員．

吉見 康一（正会員）

昭和40年生．昭和60年徳山工業高等専門学校情報電子工学科卒業．現在，九州工業大学大学院修士課程2年在学中．言語処理系，並列計算機等の研究に興味を持つ．

吉田 将（正会員）

昭和8年生．昭和33年九州工業大学電気工学科卒業．昭和35年九州大学大学院工学研究科修士課程修了．工学博士．昭和37年九州大学工学部講師，その後，九州工業大学教授，九州大学工学部教授を経て，昭和61年10月九州工業大学情報工学部長．この間，九州工業大学および九州大学情報処理教育センタ長，九州大学大型計算機センタ長を歴任．自然言語処理の研究に従事．人工知能学会，日本認知科学会，米国 ACL などの会員．