テクニカルノート

不定型とラベル付き写像記法の導入による集合指向言語SOLの仕様拡張とその評価

迫江義彦† 重松保弘†

本論文では，集合指向言語SOLの2つの機能拡張について述べるとともに，従来のSOLとの記述能力の比較評価を行う．機能拡張の1つは任意のデータ型変数を扱える不定型の導入である．もう1つは写像の動的な生成，削除を可能にするラベル付き写像記法の導入である．ラベル付き写像記法の導入により応用プログラム実行時における写像の生成，削除が，プログラムの変更，再コンパイルなしに実現できるようになった．これらの機能拡張による記述能力の改善を，意味ネットワークの例題を用いて確認した．

An Extension of the Set Oriented Language SOL by Introducing Indefinite Type and Labeled Map Notations

YOSHIHIKO SAKOE† and YASUHIRO SHIGEMATSU†

In this paper, we refer to two extensions of the set oriented language SOL, that is the introduction of indefinite type and labeled map notations. The indefinite type releases users from strongly typed programming. The labeled map notation enables users to write application programs including dynamic construction and elimination of mapping relations. In this paper, we also show, how the improvement of description ability and efficiency is obtained by these extensions, using an example of a semantic network.

1. まえがき

アルゴリズムの自然なプログラム化とプロトタイプシステム開発の効率化を目指して，筆者らの研究室ではPASCALの集合型の機能を強化し，写像・対応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を開発，拡張してきた[1)~3)]．SOLはデータベース言語SQLのホスト言語への応用[2)]やフローグラフのインターバル解析アルゴリズムの記述[3)]などに有効であることを確認したが，同時にいくつかの問題点が明らかになった．

SOLはPASCALの強い型付けを受け継いでいる．強い型付けはプログラムのエラー検出には有効という利点もあるが，集合の要素の型に制限を加えたくないような応用プログラムの記述には適していないという欠点もある．そこで，任意の型のデータを値とできる新たな型として，不定型を追加することにした．

これまでのSOL（旧SOL）では，写像はプログラム作成時に静的に宣言しなければならなかった．そのため，プログラム実行中に使用する写像名，対応名が分からないような応用プログラムの記述が不可能であった．そこで，これを可能にするために，写像・対応の動的な生成記法（ラベル付き写像と呼ぶ）を導入した．これは写像名，対応名とラベル変数（定数）の組合せで写像・対応を表現しようとするものであり，ラベル変数（定数）を変更することによって実質的に異なる写像・対応を取り扱えるようになる．また，ラベル変数に新たな値を代入するだけで，動的に新たな写像・対応を生成することができる．

なお，集合指向言語としてはNewYork大学で開発されたSETL[4)]，Clark-son大学で開発されたISETL[5)]があるが，構文として写像の動的生成機能を持っているものはない．

本稿では，拡張したSOL（拡張SOL）の言語仕様と特徴について述べるとともに，意味ネットワークの記述を応用例として示し，拡張言語仕様の有効性を評価する．

2. SOLの拡張仕様

2.1 不定型

不定型はSOLで定義されるファイル型を除くすべ

† 九州工業大学
Kyusyu Institute of Technology

ての型（整数，実数，文字，文字列，論理，集合，組，添数付き集合）の値をその値として保持できる基本型の1つである．不定型の宣言例を次に示す．

（宣言例） **var** x : anytype;

ここで“anytype”は不定型を示す型名である．この例ではxを不定型変数として宣言している．また，要素の型が不定であるような集合（不定型の集合）を扱う応用プログラムを記述したいときは，次のように変数宣言しておくと便利である．

（宣言例） **var** Y : setof anytype;

不定型の導入にあたって，型不一致にともなう実行時エラーを防止する目的で，値の型の検査を行うための関数を用意した．また，不定型の集合から指定した型の要素を取り出すための関数も用意した．

2.2 ラベル付き写像

2.2.1 ラベル付き写像の宣言

ラベル付き写像（対応）は次の形式で宣言する．

map 写像名 { ラベル集合変数 } :
始集合変数 → 終集合変数;

なお，ラベル付き写像を宣言する場合は，あらかじめ始集合変数，終集合変数，ラベル集合変数を宣言しておく必要がある．次に簡単な例を示す．

var D,R : **setof** integer; /* 始集合，終集合 */
lab : **setof** string; /* ラベル集合 */
map $f\{lab\}$: $D \rightarrow R$; /* ラベル付き写像 */

ラベル集合は通常の集合として用いることができるので，ラベル集合それ自身を始集合あるいは終集合とするような写像を使用することも可能である．ただし，1つのラベル集合を2つ以上の写像のラベル集合として同時に使用することはできない☆．なお，旧SOLでは，写像名に“*”を付けたものを対応名と呼んでいたが，これに準じて，ラベル付き写像名の後に“*”を付したものをラベル付き対応名と呼ぶ．

2.2.2 ラベル付き写像の定義

ラベル付き写像（対応）の定義は，**defmap**文，代入文，入力手続き（read, readln）のいずれかを用いて行う．以下に例を示す．

```
(1) defmap f{"日本語"}(1) = "一";
    defmap f{"英語"}*(1) = {"one","first"};
(2) f{"日本語"} ← {[1,"一"],[2,"二"]};
    f{"英語"}* ← {[1,{"one","first"}],
                  [2,{"two","second","twice"}]};
```

(1)は**defmap**文を用いてラベル付き写像（対応）を定義する例である．(1)の1行目の文は，始集合の要素1のラベル付き写像$f\{$”日本語”$\}$による像$f\{$”日本語”$\}(1)$を“一”と定義している．また2行目の文は要素1のラベル付き対応$f\{$”英語”$\}*$による像$f\{$”英語”$\}*(1)$を“one”と“first”に定義している．(2)は代入文を用いて定義する例である．1行目はラベル付き写像$f\{$”日本語”$\}$として1と“一”，2と“二”の写像を定義する．2行目はラベル付き対応$f\{$”英語”$\}*$として1と“one”，“first”，2と“two”，“second”，“twice”の対応を定義するものである．(2)の2つの代入文を実行した結果，得られる写像・対応関係を**図1**に示す．

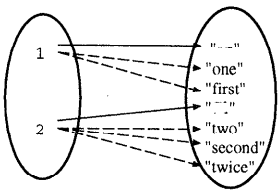


図1 ラベル付き写像，対応関係の例
Fig. 1 An example of labeled mapping and correspondence relations.

2.2.3 ラベル付き写像の変更

すでに定義されたラベル付き写像（対応）の変更には，(1) **defmap**文による再定義，(2) 代入文による再定義，(3) 入力関数による再定義，(4) 始集合または終集合の再定義，(5) ラベル集合の再定義，(6) **addmap**文による写像の追加，(7) **delmap**文による写像の削除の7つの方法がある．

このうち(5)の方法では，ラベル集合からあるラベルを削除することで，そのラベルに関するラベル付き写像（対応）全体を削除できる．たとえば，図1のラベル付き写像（対応）のラベル集合からラベル”日本語”を削除すると$f\{$”日本語”$\}$の写像がすべて削除される．

3. 意味ネットワークを例とする仕様評価

意味ネットワークでは，知識は対象，概念，状態を表現する節点（node）と，節点間の関係を表す枝（link）から構成される．**図2**は動物の分類に関する意味ネットワークの表現例である．

3.1 写像を用いた表現能力の比較

意味ネットワークは主体データから対象データへのラベル（関係名）付きのリンクであるので，写像・対応を使って効率的に表現できると考えられる．ここでは，旧SOLと拡張SOLを用いて応用プログラムを作

☆ 旧SOL処理系との互換性を保ち，かつ処理系を簡素化するための制約である．

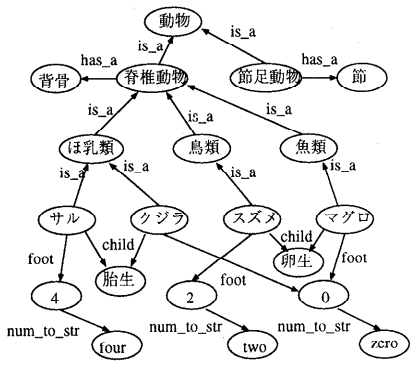


図2 意味ネットワークの例
Fig. 2 An example of semantic network.

成し，比較する．応用プログラムでは，操作プログラムを解読性良く記述するために，主体集合と対象集合を異なる変数として宣言する．また，例題意味ネットワークにおける関係名はプログラム中では写像名（対応名）として宣言し，使用する．

旧SOLでは不定型が使用できないため，**図3**(a)に示すように，主体や対象のデータ型（文字列型と整数型）の数だけ集合変数を宣言しなければならない（ここで $S1$，$S2$ は主体集合，$O1$，$O2$ は対象集合，M は多値の関係名集合を表す）．また，旧SOLでは写像（対応）を静的に宣言する必要があるため，関係名が多数存在する場合はすべて列挙しなければならない．しかも写像（対応）の宣言はその始集合と終集合の要素型の組合せごとに行う必要がある．この例から推察できるように，主体データや対象データの型や関係名がさらに増加した場合に，宣言量が膨大になると考えられる．また，この宣言方法では，関係名を示す写像が静的に宣言されるために関係の追加や削除ができない．

これに対し，拡張SOLでは不定型が使用できるので主体集合，対象集合の要素型の制限を受けない．図3(b)の例では，不定型の集合として，主体集合 S，対象集合 O の2つを宣言しており，これであらゆる型の主体，対象データが扱える．また，例題意味ネットワークを表現するために，ラベル集合 *label* とラベル付き写像 $f\{label\}$ を宣言している．このラベル付き写像の始集合 S と終集合 O は不定型の集合なので，図3(a)のように始集合と終集合の要素の型の組合せごとに写像を分ける必要がない．また，関係名はラベル付き写像のラベル（ラベル集合の要素）で表すので，図3(a)のように関係名ごとに写像を逐一宣言する必要もない．

図4 は，意味ネットワークに知識として入力された主体，対象間に関係を作る手続きを拡張SOLで記述したものである．この手続きで実行時に主体 (*sub*)，

```
var S1,O1,M : setof string;
    S2,O2 : setof integer;
map is_a,has_a,child : S1 → O1;
    foot : S1 → O2;
    num_to_str : S2 → O1;
```
(a) 旧SOLでの宣言部

```
var S,O,M,label : setof anytype;
map f{label}: S → O;
```
(b) 拡張SOLでの宣言部

図3 応用プログラムの宣言部
Fig. 3 The declaration part of an application program written in SOL.

```
procedure knowledge_input;
var sub,obj,l : anytype;
    ans : char;
begin
  readln(sub); readln(l); readln(obj);
  if l ∉ label then write("multi relation ? (y/n)");
    readln(ans);
    if ans ='y' then M← M∪{l}; fi;
  fi;
  if l∈M then addmap f{l}*(sub)={obj};
         else defmap f{l}*(sub)={obj};
  fi;
end;
```

図4 意味ネットワークのデータの追加手続き
Fig. 4 An example of data addition procedure of a semantic network written in SOL.

対象 (*obj*)，および関係名 (*l*) を入力し，続いて *l* が未登録の関係なら *l* が多値かどうか調べ，多値なら M に登録する．その後，*l* が多値関係なら **addmap** 文で，*l* が一価の関係なら **defmap** 文でそれぞれ動的に関係を生成する（*sub*，*obj*，*l* の値は自動的に S，O，*label* に各々，追加される）．旧SOLでは写像の動的生成が不可能なため，これと等価な機能を持つプログラムは作成できない．

3.2 三つ組を用いた表現方法との比較

意味ネットワークは，関係名，主体名，対象名の組合せの集合で表現することもできる．したがって，旧SOL，拡張SOLのいずれの場合も，意味ネットワークに関する応用プログラムを [関係名, 主体名, 対象名] の三つ組を用いて記述する方法が考えられる．この方法を用いると，関係の動的な変更を行う応用プログラムの作成が可能となる．

旧SOLでは，不定型が使用できないため，三つ組の各欄をある1つの型に固定しなければならない．したがって1つの欄が複数の型をとる可能性がある場合には，かなり複雑なプログラムになることが予想される．

三つ組集合を用いた例題意味ネットワークを表現する拡張SOLプログラムの宣言部の例を次に示す．

```
type triple =
```

```
procedure relation_delete;
var t : triple;
begin
   forall t ∈ K do
      if t.relation="foot" then K←K−{t}; fi;
   od;
end;
```
(a) 三つ組表現による手続き

```
procedure relation_delete;
begin
   label←label-{"foot"};
end;
```
(b) ラベル付き写像表現による手続き

図5 関係名 "foot" の削除手続き
Fig. 5 An example of elimination proccdurc of a rclation.

```
   tupleof[relation,subject,object : anytype];
var K : setof triple;
```

この三つ組宣言の変数Kに意味ネットワークを表す三つ組集合が与えられたとする．この状態で意味ネットワークを変更する例について検討してみる．

図5は関係名 "foot" を例題意味ネットワークから削除する手続き例である．(a) は三つ組表現を用いた手続きである．ここでは，関係名が "foot" である三つ組を検索し，その三つ組を削除している．そのために三つ組集合 K から要素を取り出す変数（ここでは t）の宣言と検索処理（**forall** ループ）が必要となる．

これに対して，(b) はラベル付き写像を用いた手続きであり，より簡単なプログラムになる．ここでは，関係名をラベルとしているので，ラベル集合 *label* から関係名 "foot" を削除することで関係名の削除が行える．(a) と比較して，(b) では関係名 "foot" を意味ネットワークから削除するということが直感的に理解できるという点で，記述性に優れている．

また，手続きに要する計算時間を考えると，(a) は集合からすべての三つ組を取り出し逐次検索しなければならないので不利である．現在のSOL処理系では，(a) の実行にはSコード[1]の総ステップ数で312ステップ，(b) の実行には13ステップを要する．したがって，(a) は (b) よりかなり処理時間がかかることが予想される．実際，現在のSOLの処理系では，図2の例をデータとしてプログラム化したケースでは，測定上 (a) は (b) の約57倍の計算時間がかかることを確認した．

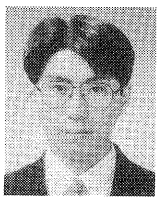
4. あとがき

集合指向言語SOLに不定型とラベル付き写像（対応）の記法を導入し，その言語処理系を開発した．不定型の導入によって，より柔軟なプログラムの作成が可能になった．また，ラベル付き写像記法の導入によって，これまで不可能であった写像の動的生成，削除が可能となり，応用プログラムの記述性を大幅に改善することができた．応用例でも示したように，意味ネットワークを動的に変更するような応用プログラムを記述するためには，ラベル付き写像記法はきわめて有効であることが分かる．このことから，一般にラベル付き有向グラフの形式を持つデータ構造を利用する応用プログラムにおいては，ラベル付き写像記法が効果的に利用できるものと考えられる．また，ラベル付き写像は関数の集合すなわち関数族を表現していると考えることもできる．図3(b)の例では，f は，*label* を添数集合とする関数の族と見なすことができる．この観点からの言語仕様の検討は今後の課題である．

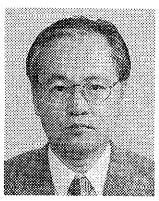
参考文献

1) 重松保弘，吉見康一，吉田 将：集合指向言語SOLとその言語処理系の開発，情報処理学会論文誌，Vol.30, No.3, pp.357–365 (1989).
2) 重松保弘，與那覇誠，吉田 将：集合指向言語SOLのデータベースへの応用，情報処理学会論文誌，Vol.33, No.8, pp.1041–1051 (1992).
3) 重松保弘，吉田 将：集合指向言語SOLの拡張とフローグラフのインターバル解析への応用，情報処理学会論文誌，Vol.34, No.2, pp.229–238 (1993).
4) Schwartz, J.T., Dewar, R.B.K., Dubinsky, E. and Schonberg, E.: Programming with Sets, An Introduction to SETL, p.493, Springer-Verlag, NewYork (1986).
5) Baxter, N., Dubinsky, E. and Levin, G.: Learning Discrete Mathematics with ISETL, p.416, Springer-Verlag (1989).

(平成9年2月3日受付)
(平成9年6月3日採録)



迫江 義彦（正会員）
昭和47年生．平成7年九州工業大学工学部電気工学科卒業．平成9年同大学大学院工学研究科博士前期課程修了．同年富士通（株）入社．言語処理系の研究に興味を持つ．



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．工学博士．九州工業大学大学院工学研究科教授．プログラミング言語，計算機ネットワークなどの研究に従事．