記　号　処　理 53－1
（１９８９．１１．２２）

引数のない論理プログラム表現

東京農工大学工学部
電子情報工学科

小谷　善行

一般の利用者にとって使い勝手のよい論理型プログラム言語の言語仕様を検討している。Ｐｒｏｌｏｇの場合、引数にある多数の変数は、数学的素養のない人にとってはなかなか扱い難い。ここではまったく引数のない論理型プログラム言語を提起し、さらにその日本語による表現形式を設計した。プログラムの要素となるものは述語である。述語は入力と出力の二つのポートを持つ。述語の間の接続関係はメタ述語で規定される。

A Logic Program Description without Arguments

Tokyo University of Agriculture and Technology
Dept. of Electronic and Information Science
Computer Science Course

Yoshiyuki KOTANI

A logic programming language specification is discussed, which is expected to be friendly for general users. It may be difficult for them to deal with many variables occurred in arguments of Prolog predicates, unless they have mathematical common sense. In this paper we propose A logic programming language in which each predicate has no arguments and design a Japanese-language oriented style.

Programs written in the language consists of predicates, which have a pair of ports, input and output. The relation between predicates is described by meta-predicates.

１．はじめに

Ｐｒｏｌｏｇ言語を勉強し始めると、たいてい、ｍｅｍｂｅｒ述語の定義：

```
member(X,[X|L]).
member(X,[Y|L]):-member(X,L).
```

に出会う。われわれは、ｍｅｍｂｅｒの定義を日本語で説明するときどのようにするだろうか。普通つぎようにいうだろう。

「リストの先頭はそのメンバーである。
それから、
リストの先頭以外のメンバーであるものも元のリストのメンバーである。」

しかし、

「ＸはＸとＬのリストのメンバーである。
それから、
ＸがＬのメンバーなら、ＸはＹとＬでできたリストのメンバーである。」

などとはあまりいうことはない。すなわち、ＸとかＹとかの変数より、「リストの先頭」などといって、既存の概念の結合で説明することが多い。

この現象は、日本語や英語をはじめとする自然言語では、「変数」に相当するものがほとんど使われないことを意味している。自然言語で変数に相当するものとは、仮につける名前である。契約書などでは、甲、乙、丙などを使って特定のものを指示する。数学などではａ，ｂ，ｃなどを使う。しかし、そうした変数は、われわれの日常会話では出現しない。むしろ概念クラス名や代名詞を使って指示物を表現する。そのほうが自然であり、理解も速い。

このことを、プログラム言語の問題の上に移せば、

「変数がない（少ない）ような言語設計で、使いやすいプログラム言語が実現する可能性がある」

ということが示唆されるのである。

一昔前、ｇｏｔｏ文は構造化プログラミングの考えのもとに批判された。ここでｇｏｔｏ文という言葉が指しているのは、ｇｏｔｏ文自体とその行き先のラベルの組である。この組で、プログラムの構造に無関係なリンクがあまりにも自由に書けることが問題であった。同じように、プログラムのなかの変数も、場所と場所を結び付けるリンクである。変数が多いこともまたプログラムの性質として悪いことであると考えられる。

われわれはこうした観点から、引数の位置に項が埋め込まれた、論理型プログラム言語を設計してきた（［１］，［２］）。そこでは、Ｐｒｏｌｏｇの項を、別の項の引数の部分に埋め込んだ形式を提案した。これは、表記上は項書換えシステムに似た、論理プログラム言語系になっている。こうして引数そして変数の出現が大幅に減らされた。

本稿ではさらに、引数のない論理型言語を設計する。引数のないプログラム言語としてはBackusの関数型言語ＦＰがある（［３］）。その意義は、並行処理によるフォンノイマン・ボトルネックの解決の可能性、およびプログラム代数を理論化することによるプログラム検証にあると思われる。ここでは、そうした視点はあるにせよ、利用者インターフェースを中心的な目的としている。つまり上に述べたように、関数的表現の方が、引数対応による表現より本来の自然言語の表現に近いという観点からの利用者親和性をめざす。

２．機能設計

述語のデータポート

Ｐｒｏｌｏｇでは述語の引数にある変数を介してデータを交換する。本システムでは、述語は二つのデータ交換ポートを持つ。ポートを他の述語のポートに接続することがプログラムになる。データの方向については、

（１）１方向性ポート、

（２）２方向性ポート

の２通りを検討している。

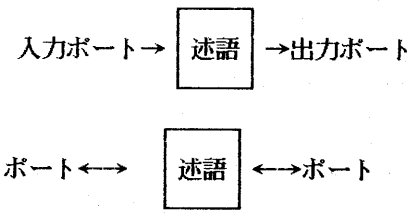

図１．二つのデータ交換

１方向性ポートはＦＰと同様のデータ交換になる。ただし、ＦＰと異なり、本システムは非決定的な機能を持つ。Ｐｒｏｌｏｇとしてみると、両方の引数に対して入出力のモードを固定したものとみなせる。

２方向性ポートは、Ｐｒｏｌｏｇのアリティ２の述語とみなせる。

１方向性ポートにすることにより、処理速度の高速化が図れる。また、自動的にモード宣言をしたことになり、Ｐｒｏｌｏｇにおけるモード宣言の手間がない。

逆に、１方向性ポートは逆方向計算ができず、また２引数とも入力であるような計算ができないという欠点を持つ。引数が二つにまとまることは、内部的な変数の分類ができない点で、高速化しにくい点もある。

両者のどちらを取るかはむずかしい問題であるが、ここでは前者の設計を採用する。その欠点は「逆」というメタ述語を用意することにより一部カバーする。

なお、プロトタイプとして作成したシステムは２方向性ポートを持っている。

データ型

ポートを通じてやりとりされるデータは、リストないし単純な名前や数値の定数である。すなわち、Ｐｒｏｌｏｇのデータと考えてよい。

述語の機能

述語は上のように１方向性としたことにより、次のような機能を持つ。

①入力ポートからデータを受け取る
②データにより計算を行い、失敗または成功をする
③成功した場合、出力ポートにデータを出すとともに、出力側に述語があればそれを起動する
④失敗した場合、入力側の述語の処理にもどる
⑤出力側の述語からもどってきたとき、さらに計算して再び②以下を行う

このようにＰｒｏｌｏｇの計算と類似するが、データは授受の相手が固定的である。

３．言語設計

式の形

本言語仕様では、式は基本的には述語をメタ述語で結びつけたものである。つまり、

＜式＞：：＝
　＜述語＞｜＜単項メタ述語＞＜式＞｜
　＜式＞＜２項メタ述語＞＜式＞

ここで、単項および２項のメタ述語というものは、述語からなる式に対する演算子である。用意されているメタ述語は次の節で述べる。

メタ述語の間には演算子優先順位があり、

その順で部分式が結合する。また、括弧（）によってさきに結合する。

節

本言語のプログラムは手続きからなる。手続きは同一の述語を頭部にもつ節の集まりである。手続きは一つの述語の機能を定義する。節は式の形をしているが、詳しくは、

＜定義節＞：：＝
　＜定義される述語＞とは＜一般の式＞。

という形をしている。これはＰｒｏｌｏｇの定義節に対応するものである。

ゴール節に対応するのは、

＜ゴール節＞：：＝
　＜定数述語＞の＜一般の式＞

という形をしている（定数述語は後述）。

４．メタ述語

メタ述語とは、引数として式を持つ演算子である。述語を組み合わせてより複雑な機能を表現するために使う。以下に個々のメタ述語とその機能を示す。その際、例となる節表現に対して、同じ機能をもつＰｒｏｌｏｇ節を示すことにより分かりやすく解説する。

とは

Ｐｒｏｌｏｇの「：－」にあたるもので、述語の定義をするものである。

例
日本語の意味：旦那とは夫のことである。
関数型の表現：danna(X) = otto(X)
Prologの表現：

```
danna(X,DANNA):-otto(X,DANNA).
```

本言語の表現：旦那とは夫。

の

この演算子でつないだ二つの式のうち、一方の出力ポートを他方の入力ポートにつなぐものである。言い替えれば二つの写像を直列につないだことになる。両者が成功しないと全体として成功しない。

例
日本語の意味：孫とは子の子のことである。
関数型の表現：mago(X) = ko(ko(X))
Prologの表現：

```
mago(X,MAGO):-ko(X,KO),ko(KO,MAGO).
```

本言語の表現：孫とは子の子。

で

この演算子でつないだ二つの式に同じ入力を入れ、両者が同じ出力を出すならば、それを全体の出力とする。同じ出力がないならば、失敗する。すなわち、両式の共通部分を表す。

例
日本語の意味：
兄とはbrotherであって年上の人のことである。
Prologの表現：

```
ani(X,ANI):-
    brother(X,ANI),toshiue(X,ANI).
```

本言語の表現：兄とはbrotherで年上。

や

この演算子でつないだ二つの式に同じ入力を入れ、両者の出力を順に全体の出力とする。すなわち、両式の出力を合わせたものを表す。
例
日本語の意味：
brotherとは兄や弟のことである。
Prologの表現：

```
brother(X,BROTHER):-
    ani(X,BROTHER);otouto(X,BROUTER).
```

本言語の表現：brotherとは兄や弟。

なお、Ｐｒｏｌｏｇで、

```
brother(X,BROTHER):-ani(X,BROTHER).
brother(X,BROTHER):-otouto(X,BROUTER).
```

と分けられるのと同様、本言語でも、

brotherとは兄。
brotherとは弟。

というように二つの節で表現することができる。

なら　（なら　および　ほかは）

二つの式に同じ入力を入れ、最初の式が成功してなににせよ出力があるなら、（後の式が「ほかは」で結び付けた形でないとき）後の式の出力を全体の出力とする。最初の式が失敗するならば、全体としても失敗する。

後の式が「ほかは」でむすびつけた形のとき、最初の式が成功してなににせよ出力があるなら「ほかは」の前の述語に入力をいれその出力を全体の出力にする。そうでない場合は、「ほかは」の後の述語で同様のことをする。

例
日本語の意味：絶対値とは、ある数が負なら、その数の符号を変えたもので、そのほかのときは、その数自身である。
Prologの表現：

```
abs(X,ABS):-(X<0 -> ABS=X; ABS is -X).
```

本言語の表現：
絶対値とは負ならマイナスほかは自身。

［　］

角括弧は、そのなかに並んだいくつかの式に同じ入力を入れ、おのおのの出力を並べたものを全体の出力とする。その式のなかで一つでも失敗するものがあれば全体として失敗する。

ｌｉｓｐやＰｒｏｌｏｇと同様の、リスト表記とドット表記をもつが、これはsyntax sugarである。つまり、これは「と」というメタ述語であり、

［式１　｜　式２］

は、内部的には、

式１　と　式２

という形をしたものとする。

例
日本語の意味：ある人の個人情報とは、その氏名自身、住所、電話番号からなる。

Prologの表現：

```
kojinjouhou(X,[X,JUUSHO,DENWA]):-
    juusho(X,JUUSHO),denwa(X,DENWA).
```

本言語の表現：
個人情報とは［自身、住所、電話］。

逆

式に対して、入力と出力を逆転した機能のものを作る。

例
日本語の意味：親とは子の反対の概念である。
Prologの表現：oya(X,OYA):-ko(OYA,X).

本言語の表現：親とは逆子。

各

全体の入力であるリストの各要素に対してその述語を作用させ、その出力をつないだリストを全体の出力とする。

ｌｉｓｐのmapcar、ＦＰのαに相当するものである。
例
日本語の意味：
個人情報表とは各人の個人情報のリストである。
Prologの表現：

```
kojinjouhouhyou([],[]).
```

```
kojinjouhouhyou([XIL],[JXIJL]):-
    kojinjouhou(X,JX),
    kojinjouhouhyou(L,JL).
```

本言語の表現：

個人情報表とは各個人情報。

総

全体の入力であるリストの先頭から二つの要素を取り出し、その組に対して述語を作用させ、それと三番目の要素との組にさらにその述語を作用させる、というように繰り返して、最後まで計算した結果を全体の出力として出す。ＦＰの「／」に相当するものである。

例

日本語の意味：合計とは一つ一つの数の総和のことである。

Prologの表現：

```
goukei([X],X).
goukei([X,YIL],GOUKEI):-XY is X+Y,
  goukei([XYIL],GOUKEI).
```

本言語の表現：合計とは総和。

全

入力に対して述語を作用させ、成功して出力されるものを順に並べて、リストとしたものを全体の出力とする。Ｐｒｏｌｏｇのｂａｇｏｆのようなものである。

日本語の意味：すべての孫の表とは孫であるものをすべて並べたリストである。。

Prologの表現：

```
magohyou(X,HYOU):-
    bagof(MAGO,mago(X,MAGO),HYOU).
```

本言語の表現：孫表とは全孫。

５．組み込み述語など

組込み述語としては、既存のリスト処理言語の持つリスト処理の多くをカバーするような述語を用意することを考えている。たとえば、「最初」（ｃａｒにあたる）、「最初以外」（ｃｄｒにあたる）などである。

ただし、ｃｏｎｓ系のものはＰｒｏｌｏｇのようにリストそのものの形を表面上はとる。中身は上述のようにメタ述語である。

データを後の処理に渡すために、「自身」という述語が用意される。これは入力されたものをそのまま出力するものである。

定数述語という表現を用いる。これは、どんな入力があろうと、一つの出力を出して一度だけ成功する述語表現である。プログラム中では、定数そのものを書けば、それを出力する定数述語を表すことになる。

６．日本語化の方法

表現形式

表現を日本語化するに際しては、たとえば、

ｆ（ｇ（ｈ（ｘ））

という形は、

ｘのｈのｇのｆ

とした。日本語の語順が逆ポーランド的であるためである。ｆ，ｇ，ｈという関数を要素とみた場合は、中置辞（ｉｎｆｉｘ）記法とみられる。ここで「の」は

ｆｏｇｏｈ

としたときの「o」にあたるものである。

日本語の言葉の選択

特定の機能のメタ述語を設計したあと、その日本語名称を決定した。日本語を含め、どの自然言語も個々の語にははっきり規定された意味がある。その意味との整合性をとるように配慮した。２項メタ述語は可能な限り助詞を用いるようにした。したがってこれは平仮名表記となる。単項メタ述語はなるべく漢字の接頭語的な形になるようにした。

語の句切り

述語となるのは平仮名以外の単一の文字種

からなる列とした。それ以外の文字列を述語名としたい場合には、Ｐｒｏｌｏｇなどと同じように引用符でくくる。

今日までの日本語プログラム言語では、すべて分かち書きで記述されていた。しかし、分かち書きは今一般に使っている日本語表現とは異なる。本言語では、

「文字種の境目で区切る」

という方法で区切る。たとえば、

絶対値とは負ならマイナスほかは自身。

という列は、

絶対値／とは／負／なら／マイナス／
ほかは／自身／。

と区切られる。

７．まとめ

一般の利用者にとって使い勝手のよい論理型プログラム言語の言語仕様を検討した。この研究に関係する理念は、

１．Ｐｒｏｌｏｇ的論理プログラム
２．ＦＰ的関数プログラム
３．文法と意味にねざした日本語表現

である。これらの融合により、日本での論理プログラミングのひとつの方向が示されると考える。

この言語について、Ｐｒｏｌｏｇを用いて試作システムを作成した。

本研究は、科研費（研究課題番号62580018）の援助を受けている。

参考文献

［１］小谷善行、変数記述を減らしたPROLOGの仕様、情報処理学会第29回全国大会講演論文集、１９８４．

［２］小谷善行、ＬＯＧＯの「ヒューマン・フレンドリ」性とその日本語Ｐｒｏｌｏｇへの応用、ｐｐ２０７－２１２，ヒューマンフレンドリーなシステムシンポジウム報告集、情報処理学会、（１９８６）．

［３］J.Backus, Can programming be liberated from the von Neuman style? A functional style and algebra of programs, CACM, Vol.21, No.8, pp.613-641,1978.

付録　プログラム例

ｍｅｍｂｅｒ述語

１．Ｐｒｏｌｏｇプログラム

```
member(X,[X|_]).
member(X,[_|L]):-member(X,L).
```

２．本言語のプログラム

memberとは最初。
memberとは最初以外のmember。

ａｐｐｅｎｄ

１．Ｐｒｏｌｏｇプログラム

```
append([],X,X).
append([A|X],Y,[A|Z]):-append(X,Y,Z).
```

２．本言語のプログラム

appendとは最初の空なら二番目。
appendとは
　［最初の最初｜
　　［最初の二番目、二番目］のappend］。

平均と分散

１．本言語のプログラム

分散とは［自身、平均」の右分配の
　　各（差の二乗）の平均。
平均とは［合計、個数］の商。

個数とは各１の合計。
合計とは総和。
二乗とは［自身、自身］の積。

クイックソート

１．Ｐｒｏｌｏｇプログラム

```
qs([],[]).
qs([A|X],S):-
        partition(A,X,X1,X2),
        qs(X1,S1),qs(X2,S2),
        append(S1,[A|S2],S).
partition(A,[],[],[]).
partition(A,[B|X],[B|X1],X2):-
        A>B,!,partition(A,X,X1,X2).
partition(A,[B|X],X1,[B|X2]):-
        partition(A,X,X1,X2).
```

２．本言語のプログラム

クイックソートとは空なら［］。
クイックソートとは
　［最初、分割の各クイックソート］の
　［二番目の最初、
　　［最初｜二番目の二番目］］の
　連結。

分割とは
　（最初以外の空）なら［［］、［］］。
分割とは
　（［最初、二番目］の大きい）なら
　　［二番目、
　　　［最初｜最初以外の最初以外］の分割］
　　の
　　［［最初｜二番目の最初］、
　　　二番目の二番目］
　ほかは
　　［二番目、
　　　［最初｜最初以外の最初以外］の分割］
　　の
　　［二番目の最初、
　　　［最初｜二番目の二番目］］。