OUKA
Osaka University Knowledge Archive

Title	知識情報処理言語PROLOG（その1）
Author(s)	溝口, 理一郎
Citation	大阪大学大型計算機センターニュース. 1984, 54, p. 37-44
Version Type	VoR
URL	https://hdl.handle.net/11094/65617
rights	
Note	

Osaka University Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

Osaka University

資　料

知識情報処理言語ＰＲＯＬＯＧ（その１）

大阪大学産業科学研究所　溝　口　理一郎

1. まえがき

最近、Prolog [1] という名の言語がコンピュータ関係の雑誌などで盛んに紹介され、注目を集めています。Prolog ブームと言っても良いような感じがします。もちろん、その原因は通産省の大型プロジェクト「第5世代コンピュータ」[2] にPrologが核言語として採用されたことにあります。この第5世代コンピュータは、日本はもちろん、世界中の注目を集めている大プロジェクトですが、その目的は「考えるコンピュータ」、「知的なコンピュータ」を開発することです。「エッ!! 考えるコンピュータなんて本当に作れるの?!」という人もいらっしゃるでしょうが、本当に作れるかどうかは別として、少なくとも従来の汎用コンピュータよりは「知的な処理」に適したコンピュータを1991年までに作ろうとしているのは事実です。そして、その第5世代コンピュータのソフトウェアの核となる言語がProlog というわけなのです。こういう話をしていると、「知識情報処理とかいうのは、どうせ訳の分からない記号処理をゴタゴタやるんだろうが、コンピュータは数値計算を高速にやりさえすればいいのであって、数値計算には役に立たない記号処理なんかに係っているのは時間の浪費である」という声が聞えてきそうな気がします。でも、ちょっと待ってください。記号処理は数値計算に関係ないとおっしゃいますが、誰があなたの書いたプログラムの構文のチェックをし、マシン語に変換し、しかもコードの最適化までしてくれているのか御存じですか？　そうです、あなたのプログラムはコンパイラがなければ、何の計算もしてくれないのです。もちろん、ＯＳがいなければログオンすらできません。ＯＳやコンパイラがやる仕事は記号処理以外の何ものでもないことを考えますと、記号処理の重要性をご理解頂けると思います。記号処理は少なくともコンピュータを使い易くすることに役立っています。もちろん、それだけではなく、記号処理技術は情報工学の重要テーマである、人工知能（最近では知識情報処理と呼ばれることの方が多いのですが）の研究に不可欠のものであり、その必要性は今後益々増えるものと思われます。

さて、まえおきはこれ位いにして、Prolog の話に入ろうと思います。Prolog は記号処理用の新しい言語で、LISP [3] と並ぶ人工知能用言語として定着しつつあります。この解説では少しでも多くの人にProlog に興味を持っていただけるようになることを願いつつ、Prolog の良さを判り易く紹介してみたいと思います。本計算センターではＳhapeUp というPrologが使えますので、それを例にとりながら具体例を多くして話を進めたいと思います。

2. Prolog 入門

PrologはPROgramming in LOGicの略ですが、その名が示す通りLOGIC(論理)正確には一階述語論理(の部分集合であるホーンロジック)に基くプログラミング言語です。従って、Prologの本質を理解するには述語論理と定理証明[4]の知識が必要になるのですが、ここではそのような難しい話は他に譲ることにして[1]、プログラミング言語としてのPrologを紹介するつもりです。

何はともあれ例をお見せしましょう。例えば、

「TOMはMARYを愛している」

という事実はPrologでは

```
LOVES(TOM, MARY).
```

と書きます。これでも立派なプログラムです。

「TOMは誰を愛していますか?」

という質問は

```
?-LOVES(TOM, *X).
```

とかきますが、コンピュータに上のプログラムを入力した後にこれを実行すると

```
*X = MARY
```

という答えがかえってきます。或は、

```
?-LOVES(*X, MARY).
```

これは「誰がMARYを愛していますか?」という質問になりますが、システムは

```
*X = TOM
```

と答えます。ここで、最初に"?-"があるものは実行命令になることを示しています。又、LOVES(TOM, MARY)における"LOVES"は関数子(functor)と呼ばれ、TOM, MARY は関数子LOVESの引数であり、それぞれ定数と呼ばれます。"*"で始まるもの(この場合は*X)は変数であり、実行することにより値が代入されるものです。「何の為にこんなつまらないことをするのだろう。」と思われる方もいらっしゃるでしょうが、もう少し我慢をして下さい。「愛しあっている男女が一緒にいればKISSをする」ということをPrologで書けば、

```
KISS(*MAN, *WOMAN) :-LOVES(*MAN, *WOMAN),
                     TOGETHER(*MAN, *WOMAN).
```

となります。このA:-B, Cの形式は、一般に if B and C then A という含意関係を表わしますが、これをAを実行するにはBとCを実行せよ、或はAが成立するかどうかを知るには、BとCが成立するかどうかを確かめればよい、等のように色々な風に解釈することができます。いずれに解釈するかはプログラマの自由であり、システムがすることは当然ですが、1つです。

ここで、次のプログラムを入力したとします。

```
LOVES(TOM, SUSAN).
LOVES(TOM, MARY).
TOGETHER(TOM, MARY).
KISS(*MAN, *WOMAN):-LOVES(*MAN, *WOMAN),
                    TOGETHER(*MAN, *WOMAN).
```

このとき、

```
?-LOVES(TOM, SUSAN).
YES
?-KISS(TOM, SUSAN).
NO
?-KISS(TOM, *X).
*X = MARY
```

のような質疑応答ができます。

これらを日本語で言い換えると、

「TOMはSUSANを愛していますか？」
「はい」
「TOMはSUSANにKISSをしますか？」
「いいえ」
「TOMは誰にKISSをしますか？」
「MARYです」

となります。「まるで、データベースを作って、それへの問い合わせをやっているみたいだなァ」と思った方はいらっしゃいませんか？　そうです。Prolog は関係データベースと密接な関係を持っているのです。上の例では、ちょうど最初の3行が「TOMはMARYを愛している」とかいうような事実としての(物と物との関係としての)データと見ることができるのです。それでは、4行目はどう考えればよいのでしょうか。KISS(TOM, MARY)という関係は、データとしては入っていないのに、3番目の質問では、TOMがMARYにKISSをします、と答えています。実はこれは、直接TOMがMARYにKISSするということを示すデータではないが、一般に「愛していて、一緒にいればKISSをする」と言えるので、TOMはMARYを愛しており、2人は一緒にいるというデータがあるから、TOMはMARYにKISSをするに違いない、と推論をしているのです。4行目のプログラムは、その推論する規則のようなものを表現していることになります。いかがでしょうか？　少しは人間らしく、頭を働かせて知的な処理をしたように思われましたでしょうか？　知

識情報処理のかすかな匂いがしたように感じていただければ幸いです。

ここら辺で、Prolog がどのような処理をして上の例題のプログラムを実行しているかを見てみることにします。先ず、Prolog の最も重要な概念であるパターンマッチング（正式にはユニフィケーション）というものを御紹介いたしましょう。パターンマッチングとは、２つのパターンが一致（マッチ）するかどうかを調べて、一致すれば成功、一致しなければ失敗するというものです。例をみてみましょう。

TOM＜－－＞TOM　　成功

TOM＜－－＞MARY　　失敗

LOVES（TOM, MARY）＜－－＞LOVES（TOM, SUSAN）　　失敗

要するに同じものは成功、同じでなければ失敗です。これでは何も面白くありませんが、変数が入ると面白くなります。

LOVES（TOM, *X）＜－－＞LOVES（TOM, MARY）　　成功

*X＝MARY

この例では成功するだけでなく、変数*XにMARYが代入されます。変数は、何にでも一致して、その値を持つことになるのです。もう１つ例をあげます。

LOVES（*X, *Y）＜－－＞LOVES（TOM, MARY）　　成功

*X＝TOM

*Y＝MARY

次の例では、

LOVES（TOM, *X）＜－－＞LIKES（TOM, MARY）　　失敗

関数子が違うので失敗し、当然*Xには何も代入されません。

LOVES（TOM, *X）＜－－＞LOVES（TOM, SISTER（JOHN））　　成功

*X＝SISTER（JOHN）

変数には、SISTER（JOHN）のようなものでも大体何でも代入できます（厳密には項と呼ばれるPrologの要素が代入可能です。詳しくは[1]を参照して下さい）。

上の例では、変数には未だ値が代入されていないことを暗黙に仮定していました。もし、既に変数に値が代入されていたとすれば、当然ですが、最初の例と同じようにその値が相手の値と等しければマッチングは成功し、等しくなければ失敗します。

さて、最も面白いのが未だ何も代入されていない変数同志のマッチングです。これがPrologの最大の武器となるのです。次の例を見て下さい。

LOVES（TOM, *X）＜－－＞LOVES（TOM, *Y）　　成功

*X＝*Y

この時点では、変数の値は決まらないが、とにかく2つの変数は等しい値を持つという情報だけが付け加わります。このように、未だ値が決まらないまま、分かる情報だけ付け加えていきながら、処理を先に進めて行けるところがユニークなところです。言い忘れましたが、ピリオドで区切られた1つのProlog単位を節と呼び、変数の値の有効範囲は1つの節だけです。従って、節が違えば同じ名前でも全く違う変数となります。

パターンマッチングについてはこれ位にして、次に制御の流れを見ていきましょう。Prologは与えられた質問(これを以後ゴールと呼びます)と節の左辺("：－"の左、"：－"がない節は右辺がないものと考える。)とのパターンマッチングを上から順にして行きます。そして、最初にマッチした節の右辺をゴールにして再び節の左辺とのマッチングを上から試して行き、マッチングするべきゴールがなくなった時点で、計算を終了します。上のプログラム例を見て下さい。?－KISS(TOM, *X)を実行した場合を考えてみましょう。KISSで始まる節は、1つしかありませんので、それとのマッチング即ち、

KISS(TOM, *X)<－－>KISS(*MAN, *WOMAN)

に成功して、

*MAN＝TOM

*WOMAN＝*X

という変数の結合が行われます。そして、次に実行すべきゴールの例としてKISSの節の右辺、即ち

LOVES(TOM, *X), TOGETHER(TOM, *X)

が選ばれます。ここで、*MANがTOMに、*WOMANが*Xに替わっていることに注意して下さい。ゴールは左から順に実行されますので、LOVES(TOM, *X)が各節のヘッドとマッチするかどうかプログラムの上から順に確かめられていきます。そして、この場合LOVES(TOM, SUSAN)とマッチして

*X＝SUSAN

という変数の結合が行われます。もし、マッチングが失敗すれば、次のLOVES(TOM, MARY)とマッチングを試みます。LOVES(TOM, SUSAN)には右辺がないので新しく実行すべきゴールはなく、従って、ゴールLOVES(TOM, *X)は成功(実行が終了)します。このときゴール例は

TOGETHER(TOM, SUSAN)

となります。もし、LOVES(TOM, SUSAN)の右辺にゴールがあれば(例えば、LOVES(TOM, SUSAN)：－BEAUTIFUL(SUSAN).)、ゴール列は

BEAUTIFUL(SUSAN), TOGETHER(TOM, SUSAN)

となります。次に、TOGETHER(TOM, SUSAN)が実行されますが、プログラムにはTOGETHER(TOM, MARY)しかなく、これとのマッチングは成功しませんのでTOGETHER(TOM, SUSAN)の実行は失敗します。システムはあるゴールの実行に失敗すると一つ前のゴールをやり直そうとします。この操作はバックトラックと呼ばれます。この場合、一つ前のゴールはLOVES(TOM, *X)ですが、前回の実行は最初のLOVES の節とマッチングすることにより実行を終了し、*X = SUSANという結果を返しています。ここで、実行をやり直すとは、マッチングの他の可能性を試すことを意味しています。即ち、一つ目ではなく2つ目のLOVESとのマッチングを試みます。ここでゴール列はLOVES(TOM, *X), TOGETHER(TOM, *X)にもどっています。このとき、LOVES(TOM, *X)とLOVES(TOM, MARY)とのマッチングは成功し、

*X = MARY

のように今度は*XにMARYが代入されます。そしてTOGETHER(TOM, MARY)が実行されます。これも成功するので全てのゴールが成功し、実行全体が成功します。答えとしては、*X に代入されているMARYが出力され、めでたしめでたしとなるわけです。

Prolog の動きは、大体御理解いただけたと思いますが、「Prologはデータベースの検索しかできないのですか?」と言われそうですので、もっと一般のいわゆる記号処理の典型と思われるリスト処理の例をお目にかけましょう。Prolog で、n個の要素から成るリストは[a1, a2, …, an]と書きます。リストも前に述べた項ですし、要素ai も項です。従って、

[TOM, MARY, JOHN]

[RICH(TOM), [MARY, SUSAN]]

もリストです。何も入っていないリスト(空リスト)は、[ ]と書きます。リスト[A, B, C] は省略した記法で、正式には " . " を関数子として、[A, B, C] = .(A, [B, C]) = .(A, .(B, [C])) = .(A, .(B, .(C, [ ])))と書くべきものなのです。2進木で書くと、図1のようになっています。従って、リストは第1要素と残りというとらえ方をします。第1要素をヘッド、残りをテールと言います。テールは常にリストです。[A] のテールは[ ]、[[A], B]のヘッドは[A]です。リストをヘッドとテールに分ける記号に " | " があります。

[A|*X] は、テールは何でもいいが、ヘッドがAであるリストを表わします。" | "の前に何を書いてもかまいませんが、後には1つの変数以外のものは書けません。リストはProlog の基本的なデータ構造で、FORTRANの配列に相当するものです。FORTRAN

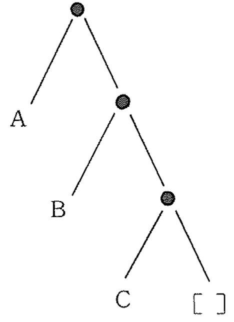

図1　リスト[A, B, C]の2進木表現

が配列に数値を入れて種々の計算をする様に、Prolog はリストに記号を入れて記号の書き換え（記号処理）を行います。

ある項が、あるリストの要素に含まれるかどうかを調べるプログラム member を次に示します。リストの先頭から調べていくことにします。

```
MEMBER(*X, [*X|*TAIL]).
MEMBER(*X, [*Y|*TAIL]):-MEMBER(*X, *TAIL).
```

最初の節は、「リストのヘッドと注目する項が同じであれば、それはそのリストに属する」と読むことができます（当たり前ですが、同じ変数は同じ値を持たなければなりません）。2番目の節は、「リストのテールに、その項が含まれていれば、その項はそのリストに含まれている」と読むことができます。Prolog は、上の節から順にパターンマッチングを行い（実行し）ますから、2番目の節が実行される時は、最初の節が失敗した時、即ちリストのヘッドとその項が等しくなかった時なのです。従って、2番目の節を分かり易く手続き的に読むと、「そのリストのヘッドがその項と等しくなければ、そのリストのテールに含まれているかどうか調べなさい」となります。この方が、さっきのより分かり易いと思います。

```
?-MEMBER(A, [B, A]).
```

を実行すると、2番目の節とマッチして、*TAIL に[A]が代入され、次にMEMBER(A,[A])が実行されます。そして、1番目の節とマッチして実行は成功し、システムはyes と答えます。

```
?-MEMBER(A, [B, C]).
```

を実行すると、2番目の節とマッチしてMEMBER(A, [C])が実行され、また2番目の節とマッチしてMEMBER(A, [ ])が実行されます。しかし、[ ]は2つの節の第2引数のいずれともマッチしないので、MEMBER(A, [ ])は失敗します。ここでバックトラックをしますが、MEMBER(A,[C])もMEMBER(A, [B,C]) も2つ目の節以外の節とマッチする可能性はないので、実行全体が失敗してシステムはno と答えます。

次に、2つのリストを連結するプログラムAPPENDを作りましょう。

```
?-APPEND([A, B], [C, D], *X).
*X=[A, B, C, D]
```

と答えてほしいわけです。今度も簡単で、たったの2行でできます。

```
APPEND([ ], *X, *X).
APPEND([*H|*X], *Y, [*H|*Z]):-APPEND(*X, *Y, *Z).
```

最初の節は、「空リストに何を連結しても、結果はそのリストです」、2番目の節は、「*X と *Y の連結の結果が *Z であれば、*X に *H を付け加えたリストと *Y の連結は *Z に *H を付け加えたものに等しい」と読むことができます。これは、MEMBER と同様、再帰的な定義になってい

ますが、階乗の定義に似ていますので、それと比べれば理解が容易だと思います。階乗！は

　　　0！＝1

　　　n！＝n x（n－1）！

で定義されます。数値の0はリストでは空リストに、整数から1を引くこと（n－1）は、リストのテールをとること（[*H|*X]の*Xを用いてAPPENDを呼ぶこと）に対応しますので、それを念頭において、もう一度APPENDのプログラムを見直して下されば理解も深まるものと思います。

以上でPrologの基本的なところの話を終えて、次回は自然言語の構文解析プログラムを例にとり、もう少し高度な話へと進んで行きたいと思います。

p. s. この解説では、多分触れないと思いますが、Prologももちろん数値計算機能を持っています。例えば、

　　　+(*Y, *X, 5)

と書くと*Xの値に5を加えたものが*Yに代入されます。御安心下さい。

参考文献

(1) 中村訳「Prologプログラミング」マイクロソフトウェア 1983

(2) 元岡編「Fifth Generation Computer Systems」North Holland, 1982

(3) 白井、安部訳「LISP」培風館 1982

(4) C. L. Chang et al.「Symbolic Logic and Machanical Theorem Proving」Academic Press 1973