7. 解　説

ＰＲＯＬＯＧの紹介

富士通静岡エンジニアリング
第２開発部第２開発課 中村　一夫・鈴木　剛

1.　はじめに

人工知能（Artificial Intelligence 略称 AI）の研究および実用化は、現在世界中で非常に活発に行われています。このような状況下にあって、日本の第五世代コンピュータ開発機構（ＩＣＯＴ）が、新世代コンピュータの核言語（機械語）のベースとして採用したのが Prolog と呼ばれる言語です。このため、それまであまりなじみのなかった Prolog は非常に注目を集め、人工知能向け言語として LISP に次ぐ地位を固めるまでに発展するようになりました。これから先も、日本の人工知能に関連する多くのプロジェクトが、Prolog を中心として進められてゆくことは、ほぼ確実であろうと考えられます。ここではこの Prolog の基本について若干説明します。

2.　Prolog の歴史

Prolog の歴史は比較的新しく、フランスのマルセイユ大学の Colmerauer らにより、1972年に定理の証明システムとして考案されたのが始まりです。その後、1974年に当時イギリスのエジンバラ大学にいた Kowalski がプログラミング言語としての解釈を与え、1977年には同大学の Warren らが実用的な処理系を開発し、この仕様を DEC-10 Prolog として世に発表、現在の Prolog 処理系の基礎をつくりました。

3.　Prolog の特徴

FORTRAN のように処理の手続きを記述する手続き型言語や、LISP のように関数ですべての記述する関数型言語と異なり、Prolog は述語論理と呼ばれるものに基づいた論理式で処理を記述するために論理型言語と呼ばれています。Programming in logic の名前の由来もここにあります。Prolog の最大の特徴は、同一化（ユニフィケーション）と呼ばれるパターンマッチングの機能を使った引き数受け渡しのメカニズムにあります。このメカニズムによって他の言語と一味違った、柔軟で優れた記述力を Prolog はもっています。

Prolog 処理系の研究開発は、現在世界中で活発に行われています。処理速度が遅いこと、処理に大きな記憶域を必要とすること、システムプログラミングに必要な機能が不足していること等、いろいろな問題を抱えていましたが、コンパイラの開発、オブジェクト指向の導入等

により、これらの問題は徐々に改善されつつあります。Prolog の応用範囲も、エキスパートシステムの開発、自然言語処理システムの開発、ＬＳＩ－ＣＡＤ、データベース検索システム等各方面に広がっており、今後もいっそう研究開発が進むものと期待されています。

4.　Prolog の基本

Prolog は、人、物、事など(以下オブジェクトと呼ぶ)やそれらの関係を扱った問題を解くためのプログラミング言語であると言えます。以下簡単なプログラム例を用いて Prolog の基本となる動きを紹介します。なお Prolog は会話型の言語であり、端末の前に座ってコンピュータと会話しながら処理を行うことを前提に話を進めていきます。また、説明の都合上すべて例には英語を用います。

4．1　オブジェクトとその関係

オブジェクトとその関係を考える例として、" Boys love girls " (少年は少女が好き)という事実を取り上げることにします。この例は、2個のオブジェクトである " boy "と " girl " (数は無視) の間に " love " の関係があることを示しています。このように関係は事実で表現することができます。なお、関係には通常順序性があるため、オブジェクトの順番を入れ替えると一般にその関係は成立しなくなることに注意する必要があります。

また、関係は規則を用いることによっても表現できます。たとえば " Two persons are brothers if their parents are same " という規則によって、「兄弟」の関係を記述できます。また、二人の人が兄弟の関係にあるか否か調べる方法も示しています。さらに、この規則を「兄弟」の定義と考えることもできます。

Prolog によるプログラミングは、この事実や規則の記述によりオブジェクト間の関係を定義する (Prologに与える) ことによって行われます。いったんプログラムができてしまうと、オブジェクトに関する様々な質問を発することができます。Prolog は与えられた事実や規則をもとに、別の事実や規則を導き出す (推論する) 動作を繰り返して、質問に合う答を探して行きます。

4．2　事実

Prolog では事実は以下のように記述されます。例えば、" Girls love dog " を扱う場合には、

```
love( girls , dog).
```

と記述します。オブジェクト名や関係の名前は小文字で書き始めることと、最後に必ずピリオド (" . ")を打つことが注意点です。オブジェクトとして数字を記述することもできます。さ

らにいくつかの例を示します。右側の文はその説明です。

```
valuable(gold).          : Gold is valuable.
integer(0).              : 0 is integer.
owns(yamada, gold)       : Yamada owns gold.
father(john, mary).      : John is father of Mary.
gives(boss, money, us).  : Boss gives money to us.
```

なお、関係を表わす名前（ love valuab など）を述語と呼び,括弧内のオブジェクト（ gold 0 など）を引き数と呼びます。上の例で love は2引き数の述語であり、valuable は1引き数の述語です。さらに、事実や規則を集めたものをデータベースと呼びます。

4．3　質問

事実を入力したら、以下のように入力することによって、質問を行うことができます。

```
?-love( girls , dog).
```

これは、" Do girls love dog ? " という意味です。この場合も文末にピリオドが必要です。先頭の " ?- " はシステムが出力するプロンプティングマークです。こうした質問を受け取ると、Prolog はすでに入力されているデータベースを探索し、質問と一致する事実を探します。事実と質問は、同じ述語（名前が同じであれば同じ）であり、かつ対応する引き数がすべて等しい時、一致したとみなされます。引き数の個数が異なるものは、異なるものと解釈されます。この一致させようとする行為が同一化（ユニフィケーション）です。一致する事実が発見されると、Prolog は " YES " を返し、そうでなければ " NO " を返します。

もう少し例を示します。以下のようなデータベースがあるとします。

```
syogun( ieyasu ).
daimyo( masamune ).
japanese( ieyasu ).
```

この事実についていろいろ質問してみます。

```
?-syogun( ieyasu ).
YES
?-japanese( masamune ).
NO
?-daimyo( masamune ).
NO
?-kanpaku( hideyoshi ).
NO
```

3番目と4番目の例については、データベース中にそのような事実が見付からないため、Prolog は " NO " を返します。ただし、" NO " は「その時点でのデータベースの内容では同一化ができない」ということを言うのであって、その質問が虚偽であるということを言っているわけではありません。

次に、「だれが」とか「何を」といった一歩進んだ質問について考えます。

4．4　変数

以下のようなデータベースについて考えます。

```
love( boys , dog ).
love( boys , cat ).
love( boys , girls ).
love( girls , cat ).
love( girls , candy ).
love( girls , dog ).
```

ここで、少女が好きなものをすべて探すことにします。その場合、

```
?-love( girls , cat).
?-love( girls , candy).
?-love( girls , dog ).
```

と3回質問をすれば良いわけですが、項目数が多くなるとこれは容易ではありません。また、データベースの内容が詳しくわからない場合には、このような質問は不可能です。" What objects do girls love ? " というような質問ができれば便利です。そのために用いられるのが変数です。変数は大文字で始まる名前であり、任意のオブジェクトを値として取ることができるものと約束します。この変数を用いて、次の質問をします。

```
?-love( girls , X ).
```

この質問に対して Prolog は、

```
X = dog
```

という答を返して、次の応答に備えます。この質問が発せられると、データベースが先頭から探索され同一化が試みられます。述語名と第1引数が一致するものがデータベース中に存在し、また変数は任意のオブジェクトを値としてとれるために、この質問と "love(girls , dog)" の同一化が成功し、変数 " X " は値 " dog "を取ることになります。このような場合にも同一化という言葉を用い、変数 " X " が " dog " と同一化されたと言います。この際に、Prolog はデータベース中の " love(girls , dog) " の位置を覚えておきます。

端末に答を出力して応答待ちになっている状態で、実行キーが押下されると Prolog はそれ

以上データベースの探索を行いません。利用者がその答で満足したという意志表示になるからです。これに対して、セミコロン(" ; ")が入力されると、Prolog は再びデータベースの探索を開始します。この場合に探索の開始位置はデータベースの先頭ではなく、先ほど記憶しておいた位置からになります。また、探索開始の直前に変数 " X " の同一化は解除され、変数は何の値も持っていない状態となります。この状態の変数を未同一化変数と呼びます。この場合も探索の結果質問と同一化可能な事実が発見され、

```
X = cat
```

という答えが出力されます。以下同じことの繰り返しとなります。

同様にして、

```
?-love( boys , X ).
?-love ( X , candy ).
?-love( X , Y ).
```

といった質問が可能です。ただし、さらに変数を増やした

```
?-X( Y , Z ).
```

という質問は不可能です。

次にさらに質問を複雑にして、" What objects do both boys and girls love ? " という場合を考えます。これは、「少年の好きなものは何でかつ少女もそれが好きか?」と解釈されます。すなわち二つの独立した要素から構成されています。Prolog ではこの質問を、

```
?-love( boys , X ),love( girls , X ).
```

と記述します。カンマ (" , ") はかつ、あるいはａｎｄの意味を表します。この質問に対して Prolog は、カンマで区切られた要素一つ一つについて記述された順番にデータベースを探索し、同一化を試みて行きます。この過程を図示すると以下のようになります。

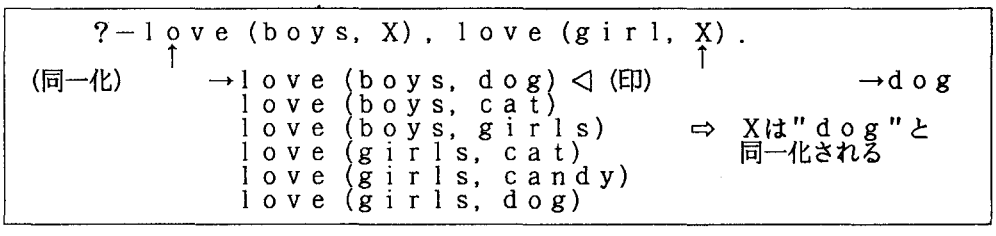

図1　" love(boys , X) " の同一化成功の直後

⇩

```
?－love(boys, X), love(girls, X).
                                  ↑      (X＝dog)
        love(boys, dog) ◁
        love(boys, cat)
        love(boys, girls)     ⇨ データベースの探索
        love(girls, cat)        は印の位置と無関係
        love(girls, candy)      に先頭から行われる
        love(girls, dog)  ← (同一化)
```

図2　" love(girls , X) " の同一化成功

⇩

```
?－love(boys, X), love(girls, X).

X＝dog;                                  セミコロンの入力
                                         によりXの同一化
                                         が解除される
```

図3　端末への答の出力とそれに対する応答

⇩

```
?－love(boys, X), love(girl, X).
    ↑                          ↑
        love(boys, dog)                →cat
(同一化) love(boys, cat)
        love(boys, girls)     ⇨ データベースの探索
        love(girls, cat)        が◁の位置より再開
        love(girls, candy)      される
        love(girls, dog)          (同一化成功後◁の
                                   位置は変わる)
```

図4　" love (boys , X) " の2度目の同一化成功の直後

⇩

```
?－love(boys, X), love(girls, X).
                                  ↑      (X＝cat)
        love(boys, dog)
        love(boys, cat) ◁
        love(boys, girls)     ⇨ 再び先頭から探索が
        love(girls, cat) ←      行われる
        love(girls, candy)
        love(girls, dog)
```

図5　" love(girls , X) " の2度目の同一化成功

⇩

```
?－love(boys, X), love(girls, X).
X＝cat;                                    ⇨ セミコロンの入力
                                             によりXの同一化
                                             が再び解除される
```

図6　端末への答の出力とそれに対する応答

⇩

```
?－love(boys, X), love(girls, X).
    ↑                                          ↑
        love(boys, dog)                          girls
        love(boys, cat)
      →love(boys, girls)      データベースの探索が◁
        love(girls, cat)      の位置より再開される
        love(girls, candy)
        love(girls, dog)
```

図7 " love(boys , X) " の3度目の同一化成功直後

⇩

```
?－love(boys, X), love(girls, X).
                                  ↑        (X＝girls)
        love(boys, dog)
        love(boys, cat) ◁
        love(boys, girls)          ⇨ 該当する事実が
        love(girls, cat)             なく、同一化失敗
        love(girls, candy) ↓
        love(girls, dog)   ×
```

図8　" love(girls , X) " の2度目の同一化成功

⇩

```
?－love(boys, X), love(girls, X).
NO                                         ⇨ これ以上答なし
```

図9　端末へ結果の出力

なお、以前の状態　(印の位置)　に戻って処理を再開する Prolog の動作を後戻り（バックトラッキング）と呼びます。

4．5　規則

Prolog のプログラムが事実と規則から成立っていることは、以前に述べたとおりです。規則は、ある一つの事実が他のいくつかの事実あるいは規則に依存している場合に用いられます。また、ある事実を成立たせるための条件を記述したものとも考えられますし、定義を記述したものとも考えられます。たとえば、

```
I put on sweater if it is cold weather.
X is a bird if X is an animal ,and X has feathers.
N+1 is integer if X is.
```

などです。ここで、 N や X は変数を表します。このような規則を Prolog では以下のように記述します。

```
puton( sweater ) :  weather( cold ).
bird( X ) :  animal( X ) , has_feather( X ).
integer( N ) : integer (M) , N is M+1.
```

記号 " : " は、コロン (" : ") とハイフン (" - ")を結合したものであり、ならばあるいはｉｆと読まれます。カンマは以前の通りです。なお、３番目の例に現れる " M is N-1 " は、変数 " M " と値 " N-1 " (通常の引算) を同一化させるもので、" is " は組込み述語と呼ばれています。詳しくはマニュアルなどを参照してください。" : " の右辺に記述される事実あるいは規則が増えるに従って、条件が厳しくなって行きます。

ここで、

```
?-puton( sweater ).
```

と質問すると、データベース中に " weather(cold) " という事実が定義されていれば、答は " YES " となり、そうでなければ " NO " となります。

```
?-bird( penguin ).
```

と質問した場合には、Prolog は " animal(penguin) " という事実と " has_feather (penguin) という事実の２つをデータベース中から探します。そして２つの事実が発見された場合に、答 "YES " を返します。変数 " X " に値 " penguin " が同一化されると、他の２つの " X " も同時に " penguin " に同一化されます。また、

```
?-bird( X ).
```

と質問することにより、データベース中に定義されている全ての鳥を検索できます。

5.　プログラム例

ここで、端末からプログラムを入力して実行させる簡単な例を示します。例に用いるのは、整数を次々に取り出すプログラムです。なお、プログラムを入力するためには、ある機能（これも組込み述語）を必要とします。詳しいことはマニュアルなどを参照してください。右側の記述は簡単な説明です。

```
?－consult(user).          "consult"はプログラム入力用
                             の組み込み述語
                           "user"は端末の意味
|:integer(0).              "|:"はシステム出力の入力促進文字

|:integer(N)  :  integer(M), N is M+1.

|:!.                              "!"は入力終了の意味

YES

?-integer(1).
YES

?-integer(X).
X=0;
X=1;
X=2;
X=3;
X=4
YES
```

図１０　プログラムの入力と実行の例

6.　おわりに

以上、簡単に Prolog の概要について紹介しました。Prolog は若い言語であるため、未成熟な面もありますが、他の言語にないユニークさもいくつか持っています。実際に使用してみて、是非新しいプログラミングの感触をつかんでほしいと願っています。最後に現在使用可能な処理系の特長を挙げておきます。

① 実質的な国際標準である DEC-10 仕様に準拠している。

② 高速コンパイラを装備している。（処理方式としては引き数コピー方式を採用）

③ アトム名として日本語を使用できる

④ １８０余りの組み込み述語と、ガーベジコレクタを装備している

⑤ 他言語（ FORTRAN など ）で作成したプログラムの呼び出しが可能である

⑥ バッチ配下でも動作する

〔参考文献〕

Prolog に関する書物は多数出版されている。ここではそのごく一部を紹介する。

① 黒川　利明　(1985) : Prologのソフトウェア作法(岩波書店)

② Kowalski (1979) : Logic for Problem Solving (NORTH HOLLAND)

③ Clocksin & Mellish (1984) : Programming in Prolog (2nd edition) (SPRINGER)

④ Bratko (1986) : PROLOG PROGRAMMING FORARTIFICIAL INTELLIGENCE (ADDISON WESLEY)

〔マニュアル〕

富士通PROLOG手引書