医療情報学 2－3
（1979. 10. 6）

述語論理と推論形式

渕　一博
（電子技術総合研究所）

背景

医療診断のコンサルテーション・システムをはじめとして，専門知識を組み込んだ「知識ベース・システム」が注目されている。このようなエキスパート・システムでは、対象世界の知識を客観化することが中心になる。それに伴って，知識をどう表現しておくかまた、その知識をどう働かせるか，という「枠組」も同時に問題になる。

この枠組の選択は、広い意味で「論理系」の選択になる。「知識工学」を推進しているスタンフォード大学（HPP- *Heuristic Programming Project*）やカーネギーメロン大学などでは、プロダクション・システム（PS）がよく用いられている。このPSもまた論理系の一種ということができる。

標準的には（狭い意味では）論理といえば「述語論理」であろうが、それに限られるわけではない。ラムダ計算のような「関数計算」も論理系の一種であり，また「プログラミング言語」も論理であるといえなくはない。

ところで、MITを中心にして，知識の「手続き的表現」と称して、（拡張された）プログラミング言語を知識表現に用いることが提唱された。その気持は，知識をスタティックにではなく、「いかに」ということを含めてダイナミックに把えたいということであったといえよう。そしていわゆる「論理」はそれに適さないものとしてしりぞけられた。しかし，それはことの一面をついたものでしかなかったように思われる。

PSは，一面においては、（手続き的表現として提案された）PLANNERと関連づけることもできるが、より宣言的な性格をもち，ずっと標準的な論理に近い。そしてまた実際に「プログラミング」に用いられている。

それとともに「述語論理」自体についても、それをプログラミング言語として用いることが提案された。これは英国（エディンバラ大学など）やヨーロッパにおいて，最近盛んに実践されはじめている。

また一方では、LISPのような関数型プログラミング言語があるが、最近「関数型プログラミング」が強く提唱されている。

述語計算，関数計算，PSなどは、それぞれ30年代にルーツを持つ、いわば「標準的な」論理である。これを考えれば、最近の動きは、プログラミング言語をより標準的な論理に近づけようというものだと表現してもよいであろう。

このような現象は、知識工学その他の人工知能分野だけではない。いわゆる「ソフトウエア工学」の中にもそのベクトルを認めることができる。

ソフトウエア工学の分野では、「構造的プログラミング」以来、プログラミング「スタイル」の反省が行われている。一方、プログラムの正しさの検証ということが言われ、検証理論や、プログラミング言語の「意味論」が展開された。

ここで興味深いのは、スタイルからいって、プログラミングにとって好ましくない要素とされるものが、他方で意味記述や検証を困難にしている部分に対応しているように見えることである。この現象は、ソフトウエア工学の実践からも、「意味論」の明快な言語

が要請されていると読むことができ、「論理的プログラミング」の動きを裏づけるものと見ることができよう。

この論理的プログラミングの流れとしては、述語論理プログラミング、関数的プログラミング、PSプログラミングを挙げることができる。

述語論理プログラミングの例

述語論理による記述をプログラムと見たてようという提案はKowalski[1]によってなされた。正確には、一階述語論理の中の「ホーン集合」と呼ばれる部分系についてのプログラム的解釈であるが、これについては後で述べる。

この提案は、マルセイユ大学のColmerauerを中心に早速実現された。[2]このシステムはPROLOGと呼ばれる。その後エディンバラ大学ではコンパイラも作成された(Warren)[3]。またミニコンピュータやマイクロコンピュータ上での実現も伝えられて、ヨーロッパ、英国ではかなり普及しはじめているようである。

PROLOGの応用としては、言語理解(QA)システム、定理証明システム、プラン作成システム、数式処理システムなどがある。また最近、関係データベースとの関連も注目されている。

PROLOGプログラムの感じをつかむために、簡単な例題[3]をとり上げる。二つのリストをつなげるappendのプログラムは、PROLOGでは、

```
append(nil, Y, Y).
append((A.B), Y, (A.B1))
        :- append(B, Y, B1).
```

と書かれる。大文字は変数を表わす。append(X, Y, Z)は「リストXとYをつなげたものはZである」ということを表わす。第一の式は、nil(空リスト)をつないでも変らないということ、第二の式は、(A.B)の構造をもったリストをYにつなげた結果は、BをYにつないだ結果をB1とすれば、(A.B1)であるということを表わしている。これらは、appendについての基本的な記述であるということができる。それと同時に、これを「プログラム」として実行させることができるのである。計算は「証明」として行われる。たとえば(a.b.nil)と(c.nil)をappendしたいとき、

append((a.b.nil),(c.nil),X)

を証明する。この式は、

X=(a.b.c.nil)

のとき正しいのだが、それが証明の結果として出てくる。

このプログラムはM-LISPでは、

```
append[x;y]=
  [null[x]→y;
   T→cons[car[x];append[cdr[x];y]]]
```

と書かれるものである。LISPでの、car, cdr, consの操作は、PROLOGでは単一の記法で統一されている。

他の例題として、多項式微分のプログラムを見てみよう。

```
d(U+V,X,DU+DV):-
        d(U,X,DU), d(V,X,DV).
d(U-V,X,DU-DV):-
        d(U,X,DU), d(V,X,DV).
d(U*V,X,DU*V+U*DV):-
        d(U,X,DU), d(V,X,DV).
d(U/V,X,(DU*V-U*DV)/V^2):-
        d(U,X,DU), d(V,X,DV).
d(U^N,X,DU*N*U^N1):-
        integer(N), N1 is N-1, d(U,X,DU).
d(X,X,1).
d(C,X,0):- atomic(C), C≠X.
```

ここに挙げた例は、関数の計算を述語形式にしたもので、非決定性は現れないが、PROLOG自身としては、非決

定的動作を基本にしている。

PROLOGの形式

PROLOGの形式は例題からも察せられるが、簡単にまとめると次のようになる。

項とは、

(i) 変数

(ii) $fn(t,---,t)$ の形

ここで t は項, fn は関数定数。なおこの関数は「Skolem」関数で, 普通の関数よりは「データ構造」に対応するものである。たとえば例題中の「.」はこの種の関数である。正式には、

$.(X,Y)$

であるが、PROLOGでは, infix, prefix, postfix の記法 (syntactic sugar) が宣言できるので、

$X.Y$

と書かれているのである。

素式とは,

(i) $pred(t,---,t)$ の形

t は項, $pred$ は述語定数.

式とは,

(i) B.

(ii) $B :- A_1, A_2, ---, An.$

の形である。ここで、BやA1,---,Anは素式である。プログラムはこのような式の集まりである。

この式は「真」である事実や公理を表わしていると考えることができる。(ii)の形は論理的には、

$A_1 \& A_2 \& ---\& An \rightarrow B$

の意味を持つ。証明(計算)はgoal orientedな推論によっている。

Bを証明するには$A_1, A_2, ---, An$のそれぞれを証明すればよい、というような順で推論が進められる。これは、計算的には、procedureを次々に呼び出していくことに相当する。

なお、この推論は、リゾリューション方式でSNL(Selective Negative Linear)と呼ばれるものである。非決定性動作のときはdepth-firstの探索が行われる。

レゾリューション(推論)が行われるときは、ユニフィケーションと呼ばれる一種の「パターン・マッチング」が行われる。たとえば、

---, append((a.b.nil), A, B), ---

(ユニフィケーション)

append((X.Y), Z, (X.W)) :- ---

では、

X := a

Y := (b.nil)

Z := A

B := (X, W) = (a.W)

というマッチングが行われる。このユニフィケーションが, car, cdr, cons の働きを統合している。

PROLOGの式の形は、実は、正なる素式が(たかだか)1個しかない、という特殊な形をしている。これは一般の一階述語論理の部分系であって、ホーン集合と呼ばれるものである。SNLは, ホーン集合に対して完全な証明手順である。

ホーン集合のような限定を加えれば力が弱くなるように見える。しかし、計算可能な関数(関係)は、ホーン集合で表現可能なことが確かめられている。むしろ, ホーン集合は, 計算, あるいは、プログラムというものによく合った性格をもっており、一般の述語論理での表現(仕様)をホーン集合の形に変換することが、「プログラムの作成」である、と考えることもできるのである。(後述)

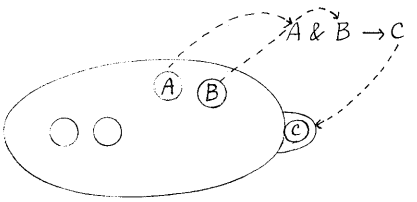
SNLは後向き推論であるが、ホーン集合に対しては、SPU(Selective Positive Unit Resolution)という推論形式がある。これは前向き推論であるが、これもまた、計算過程によく対応している。SPUはPROLOGにはとり入れられていないが、そのような拡張も可能であると考えられる。(4)

プロダクション・システムとの比較

前述したSPUという推論は、単位文（素式）からなる「事実」の言明をもとにして、

$$\frac{A_1, A_2, \cdots, A_n, A_1 \& A_2 \& \cdots \& A_n \rightarrow B}{B}$$

とする推論である。いいかえれば、単位文からなるデータベースがあって、公理（ルール）は、そのデータベースの状態がある条件を充すとき、新しい言明をデータベースにつけ加える働きをする。



このような動作は、PSによく似ている。（実際、PROLOGをPSの一種だと説明する人もいる。）

PSのルール、

$$A_1, A_2, \cdots, A_m \rightarrow B_1, B_2, \cdots, B_n$$

の左辺は、データベースの状態に対する条件，右辺は，データベースに対する操作と解釈される。この右辺の操作では、データベースへの言明の追加だけでなく言明の削除その他が許される。この意味では，PSの方が「強力」である。

もし、PSの使用で、言明の削除といった操作を許さない（使わない）と限定すれば、述語論理（ホーン集合）におけるSPUと一致する。また、この条件でPSをgoal-orientedに働かせればSNL（PROLOG）と一致する。

一般に、PSのルールを述語論理で書きあらためるとすると、

（PS）　$A \rightarrow B$

に対して、状態変数sを導入し、いまデータベースの状態がsであるということを、$S(s)$と表わすとすると、

（PC）　$S(s) \& A(s) \rightarrow S(f(s))$

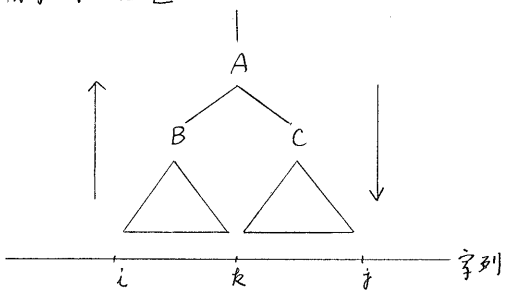
と書ける。ここでfはBに対応して、状態sから新しい状態s'を計算する関数であるとする。

このように考えると、PSを述語論理に埋め込むことができる。そこで、本質的にいうと、PSと述語論理に力の差はない。さきほど、PSの方が強力だといったが、それは表面上のことだということができる。

PSの特殊ケースとして、「句構造文法」がある。これは、非終端記号を含んだ字列自身をデータベースと考えたPSである。ルールは、字列がある条件を充すとき、字列を書きかえる書き換え規則である。書き換え規則，

（PG）　$A \rightarrow BC$

を述語論理的に書きなおしてみる。終端字列の位置について、



の規約を設ける。ここで，位置いとはの間の字列がAというカテゴリーだということを、$D(A, い, は)$ と述語的に表わす。また、PGの規則の存在については、$P2(A, B, C)$と表わすとする。そうすると、PGルールに対して、

$$D(C, ろ, は) \& P2(A, B, C) \& C(B, い, ろ) \rightarrow D(A, い, は)$$

この表現は書き換え規則の意味（性質）をより細かく表わしていると見ることができる。また、この式を（述語

論理）プログラムとしてみると，「パージング」の働きをする。SPU で働かせると、Cocke-Younger-Kasami のアルゴリズムというボトム・アップ・パーザと同じ働きをし、SNL で働かせるとトップ・ダウン・パーザの働きをする。

このようにPS表現と述語表現は一部では一致し、また全体として対応をとることができる。しかしまたそれぞれ異った性格も持っている。

もっとも大きいのは、基本的な考え方である。それは、PSでは、「副作用」を基本とし、述語論理では副作用なしを基本とする、ということであるといえよう。PSでは、データベースの状態変化をルールで表わす。述語論理では、真なる言明の集合は不変である。ただ、実際上，推論の結果，真なる言明が次々に導きだされるということである。

述語計算で、ある時点までに得られた真なる言明の集合をデータベースと考えると、それは「単調に増大」するのみである。これは、述語論理の基本的な性質である。

このことは、述語論理式の意味は、式の順序，その中の条件の書かれた順序には関係しない、あるいは、証明の順序には関係しない、という性質に関連する。順序は、正しさには関係なく効率にだけ関与するのである。これは良い性質であるということができる。

一方、PSは、副作用を主とするので、ルールの順序などで結果が変わることがある。これが実行結果の見通しの悪さにつながることがある。一方、このことはPSの強力さに関連しているのである。

状態変化（副作用）を述語論理の中で表現すると、状態変数の導入など余分のことが生じる。

副作用なしのプログラミングは、関数的プログラミングなど、最近のプログラミング・スタイルの論議で推奨されていることである。これは、プログラムの性質の見やすさ、性質の検証に便利と考えられるからである。

プログラムの多くは、副作用なしに書くことができ、その方が望ましいことである。しかし一方、状態変化は世の中の基本ということもできる。状態変数の導入は一案であるが、もっと自然な表現が求められている。

この観点からは、PSはその性質を持っている。また、副作用なしのように限定して使うこともできる。

述語論理の拡張としては、（古くから）様相論理が考えられている。様相論理は、裏に、「可能世界」のモデルをもっている。この可能世界は、多コンテキストのデータベースと見立てることができる。述語論理プログラミングに「様相」を導入して、状態変化を把えるという案が考えられるが、現段階ではその方法はまだ判然としていない。なお、様相論理の関数形式として「内包論理」がある。内包論理は、言語哲学の分野で開発され、最近は、プログラムの解析にも使われ始めているが、それを直接プログラミングに用いることについては、まだ検討が進んでいない。

論理の拡張としては、一階から高階への方向がある。PSでは高階の操作（ルール自身の生成など）が許されている。ルール自身の操作は、メタ知識に属するものとして、今後ますます重要になる研究テーマである。しかし、それを、プログラミング・レベルの論理に繰入れるかどうかには、まだ問題がある。高階論理の性質自身が、まだ十分に解明されているとはいいがたいからである。どのような階層構造を持たせるか、今後の大きな課題である。

仕様からプログラムへの変換

整数変数の関数 $f(x)$ の最大値を求める例題を考える。

$$max(k,n) \longleftrightarrow 0 \le k \le n \,\&\, (\forall i.\ 0 \le i < n \to f(k) \ge f(i))$$

これは問題の定義（形式的仕様）と考えられる。これはこのままでは「プログラム」とは認めがたい。これをホーン形式にすることを考える。そのために

$$\forall n\ \exists k.\quad max(k,n)$$

を証明することにする。この意味は、すべての n に対して条件を満足する k すなわち答が存在するということである。ここでは自然数である。これを証明するには「数学的帰納法」を用いなければならない。

(1) $n=1$ のとき、 $max(0,1)$ が成立つ。

(2) $n=N$ のとき成立つと仮定し、$n=N+1$ のとき成立つことを証明する。$max(K,N)$ が真で、これから $0 \le K < N$ と $\forall i.\ 0 \le i < N \to f(K) \ge f(i)$ が成立つ。$max(k,N+1)$ を証明するのに場合分けして考える。

$0 \le k < N$ と仮定すれば

$$(\forall i.\ 0 \le i < N \to f(k) \ge f(i)) \,\&\, f(k) \ge f(N)$$

で、$k=K$ とすれば前半が成立ち、$f(K) \ge f(N)$ の条件が残る。次に、$k=N$ と仮定すれば、

$$\forall i.\ 0 \le i < N \to f(N) \ge f(i)$$

が必要であるが、K についての仮定から、$f(N) \ge f(K)$ であればよい。この証明から、

$$max(k,n+1) \leftarrow max(k,n) \,\&\, f(k) \ge f(n)$$
$$max(n,n+1) \leftarrow max(k,n) \,\&\, f(n) \ge f(k)$$

これと $max(0,1)$ を合せれば、これらは PROLOG プログラムと認められる。一方 SPU 的に見るため、

$$\to max(0,1)$$
$$max(k,n) \,\&\, \begin{bmatrix} f(k) \ge f(n) \to max(k,n+1) \\ f(n) \ge f(k) \to max(n,n+1) \end{bmatrix}$$

とまとめられる。これはループ的プログラムと解釈される。この形式と、普通のプログラム的な、

$$k,n := 0,1$$
$$\underline{do}\ n \ne N \to \underline{if}\ f(k) \ge f(n) \to n := n+1$$
$$\square\ f(n) \ge f(k) \to$$
$$k,n := n,n+1\ \underline{fi}\ \underline{od}$$

という形式とはほとんど直接的な対応がつく。

ここでの証明はインフォーマルに述べたが、この過程は形式化することができる。

このような証明のためには、ホーン集合では足りず、数学的帰納法を含んだ一般の（一階）述語論理に対する証明系が必要である。このために「自然推論」を用いている例もある。[5]

参考文献

1) Kowalski, R., A predicate logic as programming language, Proc. IFIPC 1974, pp. 569-574

2) Colmerauer, A., Un system de communication homme-machine en Français, Université d'Aix-Marseille Luminy, 1972

3) Warren, D. & Pereira, L., The language and its implementation compared with LISP, ACM SIGPLAN Notices, Vol. 12, No. 8, 1977, pp. 109-115

4) 渕, 述語論理プログラミング－EPILOG の提案, 情報処理学会, 記号処理 1-2, 1977

5) Hansson, Å. & Tärnlund, S-Å., A Natural Programming Calculus, Proc. 6th IJCAI, 1979, pp. 348-355