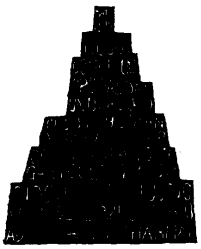


32. 述語論理型言語†

渕　　一　　博††

1. はじめに

述語論理をプログラミング言語と見なそうということが Kowalski[1] やそのほかの人たちによって提案されてから，Prolog を代表例として，言語の設定や処理系の製作が行われてきている．この考えをベースにしたプログラミング・スタイルは「論理プログラミング」と呼ばれている．

プログラミング言語としての述語論理という見方は一般にはまだなじみが薄いものであろうが，述語論理自体は目新しいものではない．形式論理の諸体系の中ではもっとも古くから確立された，いわば標準的な論理体系である．数学の分野では，自然数論や集合論などの形式化のベースとして用いられ，それ自身の数学的性質もよく調べられている．

計算機とのかかわりでも，いろいろな分野で使われてきている．たとえば，関係データベースの理論は述語論理系統のものである．また，ソフトウェア工学の分野で，プログラムの仕様記述ということがいま大きな問題になっているが，形式的仕様記述とか検証といった関連では，述語論理が標準的に採用されることが多い．

エンドユーザ言語として，述語論理を直接的に用いるのが適当かどうかは問題であるとしても，ベースのシステムとしての良さは認められるであろう．たとえば，データベース問合せ用のエンドユーザ言語のなかに 'Query By Examples' というものがある．これは表の図示や項目の例示をもとに問合せを行うものである．説明方式としては述語論理を表面に出していないが，論理的にはその一変種である．

述語論理の成立には，もともと，自然言語の抽象化，形式化ということがあった．もちろん，自然言語と述語論理の間には微妙なギャップがあるのだが，いまのところは，述語論理が自然言語に一番近い形式的体系と考えられる．自然言語を将来，仕様記述やプログラミングに援用するとすれば，さしあたり，述語論理をベースとするのが自然であろう．

プログラミング言語としての述語論理は，非手続き型の（超）高水準言語と見ることができる．プログラミング言語の高機能化の方向として考えられる多くの要素を自然に統合しているのである．それとともに，述語論理をプログラミング言語として用いていこうとすれば，いろいろな拡張が欲しくなってくる．それは逆に，論理自身の拡張につながるであろう．

述語論理型プログラミング言語には，プログラミングの観点の中だけでの魅力もあるが，それとともに，ベースを共通にするそのほかの言語，たとえばデータベース問合せ言語，仕様記述言語，あるいは，検証や合成（段階的改良）の言語などと統一的に接続しうる可能性があることも魅力的である．

ここでは，述語論理型プログラミング言語の代表例である Prolog の紹介から始めて，関連する諸問題を考えてみることにしたい．

2. Prolog の概要

ここでは，エジンバラ大学版の Prolog[2] をもとに簡単な紹介を試みよう．

まず言語の基本的な構成を述べる．

Prolog における基本的な構造単位は「項」である．項は，(i)変数，(ii)定数，(iii)構造　のいずれかである．

変数は，大文字あるいは _ で始まるアトムである．定数は，整数，あるいは変数でないアトムである．構造は，構造名（項1，項2，…，項n）の形である．構造名は定数アトムである．定数アトム自身は無変数の構造と考えられる．

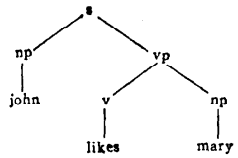
構造は，木構造と考えられる．たとえば，

† Programming Language Based on Predicate Logic by Kazuhiro Fuchi (Electrotechnical Laboratory).

†† 電子技術総合研究所パターン情報部

s (np (john), vp (v (likes), np (mary)))

は，



以上は標準形であるが，syntax sugar として，precedence 文法が導入されている．構造名を，infix, prefix, postfix のオペレータとして，precedence つきで宣言しておくと，表層ではたとえば，$X+Y*Z$ のような書き方になる．これは入力出力ともそうなる．この構造は，標準形にもどせば，$+(X, *(Y, Z))$ と書かれるものである．

もう一つの syntax sugar はリスト記法である．a, b, c からなるリストは $[a, b, c]$ と書かれる．空リストは[ ]である．tail 部をとりたてて表わす記法として，$[X, \cdot\cdot L]$ の形がある．これは Lisp 流にいえば，X と L の cons である．この書き方だと $[a, b, c]=[a, \cdot\cdot[b, c]]=\cdot\cdot=[a, \cdot\cdot[b, \cdot\cdot[c, \cdot\cdot[\ ]]]]$ である．$[a, b, c]=[X, \cdot\cdot L]$ というマッチングが成立する条件は，$X=a, L=[b, c]$ となる．

Prolog のプログラムは「文」からなる．文は「関係」から構成される．文には二つの形式がある．

（i） 関係0.

（ii） 関係0:−関係1，関係2，…，関係n.

関係は構造で表わされる．文の中で現われる変数はその中で局所的である．（i）は関係0が成り立つことを表わす．（ii）は，関係0が成り立つ条件は，関係1〜関係nがそれぞれ成り立つことであることを表わす．

Prolog でのプログラムの実行は，ある関係を成り立たせることをゴールに，そのための成立条件を次々に確かめていくことである．普通のプログラミング言語的にいえば，関係0が手続きの頭に，関係1〜関係nがその本体に相当する．この手続きの呼出しはパターンマッチングによる．すなわち，変数に適当な値を代入することにより，ゴール（サブゴール）と同一になるような頭部をもつ手続きを探し，それを実行する．

本体の実行の順序は左から右へ，文の探索は上から下へ行われる．成功のパスを先に探索していく．失敗があればバックトラックによってやり直しを行う．

Prolog の基本事項は以上の程度である．そのほかに若干の syntax sugar や100あまりの「組込み述語」が用意されている．組込み機能としては，入出力関係の操作，算術演算等のための述語と組込み関数，特殊制御の機能，高階操作などのメタ論理述語，（内部）データベース操作機能，そのほかがある．詳細はマニアル[2]を参照されたい．

上に述べた基本事項だけでも，リスト処理の多くのプログラムを表わすことができる．簡単な例を一つだけ見ることにしよう．

```
concatenate ([X, ··L1], L2, [X, ··L3]):−
                    concatenate (L1, L2, L3).
concatenate ([ ], L, L).
```

ここで，concatenate (X, Y, Z) は，リスト X と Y をつないだものが Z であることを表わす．たとえば，concatenate $([a, b], [c], [a, b, c])$ が成り立つ．第一の文は，$[X, \cdot\cdot L1]$ の構造のリストに $L2$ をつないだものは，$L1$ に $L2$ をつないだものを $L3$ とすれば，$[X, \cdot\cdot L3]$ であることを表わす．第二の式は，空リストを L につないだものは L そのものであることを表わす．これらは，concatenate の基本的性質を表わしていると考えることができる．

ここで，concatenate $([a, b], [c], X)$ を成り立たせることを考える．そのために上の二つの性質を使っていく（変数は局所的であるから，適当に名前を変更してよい）．

```
concatenate ([a, b], [c], X)
⇐concatenate ([a, ··[b]], [c], X)
⇐concatenate ([b, [c], X1), X=[a, ··X1]
⇐concatenate ([ ], [c], X2), X1=[b, ··X2]
                              , X=[a, ··X1]
⇐X2=[c], X1=[b, ··X2], X=[a, ··X1]
⇐X=[a], ··[b, ··[c]]]
⇐X=[a, b, c]
```

となるから，$X=[a, b, c]$ とおけば最初のゴールが成り立つことがわかる．すなわち，答え $[a, b, c]$ が計算されたことになる．

3. Prolog の背景

Prolog の背景には，（一階）述語論理における定理証明の研究がある．Robinson によるレゾリューション原理の提案[3]以来さまざまな証明戦略の研究が行われて来た．レゾリューション方式では，論理式をある標準形に直し，そこで拡張された三段論法を行う．Prolog では，その標準形をそのまま用いる．正確にいえば，その部分集合（Horn 集合と呼ばれる）を用いる．この部分集合の中では，ある特殊な（効率的な）証明手

続きが完全であることが証明されている．Prologに使われているのは，そのうちのSNL (selective negative linear input resolution) である．このSNLの動作は前章で述べたようなものであり，普通のプログラムの実行の過程とよく似ている．そこで証明＝計算と見立てられるのである．

これと似た提案に，Plannerを始めとする「人工知能用プログラミング言語」がある．Plannerは，論理を改作してプログラミング言語化したものという説明がなされた．PlannerとPrologは外見上非常に似ている．しかし，第一に問題意識の差が指摘できる．人工知能用言語派は，論理から離れようとした．論理プログラミング派は，論理にできるだけ忠実であろうとした．このあたりは面白い論点であるが，深入りせず興味深いエッセー[4]を紹介するにとどめよう．

第二の点としては，現実的問題として，処理系の作製法の差を挙げることができる．Plannerなどの人工知能用言語はLispの上のインタプリタとして実現された．一方，Prologは，各種の技法を駆使して，直接的に実現されている．最近ではコンパイラも作成されている[5]．Plannerがその後伸びなかったのには原理的，思想的な背景もあるが，処理系の遅さにも大きな原因があったと考えられる．この点，PrologはLisp自身の速度に匹敵することが示されている[5]．

Prologは最初，マルセイユ大学を中心にFORTRANの上で実現された．このマルセイユ版でも，多くの人の予想以上の速度が得られている．その後，DEC system 10の上で作られたエジンバラ版では，上述のようにLispに相当する速度が得られるとともに，プログラミング・システムとして，多くの機能が追加されている．そのほか，ウォータル大学版など，効率的な処理系の実現が伝えられている．

4. Prologの特徴と問題点

Prologをプログラミング言語として見たときの特徴としては，

(i) パターン・マッチングによるリスト処理

(ii) パターン・マッチングによる手続きの呼出し

(iii) 非決定性動作（バックトラッキングによる）

を挙げることができよう．（これらはPlannerにもあてはまる特徴であるが．）

(i)や(ii)はリスト処理言語の高機能化の方向として，それ自体で評価しうる項目である．Prologが論理に忠実であろうとした結果の特徴として，さらに，

(iv) 論理変数の機能の活用

(v) 入力変数，出力変数の区別をしないこと

が挙げられよう．(iv)には，適切な例題が必要であろうが，概念的にいえばdelayed evaluationの機能が自然に実現されるということである．(v)は，論議のあるところだが，有効な場面もある．実際には，コンパイルド・コードの効率化のために，入力変数，出力変数の指定が再導入されつつある．

普通のPrologでは(iii)が基本であるが，非決定動作は，コルーチン機能[6]や並列動作と結びつけて考えることができる．この方向での拡張も試みられつつあるが普通には(iii)をベースにした拡張が行われている．

普通のPrologでは，バックトラックによる縦型探索が固定されている．この上で「副作用」のある操作が拡張として導入されている．この点でよく論議を呼ぶものにcut操作がある．これは，この時点でバックトラックの可能性をキャンセルするものである．それ以降のバックトラックが全部失敗に終ることが判っている場合にはこれは効率を上げるだけである．しかしこれは副作用を導入する可能性がある．実際には，むしろその副作用を利用した「巧みな」プログラミングが行われている．しかし問題は，このcutの論理的正体が不明なことである．そのため，これはPrologのFORTRAN化であるという批判もでてくる．

副作用のほかの例としては，入出力操作，データベース操作がある．これらの副作用のある操作は，実際面で便利であるにしても，その論理的性質の再整理が必要であろう．

一つの可能性は，これら副作用のある操作を様相操作とみたて，様相論理的整理を行うことであろう．

実際のPrologには，高階の操作も含まれている．このことでLispのように，データをプログラム（あるいはその逆）として扱えるようになっているのであるが，この点の論理的再整理も必要であろう．高階のHorn集合の論理的性質は何かという理論的課題がある．

PrologはHorn集合をベースにしている．ある種の組込み述語を仮定すれば，Horn集合で帰納的関係（関数）が表現可能であるか，Hornから一般への再拡張は意味があるか，必要か，あるいはその時の問題点は何か，という課題もあるであろう．

5. おわりに

以上で Prolog の概要，特徴，問題点のいくつかを述べた．この言語にはよい処理系も作られ，実際の使用に耐える状況になっている．いろいろな応用システムも作られるようになってきている[10]．これを使いこなして，経験をつむことが一つ望ましいことであろう．

前章でいくつか述べたような拡張方向や問題点があり，これからも多くの研究が望まれるところであるがその健全な発展には，経験の蓄積をベースにした理論的な整理が必要であろう．

論理プログラミングには，最初に述べたように，ソフトウェア工学における他分野との融合の期待がある．たとえば，形式仕様からプログラム自身まで，同一の論理系（部分系を用いるにしても）の枠内で考えうることは，1段階的改良における source to source の変換の実験の場として好適であろう．

他方ではデータベース問合せ言語との統一化がある．これは，内部データベースのレベルでは Prolog において実現されていることであるが，外部データベースとの関連ではどうなっていくであろうか．

また，Prolog のような言語を，もっとエンドユーザ向きに改作するとすればどうなるであろうか．

Prolog などの論理プログラミングは，これまでのところ，ヨーロッパ（東欧も含めて）や英国で盛んである．米国でも再評価の機運が出てきている．我が国でも関心をもつ向きが増えているが，今後大いに検討されてよい項目だと考えられる．

参考文献としては直接引用したもののほか，Prolog や論理プログラミングの状況を知るためのものをいくつか含めた．7）はエキスパート・システムとの関連での解説が含まれている．8）は論理プログラミングの良い解説である．9),11）では Prolog の理論的基礎が述べられている．自然言語処理への応用としては 11),12）がある．これらには Prolog の概説も含まれている．

Prolog 関連，論理プログラミング関連の研究の現状を見るには 13）がよい．これは昨年ハンガリーで開かれたワークショップの記録で 42 の論文が収められている．

参考文献

1) Kowalski, R. A.: Predicate Logic as Programming Language, Proc. IFIP 74, pp. 569-574, North-Holland (1974).
2) Pereira, L., Pereira, F. and Warren, D.: Users Guide to DEC system-10 Prolog, Dept. AI Research, Edinburgh Univ. (1978).
3) Robinson, J. A.: A Machine-Oriented Logic Based on the Resolution Principle, JACM, 12 (1965).
4) McDermott, D.: The Prolog Phenomenon, ACM SIGART Newsletter, No. 72 (1980).
5) Warren, D., Pereira, L. and Pereira, F.: PROLOG: The Language and its Implementation compared with LISP, Proc. Symp. on AI and Programming Languages, ACM SIGART/SIGPLAN (1977).
6) Clark, K. L. and McCabe, F. G.: The Control Facilities of IC-PROLOG, in 7).
7) Michie, D. (ed.): Expert Systems in the Mirco Electronic Age, Edinburgh University Press (1979).
8) van Emden, M. H.: Programming with Resolution Logic, in Elcok, E. W. and Michie, D. (ed.): Machine Intelligence, vol. 8 Ellis-Horwood (1977).
9) van Emden, M. H. and Kowalski, R. A.: The Semantics of Predicate Logic as a Programming Language, JACM, Vol. 23 (1976).
10) Bundy, A., Byrd, L., Luger, G., Mellish C. and Palmer, M.: Solving Mechanics Problems using Meta-Level Inference, in 7).
11) Colmerauer, A.: Metamorphosis Grammars, in Bolc, L. (ed.): Natural Language Communication with Computers, Lecture Note in Coupruer Science, No. 63, Springer (1978).
12) Pereira, F. and Warren, D.: Definite Clause Grammars Compared with Augmented Transition Networks, Dept. of AI, Univ. of Edinburgh (1978).
13) Tarnlund, S-A (ed.): Proc. Logic Programming Workshop (1980).

(昭和 56 年 4 月 27 日受付)