人工知能と対話技法 20－4
（1981. 3. 20）

# 知識工学用マシンの可能性

渕 一博（電子技術総合研究所）

はじめに 知識工学を今後さらに発展させるさいの問題点として、Feigenbaumは6つの項目を立てて論じている[1]。その中で「適切なハードウェアの欠如」を第一に挙げていることが興味深い。これまでの知識工学システムは従来型コンピュータの上に（ソフトウェアを介して）実現されてきたが、いまや、「それは誤りである」と彼は断言する。

知識工学に適した高性能の記号処理マシン（具体例としてLISPマシン）の出現に期待するわけであるが、その背景には、知識工学システムが実用を目指して大規模化していることがある。ハードウェア価格の低下は、その解決要因であるが、一方では、記号処理専用マシンの作り方についてのノウハウの蓄積がある。

LISPマシンへの動きは、米国だけでなく我国でも活発化している。また最近、LISPマシンと関連研究のサーベイが出された[2]が、これはドイツで行われたものである。世界的に関心が高まっていることが察せられる。

ここでは、サーベイ風にではなく、いくつかの論点をひろって、知識工学用マシンの可能性を考えてみることにする。

ベース言語の選択 知識工学（人工知能）の研究の中で、知識表現用あるいは問題解決用としていろいろな言語が提案されている。それらは、直接マシン語の上に実現されるというより、ベースとなる別の（記号処理用）高級言語の上で作られるのが普通である。そのベース言語としてはLISPが用いられることが多い。

ベース言語は、他のさらに高級な機能の実現手段として十分な機能をもっていることが必要である。それとともに、今の場合は、新しい（ハードウェア）マシンを考えようとするのであるから、その観点からもよい特性をもつものでなければならない。

過去の実績からすれば、LISPが第一の候補であることを認めてよい。それとともに、ここでは、PROLOGをその有力候補として挙げたい。PROLOGはLISPに比べれば伝統と蓄積が少ないが、優れた特性をもっている。

PROLOG LISPは関数型の言語であるが、PROLOGは述語論理をベースにしている。それは、かつてのPLANNERに似ているが、それを論理の側にもう一度ひき戻したものである。人工知能用言語といわれたPLANNERやCONNIVERなどはLISPの上で実現されたがPROLOGは直接実現されている。

PROLOGとPLANNERの比較について、McDermottが興味深いエッセーを書いている[3]。PROLOGは欧英を中心に展開されてきたが、米国では、それをPLANNERの遅れた発見と見る向きが多かった。その見方は変えた方がよいというのである。

PLANNERやCONNIVERなどは、問題解決用言語と把えられていた。PROLOGはむしろ「プログラミング言語」と見立てられ、直接実現された。それは他のもっと高度の問題解決システムのベース言語として用いられている。この位置づけの差が重要だというのである。

PROLOGとLISPの比較論はあとで行うことにして、PROLOGの簡単な紹介をしておこう。

例として、二つのリストをつなげるappendのプログラムを考える。PROLOGでは、

```
append([],Y,Y).
append([A|B],Y,[A|C]):-append(B,Y,C).
```

と書かれる（エジンバラ大版[4]）。ここで大文字は変数を表わす。[]は空リストを、[A|B]は、AとBをconsしたリストを表わす。append(A,B,C)はリストAとBをappendした結果がCであることを表わす。第一の式は空リストをappendしても結果は変らないことを表わす。第二の式は、[A|B]の構造のリストをYにappendした結果は、BにYをappendしたものをCとすると[A|C]というリストであるということを表わしている。これらはappendの基本的性質の記述であるが、それとともに、プログラムとして働かせることができるのである。たとえば、

```
append([a,b],[c],X)
:-append([b],[c],X1),X=[a|X1]
:-append([ ],[c],X2),X1=[b|X2],
                      X=[a|X1]
:-X2=[c],X1=[b|X2],X=[a|X1]
:-X=[a|[b|[c]]]
:-X=[a,b,c]
```

となって結果が得られる。この過程はリスト処理の過程であるが、レゾリューション（三段論法）を進める過程でもある。推論（証明）のステップが計算過程に対応するのである。レゾリューションを行うときの変数の対応をユニフィケーション（同一化）と呼ぶがこれは、パターン照合の一種でありリストの分解合成を統一的に行うものである。なお対応するLISPプログラムは、

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

PROLOGとLISP　プログラム例によって、PROLOGとLISPの感じはつかめると思われるが、いくつかの点で比較をしてみよう。LISPは関数型であり、PROLOGは述語（関係）型である。このことから

- パターン照合によるリスト処理と手続きの呼出し
- 非決定性動作

などの機能がPROLOGにつけ加わる。これらの機能は、LISP自身の拡張方向として考えられたことのあるものである。

どちらについてもインタープリタやコンパイラが作成されており、同じ問題については同程度の性能を出している。（この点、PLANNERはLISPでインタープリートされていたため、格段に遅かった。）PROLOGはLISPに比べ機能拡張になっているが、その実現法はLISPに比べやや複雑なだけであり、多くの手法は共通である。しかも性能低下をもたらさない。

知識工学システムでは、プロダクション・システムが使われることが多い。これも普通LISP上でインタープリートされている。ところでPROLOGもプロダクションシステムの一種と見ることができるが、直接実現されているため、性能は1桁以上良いことになろう。

同じことはパーザについても言える。PROLOGでは、文法記述がほとんどそのままPROLOGプログラムになるので性能の高いパーザがひとりでに得られることになる。

LISPとPROLOGについて、エジンバラ大における教育実験の例がある[3]。POP-2（LISP相当）とPROLOGの二つに学生たちを分けて試みた結果、2ケ月のコースのあと、学生たちは、POP-2コースでは、パターンマッチャを作っており、PROLOGコースの学生は自然言語質問応答システムを作って

いたという。PROLOGにはパターン照合が組み込みになっているからである。

パターン照合や非決定性動作が、知識工学にとっていずれ不可欠の要素であるならば、それらを含んだベース言語の方が望ましいであろう。

自然言語への応用例[5]では、PDP11上でのインタープリタで、PROLOGの方がLISPよりシステムがコンパクトになり、大きな機能を実現できると報告されている。

PROLOGはLISPに比して、伝統と蓄積において劣るであろう。しかし優れた特性をもっており、知識工学用のベース言語、そして新しい記号処理マシンのベース言語として、LISPより良いのではないかと思われる。

マシン化の方式　コンピュータのアーキテクチャについては古くから各種の提案がある。いわく、タグマシン、スタックマシン。あるいは連想マシン、並列マシン。これらの提案は、現在主流のアーキテクチャを越えようとするものであるが、これまでのところ必ずしも実を結んでいない。

それは個々の提案の良否ではなく、それらを総合的に活かす指導原理が欠如していたことにあろう。もちろん、部品技術等の状況の時代環境の問題もあるが。

そのような指導原理として「高水準言語マシン」の提案も古くからある。しかし、採用言語がFORTRAN, COBOL---では格別有利とはされていないようである。それは一つには、それらの言語自身が、従来型コンピュータを前提にした言語だからであるだろう。PASCALマシンやADAマシンではどうであろうか。

前提が既存マシンのアーキテクチャと無関係な方が、専用マシン化の効果が大きいであろう。その点LISPやPROLOGでは、専用化や新アーキテクチャへの指向が強くなるだろう。

LISPやPROLOGは、タグ方式やスタック方式など、改良的な手法の統合化にも指導原理を与える。それとともに、データフロー・マシンのような革新的手法にも指導原理を与える点がさらに魅力的である。

データフローマシンが注目される一因に関数型言語(LISPもそれに属する)との深い結びつきがあることは、いまや広く認められている。PROLOGの場合も、データフローマシンと結びつきがある[6]。また、データフローをコンストレイント型に拡張するような方向は、関数型から関係型への拡張と自然に対応している。

マシン化による効率化　データフローマシンとの対応などは、これからの興味深い研究テーマであると思われる[7]。ここではその指摘だけにとどめ、これからの議論は、さし当りの改良的なアーキテクチャの効果に限ることにしよう。ここで改良的アーキテクチャといっているのは、タグやスタック、あるいはマイクロプログラミングその他、これまでも使用されたような、現在実施可能な手法の総合援用にもとづくものを言っている。部分的に並列性をもたせるにせよ、基本的には逐次型制御にもとづくものである。

このような改良型という枠の中でもLISPやPROLOGについてはマシン化の効果が非常に大きいと考えられる。

MITのCONS-LISPマシンでは、DEC KA-10と比べ、TSの遅れなども入れると、3倍(MACLISP), 6~12倍(INTERLISP)に速度向上する。ここでKA-10は30万ドルマシン、CONSマシンは8万ドルである[2]。

CADR-LISPマシンはさらに性能向上しているであろう。

神戸大学のLISPマシンの試作例では（試作費は数百万円といわれているが）、その速度は、大型機HITAC 8800やACOS800上のインタプリタの3～4倍，それらのコンパイルドコードの1/2～1倍になっている[8]。使用条件その他で単純な比較はいけないだろうが、適切な構成のマイクロプログラミングマシンにおけるファームウェア、インタープリタの効果は顕著である。

PROLOGマシンの実例はまだないがそれに近い例として、ユニットレゾリューションのマイクロプログラム化の例を見てみよう。[9]（PROLOGはinput linear resolutionをベースにしている。）ここで用いられたTPUはLISP評価の例題にもなっている。

UNIFICATION, RENAME, MOVE, SUBSTの関数だけをマイクロプログラム化し他はアセンブラコーディングしたものであるが、その効果は著しい。

たとえばTPU-6は376msになっているが、これは神戸大マシンでは、6.728ms，HLISP, OLISPのインタープリタで約20,000ms，コンパイルドコードで約4,000msである。

大まかにいうと、インタープリトからコンパイルでほぼ1桁上るが、この関係が何重にも効いてくるということである。

知識工学用マシンのストーリ　以上のような考察から，知識工学用マシンの理想像を描けば次のようになろうか。

ベース言語としては、PROLOG（の拡張版）を用いる。これには、LISPやプロダクションシステムの基本的機能を整理してとり入れる。

次に、これを機械語と考え、タグ、スタック，ハッシュ等，これまで提案された着想を十分にとり込む。さし当りはマイクロプログラミングが主体になるだろうが、効果の大きい部分は回路化も考える。このマシンは、いずれVSLI化されるだろう。そうすれば、現在，従来型大型機で実現できる以上のことが、パーソナル化されたマシンで実現されるであろう。将来はさらにデータフローやその他の革新的アーキテクチャをとり入れた、スーパ記号処理マシンを考える。

記号処理マシンの汎用性　上で描いたような記号処理専用マシンは、本当に「専用」であろうか。理論的な意味では、これもまた汎用マシンである。それとともに、将来の情報処理の展開分野を考えれば、その中核的なマシンとも考えられよう。

参考文献


1) E.A. Feigenbaum: Knowledge Engineering-The Applied Side of Artifical Intelligence (1980)

2) H. Boley: A Preliminary Survery of Artifical Intelligence Machines, ACM SIGART Newsletter (NO.72, July 1980)

3) D. McDermott: The PROLOG Phenomenon, ACM SIGART Newsletter (NO.72, July 1980)

4) L.M. Pereira et al: User's Guide to DEC-10 PROLOG (Sept. 1978)

5) G. Silva et al: Toward a PROLOG Text Grammar, ACM SIGART Newsletter (NO. 73, Oct. 1980)

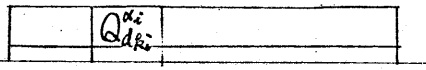
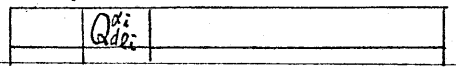
6) P. Morris: A Dataflow Interpreter for Logic Programs, Proc. Logic Programming Workshop (July, 1980)

7) 雨宮他：リスト処理向きデータフローマシンの検討，記号処理研究会資料 13-3 (1980年10月)

8) 瀧他：LISPマシンの試作－インタープリタの構造とシステムの評価，情報処理学会論文誌 Vol.20 NO.6 (1979)

9) 横井他：定理証明のファームウェア化，情報処理学会第16回大会 (1975)

# 正誤表

| 頁　行 | 誤 | 正 |
|---|---|---|
| 5頁　5行目 | (SIN z:x) | (SIN z:$\underline{x}$) |
| 23行目 | [PASS $X^i$ #INSPEC-A) | [PASS $x^i$ #INSPEC-A) |
| 6頁　12行目 | 中集合. | 中集合の場合、 |
| 14行目 | $(\forall Y_2^{i+k-2}/Y_1^{i+k-1})\cdots(\forall y^i/Y_{k-1}^{i+1})$ | $(Q_2 Y_2^{i+k-2}/Y_1^{i+k-1})\cdots(Q_k y^i/Y_{k-1}^{i+1})$ |
| 30行目 | (　)…(　)[FGET T:$X_1\cdots X_m$) | (　)…(　)[FGET T:$x_1\cdots x_m$) |
| 37行目 | その役割は | その役割は |
| 7頁　35行目 | (　)=(　)$(Q_{bki}^{\alpha_i-1} x_{ki}^{S_i+\alpha_i-\alpha}/x_{ki}^{S_i+\alpha_i-1})\cdots$ | (　)=(　)$(Q_{bki}^{\alpha_i-1} x_{ki}^{S_i+\alpha_i-1}/x_{ki}^{S_i+\alpha_i-1})\cdots$ |
| 8頁　1行目 | $(Q_{bki}^r, Q_{dli}^r)\neq$(　) | $(Q_{bki}^r, Q_{dli}^r)\neq$(　) |
| 9行目 | $Q_{dli}^r \Rightarrow \forall$ | $Q_{dli}^r = \forall$ |
| 表 |  |  表1 |
| 20行目 | P:(　)…$(Q_{kb}^* x_{kb}/\!/X_{kb})$[　] | P:(　)…$(Q_{kp}^* x_{kp}/\!/X_{kp})$[　] |
| 21行目 | Q:(　)…(　)[　] | Q':(　)…(　)[　] |
| 27行目 | (　)=(　)(　)…$(Q'_{rmt} z_{mt}^{St}/\!/z_{mt}^{St+1})$ | (　)=(　)(　)…$(Q'_{rmt} z_{mt}^{St}/z_{mt}^{St+1})$ |
| 36行目 | のものを引きつく. | のものを引きつぐ. |
| 39行目 | $\beta_i$ | $\beta_j$ |

註　(　)内空白は詳細省略