解　説

記号処理専用マシンへの期待†

淵　一　博††

1. まえがき

いまのコンピュータが記号処理マシンであることはいうまでもない．しいていえば，記号処理的「数値演算」に片寄っているということであろう．そういえば次には，記号処理らしい記号処理とは何かということが問題となるだろう．そのとき，「記号処理用」プログラミング言語と呼ばれるものに具現されている機能がそれであるといってよいだろうが，トートロジ的表現のようでもある．

もっと根源的には，応用的な（高次応用的な）側面から要求される記号処理機能とは何かを考えるべきであろう．それが，（記号処理用）プログラミング言語への要望を介して，さらには，コンピュータアーキテクチャへの要望につながっていくことが考えられる．

ここでは（現場的応用ではまだないが）応用的な観点から，いわゆる知識工学，自然言語処理を取り上げてこの問題を考えてみることにする．ただし，客観的公平な記述というより，（いまの段階ではまだ）個人的な好みに片寄った記述になることをお断りしておきたい．

2. 知識工学的視点

知識工学は「応用人工知能」ともいわれている．ただ，人工知能といえば範囲は広く，あとで取り上げる自然言語処理も（まだ）人工知能研究の中に含まれるのが普通である．狭い意味では，エキスパートシステムと呼ばれるように，ある専門領域を設定して，その領域での知識を整理して，一種のコンサルテーションシステムを構築することを指している．

著名な例としては，医療分野における診断，投薬助言のシステムがある．

専門領域の知識が取り上げられるのは，常識と呼ばれるような一般的な知識に比して，整理が容易であるからである．といってもそれは相対的な話で，専門領域の（コンピュータ向きの）整理が必ずしも容易であるわけではない．むしろ，知識工学の（当面の）眼目は，そのような整理の手法を確立することと，それを実践することであろう．

知識工学の主眼がそうであるにしても，それを支えるツール的視点が不用なわけではない．既成のツール（ソフトウェア，ハードウェアを含めて）が十分であるわけではない．知識工学の場合も，その進展はツールの進歩とほぼ並行しているといえる．

知識工学のためのプログラミング言語としては，プロダクションシステム（PS）と呼ばれる言語がよく使われている．もっともそれに限るわけでなく，フレーム的言語とか意味ネットワーク的言語も試みられている．しかしここでは PS を知識工学におけるいわば標準的なものと（かりに）考えて話を進めることにする．

いま（小さな）データベースを考え，それに対する操作規則を考えてみる．そのような規則の一般形は，

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

のようになるであろう．ここで $A_1, \cdots, A_m$ はデータベースについて成り立つ条件である．$B_1, \cdots, B_n$ は，条件 $A_1, \cdots, A_m$ が成り立ったとき，データベースにほどこす操作（の系列）である．その操作は，ある言明をデータベースに追加したり，変更したり，あるいはデータベースから削除するものである．

そのような規則の集まりが，PS における「プログラム」である．

このような PS の処理系は，直接実現されるというより，むしろ他の言語，たとえば Lisp の上で実現されているのが普通である．となると，PS は知識工学的システムの作製者にとってのユーザ言語であるが，より基本的な言語は Lisp ということになろうか．PS が Lisp に埋め込まれるのは，Lisp 処理系がすでに整備されていて他の多くの機能をもっているからで

† Expectations to Dedicated Symbol Manipulation Machines by Kazuhiro Fuchi (Institute for New Generation Computer Technology).

†† (財)新世代コンピュータ技術開発機構

ある．別の道として，PS 処理系に，Lisp 処理系のもつ機能を吸収することが考えられなくもない．ただそのためには，逆転を正当化する論理が必要であろう．PS 自身はその道は歩んでいない．しかし，それに近い線にあるプログラミング言語として後述する Prolog がある．Prolog は PS に近いが，独立したプログラミング言語である．ある意味で Lisp の拡張にもなっているが，言語としてはより単純である．処理系も Lisp に比べてそう複雑になるわけではなく，また速度もそう落ちないことが示されている．

3. 自然言語処理の視点

自然言語は，コンピュータとの会話形態として一つの理想として語られている．ただ，自然言語の現象は複雑多岐にわたり，言語学でもそのすべてが押えられているわけではない．そこで，自然言語の有用なサブセットを設定することが提唱されている．適切なサブセットが設定できれば，それは，問合せや指令の言語あるいはプログラミング言語として使われるだろう．

しかし，あまり単純化したサブセットでは「自然」言語ではなくなるだろう．自然さを保つためには（ユーザに不自然さを感じさせないためには）相当の機能が要求されるだろう．そして一方，そのような（サブセット）自然言語の処理コストがリーゾナブルであることが要求されるだろう．それは必ずしも容易でないにしても，いずれ実用の段階に至ることは不可能ではない．そのためには，自然言語自身をもっと研究することが必要だが，それとともに適切なツール系の設定が（ソフトウェア，ハードウェアを含めて）必要である．

自然言語の処理には，テキスト的処理，辞書的処理だけでなく，構文的処理，意味的な処理が（いずれ）必要になってくる．辞書的な処理には，データベース技法が有用，必要になる．このことは，データベース技法を記号処理的視点から把える一つの動機づけになるであろう．

自然言語の文法を記述するのにいくつかの様式が考えられているが，多く用いられるのは句構造文法である．そこで用いられるのは「書きかえ規則」である．たとえば，

A⇒B C

は，Aという文法カテゴリは B C という並びに書きかえられる（書きかえてよい）ということを表わす．これは実は，プロダクションシステムの一種である．PS はその一般的な形であるが，PS を文法と見立てるためには，そのデータベースに線状の並びの構造を導入しなければならない．そこで（句構造）文法は PS の特殊形とみることができる．

構文的な処理というのは，そのような文法に基づいた処理である．自然言語処理には構文解析システムが必要な一要素になる．ただそれは独立したシステムではなく，他の処理と自然に接合するものでなければならない．そのような観点からすると，成功したものの多くは Lisp の上で作られている．Lisp ではそのような機能を比較的自然に埋め込めるからである．

構文解析システムを単独に作るだけならば，たとえば Fortran や PL/I で作ってもそう難しくはない．しかし，他の機能との接続を考えるとそれは良い選択ではない．その上，Lisp ベースの方が速度においても速いことが多いのである．それは Lisp が（他の言語に比べて）良い記号処理機能（と能力）を持っているからである．

Lisp は自然言語処理にとって良い選択であるが，しかし唯一最良の選択とはいえないだろう．テキスト（文字列）に重点を置いたとき，Snobol も良い選択である．しかし，文字列処理から出発した言語という宿命をもっているように思われる．筆者は，自然言語処理にとって，Lisp より Prolog の方が良い選択であろうと思っている．また，前述した知識工学にとってもそれが良い選択であると思っている．自然言語処理がこれから意味的な処理に入っていくとするとそれは，知識工学の手法と大きく重なってくる．その点からも両者に共通した良い選択が求められる．

4. Prolog—論理プログラミング

Prolog も記号処理言語の一種である．これについては別途（簡単に）紹介したことがあるが，ここではさらに簡単に一例によって紹介することにする．Prolog（エジンバラ大学版）では，リストは [a, b, c] のように書かれる．X, Y を変数としたとき $[X, \cdots Y]$ は X と Y の cons であることを表わす．Prolog の基本操作の一つはマッチング（ユニフィケーション）である．

[a, b, c]=[X, ⋯Y]

というマッチングが成立するのは，

X=a, Y=[b, c]

のときである．なお空リストは [ ] と書かれる．

いま文法規則，

A⇒B C

があるとする.

語の並びをリストで表わし，それをLとする．その頭の何語かがまとまりとして文法カテゴリAを満足するということを表わすのに，残りの部分を L1 として，a(L, L1) と書くことにする．B, C についても同様の表わし方をするとすると，上記の文法規則は，Prolog では，

a(L, L1) :−b(L, L2), c(L2, L1).

と書かれる．これは，a(L, L1) が成り立つためには左辺の b(L, L2), c(L2, L1) がそれぞれ成り立たなければならないことを表わす．

A⇒w

を，Aは語wであってよいという文法規則とすると，これは Prolog では，

a([w, …L], L).

と表わされる．

このように文法規則は簡単な対応で Prolog のプログラムに移される．いま，

文⇒主 述

主⇒dog

主⇒cat

述⇒run

という文法があるとしよう．これに対応する Prolog プログラムは，

文(L, L1) :−主(L, L2), 述(L2, L1) (1)

主([dog, …L], L). (2)

主([cat, …L], L). (3)

述([run, …L], L). (4)

となる．[cat, run] が文であることを確かめるには，文([cat, run], [ ]) が成り立つかどうかを確かめればよい．(1)より，

文([cat, run], [ ])

⇐主([cat, run], L2), 述 (L2, [ ])

ここで第1の条件は，(3)により主([cat, run], [run]) として成り立つから，

⇐述([run], [ ])

これは(4)によって成り立つ．したがって [cat, run] は文である．

Prolog はその他のリスト処理や計算を表わすことができる．そこで単純な（句構造の）文脈自由文法の解析だけでなく，それを拡張した文法処理や意味処理を追加することができる．逆にいえば，文法規則は Prolog に簡単に埋め込むことができる．

この埋込みは Lisp におけるより単純である．このことは Prolog の方が自然言語処理に適しているということの（簡単な）一例になっている．Prolog は，

（i） ユニフィケーションによるリスト処理

（ii） 非決定動作

などの特徴をもち，Lisp の拡張と考えることができるのである．

上は文法規則に対する例であったが，文法規則はまた PS の特殊例であった．このことを考えると PS と Prolog には同様の簡単な対応がつく．文法規則や PS では(ii)の非決定動作が本質的に含まれているからである．

これだけをもって Prolog の優位性を主張するのに不十分なのはもちろんである．Lisp は長い歴史をもち，処理系がよく整備され，また Lisp 上のソフトウェア蓄積も多い．Prolog はそれに比べると未成熟である．現時点における現実的優位性は Lisp にあるが可能性，将来性は Prolog にあると筆者は考えている．

Prolog は，述語論理をプログラミング言語とみなそうという考えに基づいている．述語論理は，形式論理の諸体系の中ではもっとも古く確立された，いわば標準的な論理といってよい．その源の1つは自然言語の形式化にあるが，一方数学的体系およびそこにおける論証の形式化もその源泉の1つである．その意味では一番自然な論理といってよいであろう．

述語論理は計算機科学の諸分野で現われる．プログラムの検証の表現形式には（証明という性質上）それがもっとも多く使われている．プログラムの仕様記述自体も（表層表現はともかく，そのベースに）述語論理を用いることが多い．

述語論理は関係の論理ということができる．関係データベースの基礎理論は述語論理に由来している．それらを考えると，計算機科学の諸分野の統一的な表現のベースとして，述語論理を見直してよいと思われる．プログラミング分野でのその考えの適用を，(述語) 論理プログラミングといってよいであろう．Prolog は，論理プログラミングを実践するプログラミング言語の一例と考えられる．

より応用的観点からは，数式処理の分野がある．数式処理は高次記号処理の代表的分野とされている．数式処理システムには Lisp をベースにするものが多いが，Prolog はそれにより適した特性をもっていると思われる．

5. 記号処理マシンへの期待

これまで，記号処理の高次応用の例と考えられる分野での，ベースとなるプログラミング言語について考えてきた．これらの分野は今後，計算機応用において比重を増してくるものと考えられる．そこにおけるプログラミング言語として，Snobol や Lisp は優れた言語である．それとともに今後の発展性から Prolog が注目される．

それらの言語が想定する応用分野が拡がり，したがってそこで実現される機能が大きくなるにつれて，その処理系の性能の問題が大きくなってくる．

これらの言語は従来主として，普通のタイプのコンピュータを前提として，その上でソフトウェアによって実現されてきた．ソフトウェア的実現技法が種々考案され，かつ実効も上げてきた．たとえば Lisp 処理系について（Snobol でも同様だろうが），十数年前の処理系と比べるといまのものは1桁以上（場合によっては2桁）の性能向上がみられる．ベースになっているハードウェアの進歩を相乗すれば大幅なパワーアップになっている．一面ではそれが，たとえば知識工学の実用（それがまだ初歩的段階であるにしても）をもたらしたといえる．

それでは，今後のハードウェアの進歩を見込めば，（ソフトウェア的技法の改良もまだ幾分はあるとして）それでよいのであろうか．たとえば知識工学の場合，その実用可能性（有用性）が示されると，次にくるのがより広い領域での適用，より高機能の実現に対する期待である．それはより大きなコンピュータパワーへの要求をもたらす．しかし，知識工学的応用の趣旨からすれば，それは超大型機の上でやっとできるというだけでなく，分散化されたパーソナル機の上でも働くというのが本来のイメージ（コンサルテーション・システムとして）であろう．

このことは，自然言語の実用的使用についてなおあてはまるであろう．自然言語が人との会話の形態である以上，それはフロントエンドでそれぞれ十分働くものでなければならない．しかし本格的な自然言語処理は現状では大型機を必要とする．自然言語処理の今後の研究はより高次の機能の実現へと向うであろうから，ますます大きなパワーを必要とすることになろう．技法の改良による省パワー化より，需要の方が大幅に上回るであろう．

知識工学，自然言語処理の将来の実用イメージを考えると，その時代のパーソナル・マシンは，現在の超大型機のパワーを必要とするだろう．これは，素子技術に基づく，ハードウェアの単純な進歩（高速化，小型化，低価格化等）を待つだけでよいのであろうか．それしか道がないのであろうか．

ここに，記号処理「専用」マシンへの期待がある．それは，Lisp マシン，Snobol マシン，あるいは Prolog マシンと考えてよい．それに限るわけでもないにしても，何らかのプログラミング言語を想定しないマシンは考えられない．ユーザ向言語へ向っての工夫（シンタクス的シュガ，あるいはセマンティックシュガ）はいろいろこれからも考えられるにしても，ベースとなる言語（論理系）にはそう多くの選択があるわけではない．論理系として（やや狭くは）関数型があるが，Lisp は関数型言語の一例である．述語型の論理に基づく一例が Prolog である．

Lisp や Prolog について，その処理系が純粋にソフトウェア的技法の改良によって改善されているにしても，それはいずれ限度に近づくであろう．それは（とくに Lisp について）今後1桁近くも向上するとは考えにくい．その限度は見えてきているといってよいであろう．となると，アーキテクチャ的工夫によって，（専用マシンを考えることによって）ハードウェアを含めた処理系の向上の可能性があるかどうかということになる．

その可能性があるとすれば，それは記号処理応用として想定される分野の実用化を大幅に加速するであろう．

Lisp マシンについては，その製品化がすでに始まっている．それは，記号処理専用マシンとしては第一世代であるが，その時代が始まろうとしているわけである．観念的な可能性の論議の時期はすでに過ぎている．Lisp マシンの現状については，本特集号で別途取り上げられている．しかし，ここでは観念的な議論をもう少し続けることにする．

6. 記号処理マシンの可能性

記号処理専用マシンというものを想定するとき，その展開の可能性には2つの軸があるように思われる．

その1つは，逐次型処理，あるいは単一プロセッサという枠内での可能性である．

もう1つは，並列処理への可能性である．

前者の枠で考えるとき，これまで行われてきた（ソフトウェア）処理系の作製法の研究がベースになるで

あろう．それを加味したアーキテクチャ上の可能性については具体的には本稿に続く，それぞれの解説で述べられている．

専用マシンの可能性というのは，インタプリタ対コンパイラの対比に似ている．専用マシン自身，回路によるインタプリタだからである．コンパイルドコードの速度は，インタプリタを介する場合に比べて数倍から数十倍になる（のが普通である）．

そこで第1近似として，専用マシンの効果はその程度と考えられる．それでは，コンパイラを介せば専用マシンが不用になるだろうか．この問題には計算機の使用形態の問題がかかわってくると思われる．これからますます，会話的使用法が拡がっていくだろう．そのような形態には，インタプリタの方が性格が合っている．コンパイルというのは，その変換を，バッチ的に行う性格をもっている．コンパイルのそういう性格のほか，コンパイルという処理自体オーバヘッドである．問題によってコンパイル時間と実用時間の比が異なるが，多数回繰り返して使うものでなければそのオーバヘッドが大きい．またコンパイルして数十倍の速度を得るのは理想に近く，それを実現するような最適化を含むコンパイルには大きな時間がかかるといってよい．

専用マシン化のためにはより多くの回路がいるが，一方，コンパイラのためにも記憶スペースが必要である．コンパイラを会話的モードの中で頻繁に用いるとすればそれは常駐になる．

良質なコードを生成するコンパイラは一般に大型になる．コンパイラというとき，それは何らかのマシンを想定している．普通それは既存のマシンである．しかし従来型のマシンは，必ずしも記号処理に適した機械語体系になっていない．そこをやりくりして良いコードを作るには高級な技法が必要になるだけでなく，コンパイラも大型化してくる．

Lisp や Prolog を想定して機械語を選びマシンを構成すればコンパイラは楽になる．小型高速になるとともに良質のコードを生成しうることになるだろう．その検討は，インタプリタをハードウェア化（専用マシン化）するにしても必要である．しかしその極限は従来イメージのコンパイラというより，むしろローダのようになると考えられる．

ところで，これまでの専用マシン（主として Lisp マシン）の試作例では，インタプリタは直接回路化されるというより，マイクロプログラム方式のマシンによりファームウェアによって実現されている．それでもその効果は大きい．性能-コスト比で1桁は上っている．4年ほど前の試作例に神戸大の Lisp マシンがある．これはミニコン程度のマシンであったが，評価データによると，大型機である Hitac 8800 や Acos 800 などの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．これには適切なマイクロ命令の選定が効いている．

マイクロプログラム方式の場合，次には，マイクロコードへのコンパイルの問題が出てくる．マイクロコードへの最適化コンパイルは，通常の場合より難しい問題を発生する．一方では，メモリ速度の限界から，過度の最適化は必ずしも必要でないという事情が出てくるにしても，キャッシュなどを導入すると事情はまた元に戻るだろう．

コンパイルを考慮すると，垂直型マイクロプログラム方式の方がよいだろうが，キャッシュ・メモリなどを前提にするとその存在意義が問題になってくる．RISC (reduced instruction set computer) のような構想はそこから出ている．マイクロ命令のレベルを上げれば，機械語命令との差がはっきりしなくなる．むしろ通常の命令とマイクロ命令の中間に命令系を設定し，2層ではなく，1層で済ますことが考えられる．RISC 方式ではコンパイラに比重がかかってくる．RISC 方式は記号処理を対象としても研究されてよいものであるが，現状の研究では PL/I 系や Pascal 系の言語が選ばれている．これらの言語でインタプリタを介したり，そこへのコンパイラを想定したりすれば，記号処理の観点からはあまり有意義でない．Lisp や Prolog について直接 RISC 方式が成り立つかが検討されなければならない．しかし，この場合も，重いコンパイラが介在することは，会話的使用の観点からは嬉しいことではない．

一方，水平型マイクロプログラミングの場合，それをさらに一段下に下げた構成が考えられてよいと思われる．

Prolog や PS の方が，Lisp より知識工学，自然言語処理に適しているとして，それを Lisp の上で作るとすると，同様の問題が発生する．それでは直接専用マシン化したらどうなるかということになるが，これらについての専用マシンの試作例はまだ出ていない．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がある．その例題 TPU-6 の例だと，HLISP, OLISP のインタプリタで約 20,000 ms,

コンパイルコードで約 4,000 ms かかる．これは神戸大 Lisp マシンでは 6,728 ms である．ユニットレゾリューションの一部をマイクロコード化するとそれは 376 ms になっている．ベースとしたマシンの違いを考えてもここでも1桁以上の差がでてくる．

PS から Prolog への変換がローディングの中に組み込める程度に単純なものであることを考えると，Prolog マシンの存在意義は大きいと思われる．

Lisp マシンや Prolog マシンの実現方式を考えてみると，これまで提案されそれぞれ魅力的だが，これまでの商用機では必ずしも取り入れられていない方式が有機的に活用される可能性が出てくる．タグとかハッシュとかの手法は（スタック等の取り込みはもちろんとして），このような記号処理（専用）マシンの枠組で活きてくると考えられる．

より将来的なイメージでは，並列処理方式がある．データフローマシン，リダクションマシンのような構想は，数値計算ばかりでなく，むしろそれよりも，記号処理マシンの枠組において有機化されよう．これから生れ出るであろう新しい並列方式についてもそういうことになる可能性が強い．

7. むすび

現在のマシンは低水準の記号処理マシンである．それは安定した記号処理の水準をもとにしている．高水準でかつ安定した記号処理のレベルは存在するであろうか．それはそれぞれの記号処理用言語が示そうとしていることである．いくつかの将来的応用から考えると，それはもう少し高水準なところにありうるように思われる．それは Prolog が示しつつあるような「推論」のレベルであろう．あるレベルの推論を記号処理の次の基本水準と考えてよいように思われる．

そのような水準を具現する記号処理専用のマシンは，推論マシンと呼んでもよいであろう．推論マシンは記号処理に「専用」と考えてもよいが，一方記号処理の普遍性を考えると，それは一種の汎用マシンでもある．そのような高水準マシンが高効率を発揮するようになれば，それはコンピュータの新しい主流を形づくることになろう．そのようなマシンの出現は，高次の応用，たとえば自然言語処理や知識工学的システムの実現を容易にするであろうから，そのような応用自体，いまよりもっと高度化されるであろう．（高水準）記号処理専用マシンへの期待は，そのような高度のコンピュータ応用への願望に基づいている．それとともに，その実現可能性はいままでのいろいろな研究によってもみえ始めているし，今後の研究によって現実のものになってくるであろう．

（昭和57年7月19日受付）