解　説

問題解決と推論機構*

淵　　一　博**

1. はじめに

およそ知的な精神活動は，何らかの意味で，問題解決を目標とするものといえよう．「問題解決」は古くから哲学の主テーマの一つであり，また心理学のテーマであった．人工知能研究 (AI) は，知能の機械化あるいは逆に，機械モデルによる知能の解明を意識するものである．したがって，問題解決の問題は，そこでの中心的な課題になってくる．AI の実際的応用の面から見れば，(役に立つ) 問題解決システムの作成がその目標ということになろう．

いま話を，「プログラミング」のような合目的な活動に限れば，それはまぎれもなく，問題解決の過程である．プログラマは，いわば，問題解決マシンであり，問題解決の成果として，プログラムを生産しているのである．

一方，現代のすう勢としては，ますます多くの知的分野において，そのプログラム化が進行している．プログラミングは問題解決の一種であるが，この傾向は逆に，問題解決がプログラミング自身に埋め込まれて実現されるようになっていく，ということを意味している．

とすれば，問題解決研究の対象が，プログラミング自体にシフトしてくることが予想される．実際，人工知能研究の一つの大きな流れはその方向に進んでいることが観察されるのである．

プログラミング自体を対象とする研究は，計算機技術プロパーの中からも，「ソフトウェア工学」として生れてきている．ここはいくつかの流れの合流点であり，しかも広大な分野である(学派によって，問題意識，個別問題の選定，問題の処理の仕方あるいはまた「言葉づかい」に差はあるであろう．しかし，それらは一つの大きな流れに合流していくべきものと考えられる.)．

問題解決の過程を操作としての面から見れば，それは，広い意味で，「推論」のプロセスである．哲学者パースは「すべての精神作用は推論である」という議論を展開している[1]***．

一方，1930年代の数理論理学の成果によって，推論と計算が本質的に同一のものであることが明らかにされている．問題解決的な見方（推論）をとるか，プログラミング的な見方（計算）をとるかいずれにしても本質的な差はないのである．現実的なシステムの上での，両者（推論と計算）の一体化は，情報システムの今後の展開への大きなポイントになるであろう．

2. 問題解決の諸要素

2.1 推　論

問題解決のための原理的な手法の一つは，問題をより解きやすいいくつかの副問題に分割・還元していくことである (problem reduction)．また，可能な解の組合わせを次々に作り出してみて，問題の条件に合うか検査してみる方法もある (generation and test)．これらは解の「探索」法にかかわることであり，抽象的には，(問題を表わす)グラフの上での探索の問題と見ることができる．

また，これらは，論理学の用語を使って，「推論」と見ることもできる．

推論の形態としては，演繹 (deduction) と帰納 (induction) が古くから論じられてきた．パースは，これに加えて，仮説形成 (abduction) という，問題解決にとって重要な推論があることを指摘した．パースによ

* Problem Solving and Inference by Kazuhiro FUCHI (Electrotechnical Laboratory).

** 電子技術総合研究所

*** 計算機科学の研究者に影響の強い哲学者としてヨーロッパ勢について，「論理哲学」[2] のウィトゲンシュタイン (1889～1951) を挙げるとすれば，米国勢についてはパース(1839～1914)であろう．パースはプラグマティズムの始祖の一人であるが，記号論理学の開拓者の一人でもあり（「パース・ストローク関数」は彼の名を留めたものである），今の眼で見ても新鮮で，非常に計算機科学的 (AI 的) なのである．

れば，これらは別々の論理ではなく相互に関連し合っている[1),3)].

いま，推論のもっとも基本的な形式である，三段論法を考えてみる．

Ⅰ．大前提 (rule) $\forall x.\ P(x) \supset Q(x)$

Ⅱ．小前提 (case) $P(a)$

Ⅲ．結論 (fact) $Q(a)$

演繹は，大前提Ⅰと小前提Ⅱから，結論Ⅲを導き出す推論である．一般的な規則から個別的事実を導くものである．帰納は，逆に，小前提Ⅱと事実Ⅲがあったとき，Ⅰが多分成立つだろうと推測することであり，個別的事実から一般的法則を抽き出すものである．

パースのいうアブダクションは，事実Ⅲがあるとき規則Ⅰをもとにして，多分Ⅱが成立っているからであろうと推定するものである．たとえば，ある症状（事実）があったとき，その原因（小前提）を推定するプロセス（診断）はこれに相当する．

このパースの解釈は，いまでいう論理式の「手続き的解釈」になっているのである．

証明は演繹的推論と考えられているが証明を実際に進めるときの手続きは幾通りもありうる．Ⅰ＋Ⅱ→Ⅲの方向で証明（推論）を進めることを前向き推論 (forward chainig) と呼ぶ．逆に，Ⅰが成立っているとき（公理などとして），Ⅲを証明するには，Ⅱを証明すればよいという(アブダクションと同じ方向での)推論を後向き推論 (backward chaining, goal-oriented inference) という．

これはまた問題解決の探索の方向に対応させることができる．すなわち，generation and test は前者に，problem reduction は後者に相当する．このように，問題解決の過程は，推論（あるいは，証明）と見立てることができるのである．

論理をプログラミング言語と類比させると，証明(推論)は，その「制御構造」と見ることができる．

2.2 知　識

推論は問題解決の一つの側面（制御構造）である．もう一つの側面は，その問題の属する分野についての「知識」である．問題はその知識を利用して解かれる．問題解決は知識の利用の問題であると考えられる．

知識を蓄積し，利用するためには，知識をどう表現しておくか，その形式が問題になる．知識表現の問題は，プログラミング言語でいえば，「データ構造」の問題である．その構造はまた，制御構造である推論の様式に深く関連する．

「知識表現」はいまの人工知能研究の最前線の問題になっている．狭い意味の問題解決だけでなく，これもまた，いまの人工知能研究の最前線の問題である「言語理解」にとっても知識表現と推論は重要な問題である．

知識表現の問題は論理系の選定の問題と考えることができる．もっともよく整備された論理系は（一階）述語論理であろうが，それが最適であろうか．述語論理の中にもいろいろな階層，いろいろな形式がある．どれがよいか．

述語論理に対応して，関数論理（ラムダ計算）もある．ラムダ計算は，プログラムの論理の表現によく使われている（LISP はそれをもとにしたプログラミング言語である）．また，言語学でも，それを意味表現に用いる動き（モンタギュ文法など）がある．

また，生成（書きかえ）規則をもとにした，プロダクション・システムのような「論理」もある．

あるいは，意味ネットワークのような表現がよいのか．また，フレーム理論をベースにしたシステム (KRL など) がよいのか．また，「プログラミング言語」そのものを知識表現（「知識の手続き的表現」）のベースと考えるか．

表現の問題だけではなく，知識そのものの蓄積と検索の問題がある．これは「データベース」の問題である．問題解決システムは，いまや知識ベースシステムとして捉えられるようになってきている．このように考えたとき，そのシステムの基本構成は，

┌推論機関 (Inference Engine)
└知識ベース (Knowledge Base)

と考えることができよう[4)]．

3. 証明システム

3.1 定理の証明

定理の証明は，人工知能研究の初期からのテーマの一つである．証明についての関心は二面的であるように思われる．一つは具体的な「定理」の証明についての関心である．個別的な問題分野，たとえば初等幾何の中で，どの程度のことが証明できるかということである．もう一つは，もっと抽像的に，ある論理系の中での証明方式についての関心である．

証明システムの研究にとって，Robinson (1965) による resolution 原理の提案は一時期を画するものであった．これは非常に単純（論理的公理を必要とせずただ一種の推論規則からなるもの）で，しかも強力（一

階述語論理の中で完全）なものであった．

その後，その証明図の中から，プランやプログラムを抽出できることが発見された．これはプラン作成やプログラム合成に新しい道を拓くものと考えられ，(一階述語論理)証明システムは，各種の問題解決の基礎と考えられた．

しかし，原理的なresolution方式だけでは，無駄や重複のある中間結果を多数作りだしてしまうことがある．証明の効率（速度）を上げるため，resolutionの適用を制限し探索空間を狭める改良——証明のストラテジーの研究が進められた．

このような動きに対し，一方では，resolution方式あるいは述語論理そのものに対し，その本質的な非効率性や適用の限界について強い批判が出された．そして，対案として「AI用プログラミング言語」が提出された．

この批判には正当な，生産的な部分があったが，一方，いま考えると，言いすぎの点もあったと感じられる．**4.5**で述べるようにその後の歴史的経過をふまえて，証明システムを見直すことも必要かと思われる．

3.2 Resolution 方式

Resolution（分解あるいは導出と訳されている．溶解とか融合の方が良いのではないかと思われる）は，一階述語論理式のSkolem乗法標準形といわれるものから出発する．単一の述語（定数，変数，関数からなる項をもつ）からなる式をアトム，アトムまたはそれに否定のついたものをリテラルという．リテラルをorでつないだものをクローズという．クローズの中に現われる変数は，すべて全称記号$\forall$で束縛されていると考える．このようなクローズの集合がSkolem乗法標準形である．任意の論理式はこのようなリテラルの集合に変換することができる．

証明は，証明すべき命題の否定を加えて，それから矛盾を導く帰謬法（refutation）による．

推論規則は三段論法の一種で，

$$\frac{A\vee\cdots\vee B\vee X \qquad \sim X\vee C\vee\cdots\vee D}{A\vee\cdots\vee B\vee C\vee\cdots\vee D}$$

この推論がresolutionと呼ばれる．

一般にアトムは，変数を含む項を持っている．resolutionを働かせるには，変数に適当な代入を行って相補的なリテラル同志を「同一化」(unification) しなければならない．この同一化はmgu (most general unifier) によって行われる．

以上は大まかな記述であるが，詳細は教科書[5),6)]を参照していただきたい．

原理的なresolution方式では，互いに同一化できるリテラルがあれば，それらを含むクローズを自由に融合できる．矛盾を見い出すにはresolutionの木の上での「探索」が必要である．横型（breadth-first），縦型（depth-first），その他の探索技法が使われる．

しかし，原理的方式だけでは，探索の空間が広すぎるのである．証明の性質を利用して探索空間を狭くするために各種の改良法が考案されている．ここでは，そのいくつかを取上げて特徴を見よう．

クローズの除去　どれとも融合する可能性のないリテラルを含むクローズは不要である．つねに真であるクローズは不要である．他のクローズから含意されるクローズは不要である．これらの不要なクローズを除去することにより，探索空間を減らすことができる．

支持集合　resolutionの連鎖の出発点となるクローズを，ある集合に限定する．たとえば，証明すべき命題の結論部から生じたクローズを支持集合とするとgoal orientedな推論になり，前提部を支持集合にすれば前向き推論になる．

リテラルの順序づけ　クローズの中のリテラルに順序をつけ，融合をその順序に制限する．これは探索空間を大幅に減少させる．もっとも順序のつけ方で効果も大きく変わることがある．

linear resolution　ある出発のクローズから矛盾に至るまで一本の連鎖からなると考える．融合はその連鎖上でのみ行われるとする．これも効果的であり，このような証明図が存在することが証明されている．

semantic resolution　矛盾を生じるということは，変数，関数，述語にどのような解釈（モデル）をほどこしても矛盾になるということである．そこで，あるモデルについて，矛盾にいたるのに不要であることが，別途何らかの手段で確められれば，その経路は無視してよい．モデルの立て方で種々の方式が派生する．通常は，構文的に検査できる程度に簡単なモデルが選ばれる．たとえば，融合するクローズの片方が正のリテラルからのみ成るものに制限するという方式がある．

resolution方式の改良では，完全性（真なるものは必ず証明できるという性質）が意識される．すなわち，制約的なストラテジーによって探索空間を小さくして効率を上げる一方，証明力は保存しておきたいということである．

完全性を保存するようなストラテジーも，いくつか組合わせれば完全でなくなることがある．かたや，

いくつかのストラテジーの組合わせは一般的に効率を上昇させるであろう．完全性を犠牲にすることによって，効率を上げる方式もある．

完全性をもつ組合わせとしては，OL (ordered linear), SL (selective linear), ME (model elimination) などと呼ばれる方式がある（名称は創案者により異なり，技術上の小さな差はあるが互いに同等である）．その基本は，リテラルの順序づけと，linear resolution の組合わせである．この方式は，形式を若干変更すれば，problem reduction 形式の拡張と見立てられる[7]ものである．

改良法の詳細もやはり教科書[5),6)]にゆずろう．

3.3 その他の証明方式

Resolution 方式に対する批判の一つは，証明の動作に対する制御がきかないという点にある．いろいろな証明手順が提案されているが，すべてを一種にまとめられるわけではない．どれか一つをとれば問題によって得意・不得意を生じる．証明の制御をどうするかは未解決の大きな問題である．

ところで，人の書く論理式の書き方には，制御情報的なものが含まれていることがある．resolution 方式は，正規形を出発点にするためそれさえもこわしてしまう．

Gentzen を源流とする自然推論 (natural deduction) の問題意識の一つはそこにある．自然推論の基本は，「場合分け」である．問題によっては，こちらが有利なことがある．

Resolution だけが推論ではない[8]．自然推論も述語論理をベースにしているが，述語論理だけが論理なのではない．ラムダ計算（関数形式）も論理の一形態であるし，書きかえ規則（プロダクション・システム）も論理の一種であろう．

しかし，証明のストラテジー論の観点からは，resolution 方式がもっとも広く，深く検討されているといえよう．resolution 方式は，その原理的な枠組では，制御構造についてほとんど無構造である．むしろその無構造さが，ストラテジーの研究を容易にしたのではないかと思われる．

4. 人工知能用プログラミング言語

4.1 問題意識

一階述語論理での証明システムに対する批判をもとに，Hewitt は対案として PLANNER を提案した．これは 1970 年前後の人工知能研究に大きな衝撃を与えた．これから生れた一群のシステムは，人工知能用プログラミング言語と称された[9]．

知識には，何をということだけでなく，どのようにという知識があり，問題解決にとって，これは非常に重要である．述語論理はそのような手続きに関する知識を表現するには適していない．

一階述語論理に，問題の記述を公理としてつけ加え，一般的な証明システムを働かせるだけでは，良い問題解決システムにならない．効率的なシステムにするには，いろいろな経験則（ヒューリスティックス）が組み込まれなければならない．しかし，(一階)述語論理（や resolution システム）はそれに反する性格のものである．実際，論理における証明システムは広大な探索空間の中を盲目的にさまよい，非常に非能率である．証明システムを問題解決に用いようというのは，根本的な考え違いではないか．そもそも知識は，論理におけるような宣言的な形ではなく，手続きとして（プログラミング言語の中で）表現されるべき（知識の手続き的表現）ではないか．しかし，既存のプログラミング言語が，知識の表現に適しているとはいいがたい．論理を改作して，新しいプログラミング言語を設定すべきである．というような主張がなされた．

これらの主張の当否は後に回すとして，そのような問題意識から新しい動きが始まったのである．

4.2 手続き的解釈

前に述べたように，論理式 A→B は，証明に用いられるとき，前向きにも後向きにも使うことができる．これを証明手続きとして，システムの内部にかくしておくのでなく，おもてに出してユーザが利用できるようにしておく必要がある．PLANNER[10] では定理を三つの型に分類する．A→B は必要に応じて，三つの型のいずれかで表現される．

(1) 前提 (antecedent) 型：A が主張 (assert) されていれば，B を主張する（主張はデータベースに登録されるとともに前提型定理を起動する）．

(2) 結論 (consequent) 型：B を証明することが目標 (goal) であれば，A を証明する（目標はすでにその主張がデータベースにあれば満足される．そうでなければ，結論型定理を起動し，定理は新しい目標を立てる）．

(3) 打消し (erasing) 型：B でなければ A でない（打消し (erase) があれば，前にデータベースにあった主張がとり消され，打消し型定理を起動する．定理はまた別の主張を打消したり，新しい主張をデータベー

スに加えたりする).

これをベースに新しいスタイルのプログラミング言語が作られた．これをプログラミング言語的に見ると

(1) パターンマッチングによる手続きの呼出し：定理は，それが指定するパターンに合った主張によって起動される．定理は手続きと見立てられるから，これは新しいスタイルの手続き呼出し法になっている．

(2) パターンマッチングによるデータベースの管理：データベースの中へのデータの追加，削除がパターンマッチングによって行われる．

(3) 非決定性とバックトラッキング：パターンに合うデータや手続きは必ずしも1個ではない．そのうち一つを選択して先に進み，失敗すればあと戻りして(それまでの効果を取消して)，別の可能性を選択し，やり直しを行う．

(4) 助言リスト：非決定性を制御するために選択の範囲を限定する助言リストをつけることができる．

実際に作成されたのは縮小版 PLANNER であるが，これは言語理解システム SHRDLU (Winograd) において，意味表現，質問に対する答の作成，ロボットのプラン作成に使われた[11].

PLANNER にならって，QA 4 (QLISP), POPLER などが作られた．

4.3 バックトラッキングへの批判

ところが PLANNER に対しても，Sussman から批判が出され対案として CONNIVER[12]が作られた．これは PLANNER と同じ問題意識に立つが，ある意味でそれをさらに徹底させたものということができる．批判は主としてバックトラッキングに向けられた．バックトラッキングは総型探索である．助言リストはあるが，その動作を制御するのは難しい．

CONNIVER では，可能性リストを使って横型的な探索を復活させている．可能性リストには generator を含みうる．generator は Au-revoir の機能によって，リスト生成を一時休止することができる．その他の特徴として，(1) PLANNER は証明システムのスタイルを残しているが，CONNIVER はもっと徹底して，普通のプログラミング言語のスタイルに近づけている．(2) CONNIVER の方が低水準的であり，細かい制御が可能になっている．(3) データベースについて，コンテクスト構造を導入している．

4.4 AI 用言語のその後

PLANNER や CONNIVER は，世に与えた衝撃のわりには，その後定着しなかったように見える．そのままでは決定版ではなかったのであろう．これにはいくつかの事情があると思われる．

PLANNER は，CONNIVER の打撃のせいか，完全版の処理系が作られなかった．創始者の Hewitt は，それをベースに，新たに Actor 形式[13]を提案して，それを具体化する PLASMA の作製に転進した．

CONNIVER の提出した機能の多くは，INTERLISP に取入れられた．結果的には，親元である LISP の機能拡張として処理されたようである．(QA 4 も QLISP としてやはり LISP に埋めこまれた.) CONNIVER の創始者の Sussman のグループはむしろ，推論システムに戻って新しい方式を試みていると見られる．前向き推論をベースにした AMORD[14] などの開発に進んでいる．

AI 用プログラミング言語の提案の裏には，証明システム（とくに resolution システム）の「遅さ」があったと思われる．しかし，実際に作られた CONNIVER などの処理系は決して速くはなかった．LISP の上でインタープリタによって働かせるのではおのずからその速度に制限がある．そこで，速度を要する大規模なプログラムは LISP 自身に戻ることになる．

4.5 証明システムの見直し

処理系の速度にはいろいろな要因がある．たとえば，「LISP は遅い」という昔の評判を想いだしてみよう．実現技術が進んで，現在の LISP システムは高速になっている．遅さは LISP に内在するものではなかったのである．

証明システムでいえば，それに対する批判は，証明手続きの研究が十分なされる以前に出されたものである．その上，ここにもまた実現技術の問題があった．最近の展開によれば，SL (OL, ME) 方式をもとにする処理系は LISP に近い速度を持ちうることが分ってきている[15].

かつての激しい批判のために，証明システム，とくに，resolution システムは実態以上に悪く見られている節がある．実はそれほど悪くもないのである．

前に，SL をもとにすれば problem reduction 形式の拡張が考えられることを[7]述べた．PLANNER のベースになった手続き的解釈は，SL のような証明手続きをもとにすれば，直接的に適用できるのである．これは，述語論理を「プログラミング言語」そのものと見たてることに通じる[16].

この「述語論理プログラミング」を実現するものとして PROLOG[15] が作られている．PROLOG をもと

に，プラン作成システム，数式処理システム，言語理解システムが開発されている[17]．

PROLOG は，PLANNER の goal-oriented な部分を，論理の方に引き戻したものと見ることができる．前向き推論の部分は PROLOG には含まれていないが，それへの拡張も考えることができる．とすれば，PLASMA や AMORD をその中で捉えなおしてみることも可能と思われる．AMORD[14]は，制御を明示的に表現しようとするものであり，今後の方向を示唆している．この考えを（拡張）PROLOG に取入れることもできるのである．

論理に戻ることの良さは，その意味論が確立していることにある（Hewitt らはそれに反対しているが）．ところで，プログラミング言語の意味論の問題は，ソフトウェア工学での新しい展開の一つである．その考えをいわゆる人工知能用言語について適用してみるとどうなるか．この見地からも（述語）論理プログラミングは今後の可能性として大いに検討する価値があると思われる．

述語論理に対する批判には，その汎用性に起因するものがある．汎用性は具体的な問題に対して無力であるということに通じる．しかし，これは「汎用」プログラミング言語についても言えることである．上手にも下手にも使える可能性をもっているのである．証明システムは問題解決システムそのものではない．もともとそれだけのものだという認識に立った上での批判が必要だと思われる．

5. プラン作成システム

5.1 プラン作成と問題解決

知能ロボットの時代には，ロボットの行動計画（プラン）の作成が問題解決の典型と考えられた．

プランの作成が通常の証明問題と異なるのは，行動によって状態の変化がひきおこされることである．一階述語論理でも，状態変数を導入すればこのことを記述することができ，それを証明システムにかければ，プランを取り出すことができる[18),19)]．この可能性は当初大きな期待をもたれたが，すぐ行きづまりをきたした．

状態の存在によって，探索空間が極端に広がるのである．ある行動のひき起す状態変化は通常局部的なものである．しかし，その局部的変化だけでなく，他の部分には影響を及ぼさないということも合わせて記述しなければならない．これを枠の公理（frame axioms）の問題という．

この公理の存在は，記述上も問題であるが，解法上も問題をひきおこす．枠の公理によって直接関係のない行動も許されるからである．さらに，いくつかの副目標の間の干渉をひきおこす．これに対しては，前向き推論も後向き推論もあまりうまく働かないのである．

枠の問題を処理するには，たんなる状態変数でないうまい状態の記述が必要である．そこで STRIPS[20]では，追加リストと削除リストの対で変化を表わし，それらのリストの積み重ねで状態を表わす方式が導入された．これに対する処理系としては resolution システムが用いられた．

一方，その頃出現した対案として PLANNER などでは，データベースの状態そのもので状態を表わそうとした．打消し操作があるのはそのためである．

しかし，それらでも副目標の干渉（並立目標）の問題はよく解けなかった．一見複雑そうに見える問題が解ける一方，簡単な積木の問題ができなかった．

5.2 並立目標の問題

最近この問題についての解がいくつか出てきた．一つは並列プランの考えを用いるもの[21),22)]である．もう一つは regression の考えを用いるもの[23),24)]である，ここでは後者の考え[24)]を紹介する．

状態は初期状態に加えられた動作の系列で表現する．ある事実が成立っているかを見るには，その系列をたどって，それらが及ぼしている影響を見ていけばよい（STRIPS の考えの復活）．ここで，すでにいくつかの副目標が実現されているとする．その次の副目標を実現するためにある動作が必要であるとする．この動作は，それまでに実現されている状態をこわしたくない．この動作を最後につけ加えることは必ずしも可能でない．そこでこの動作を，すでにある動作系列のどこに挿入すれば，すでに実現されている状態をこわさないですむか，後の方からたどって（regression）その場所を見つける．次に必要とされる行動も同様にどこに挿入すればよいか探していく．このようにして並立目標のプランが作成される．

ここでの教訓は，枠の公理をどうシステムに組み込んでしまうか，また動作と状態変化の関係がもつ性質を解の発見法の中に組込みにしてしまうか，ということである．いいかえれば，問題領域の知識を処理して，システムの中に組込みにしてしまうということである．

ところで，積木の問題については，最適解でなくてよければ，一度全部バラして組立てなおすという解法がある．バラす方も組立てる方もそれぞれは直線的である．このような解法は一般性がない(あるいは自明)と考えられるのか，ほとんど取上げられることがない．しかし，これもまたやはり問題の性質を利用したものである．むしろ，このような簡単な解法をどのように発見するか（正しさを保証しつつ）が，プラン作成(問題解決)の今後の一つの大きな課題というべきであろう．

6. 知識ベースシステム

6.1 エキスパート・システム

問題解決システムの最近の大きな動きとして，個別的な（また実際的な）問題領域において，専門家の知識を集約し，専門家と同程度の働き（パーフォーマンス)を持つ，実用的なシステムを作ろうとするものがある．その古典的な例は数式処理システムMACSYMAなどであろうが，最近では医療診断の分野でのシステムなどが注目されている．

そのような動向の中心の一つであるスタンフォード大学での Heuristic Programming Project の例を見てみよう[25]．ここでは，いろいろなシステムが試作されている．

DENDRAL は質量分析と核磁気共鳴分析のデータから有機分子の構造を推定しようとするものである．対象分野では専門家より速く正確で，実用レベルで用いられている．META DENDRAL はスペクトルデータと分子構造の実際から，DENDRAL で用いるための生成規則をつくりだすもので，専門家の評価にたえる規則を発見している．

MYCINは血液関係の病気について，症状，検査データから診断し，投薬の案を作成する．これは医師への助言システムと考えられている．MYCIN では生成規則を goal-oriented に使っている．また「説明システム」が新しい．「説明」は，推論の過程を要約して提示することである．TEIRESIUAS は，MYCIN の「知識獲得システム」で，医師が診断や投薬プランに不満なとき，推論の過程をたどって，規則を修正したり追加するためのものである．

AMは，初等数学の分野で，興味ある概念を発見するシステムである．新しい規則の生成のための規則と，それが面白いものかどうかを判断する規則からなる．加算，乗算，素数，素因数分解の一意性などの概念を発見している．

MOLGEN は分子遺伝学の分野で，DNA を扱う実験の計画について助言するものである．CRYSALIS は，電子密度図からたん白質の構造を推定する．

これらの一連の研究で特徴的なのは，(1)generation and test の考えで処理する（ただし MYCIN は後向き推論)．(2) situation⇒action という形の生成規則を使っている．生成規則は十分に強力（ストラテジー的な知識もとらえられる）で実用的に役立っている．(3)これらのシステムはそれぞれ数百程度の規則から成立っており，それがエキスパートの能力に近づく目安とも考えられる．

なお，生成規則（書きかえ規則）は，述語論理の推論と似ているが，短期記憶（小さなデータベース）の書きかえが可能であり，また高階の操作（規則自身を作りだす）があり，強力である．しかし，動作の追跡が難しいという欠点もある．

6.2 知識の組込み

これまでの研究の流れから明らかになってきているのは知識の重要性である．強力な問題解決能力を持つには，その知識を分析し，それをいわば「コンパイル」してシステムに組み込んでいくことが必要と考えられる．そのような問題解決の構造化の試みとしてHACKER[26]がある．コンパイルの概念（これを「熟練」と考える）の他，デバッグ，批評などの概念を有機的に統合しようとしている．

このことは，たとえば，「定理の証明」という課題自身についても必要なことである．エキスパートとしての証明システムでは，算術（演算，等号，不等号)，関数の結合性や可換性など処理を組込みにすることが必要であろう．それらをいちいち「公理」から説きおこしていたのでは，実際的な証明は実現できないだろう．また，初等幾何での「図」の働きのような，モデルの取り扱いが容易になる枠組みが必要になるだろう．これらも，知識の組込みと考えられる．

また，プログラムの性質の証明に使われる証明システムは，単に汎用の証明システムでなくて，プログラムという問題領域の性質を組み込んだ証明システムでなくてはならないだろう．

6.3 知識の OS

エキスパート・システムで現われてきた「説明システム」や「知識獲得システム」のような部分は，今後の問題解決システムのあり方を示唆するものといえよう．これらは，いわば，「メタ知識」システムの萌芽

ともいえよう．問題解決のフロント・エンドだけでなく，バックアップもまた重要である．とすれば，これらは，全体として，総合的な「オペレーティング・システム」を構成していくものと考えられるであろう．

参考文献

1) パース：論文集第4章人間記号論の試み，上山春平編世界の名著第48巻（中央公論社），pp. 128～167.
2) ウィトゲンシュタイン：論理哲学論，山元一郎編世界の名著第58巻（中央公論社），pp. 307～429.
3) Pople, H. Z. Jr.: On the mechanization of abductive logic, Proc. 3rd IJCAI, pp. 149～152 (1973).
4) Davis, R. and Buchanan, B. G.: Meta-level knowledge: Overview and applications, Proc. 5th IJCAI, pp. 920～927.
5) Chang, C. L. and Lee, R. C. T.: Symbolic logic and mechamical theorem proving, Academic Press (1973).
6) Loveland, D. W.: Antomated theorem proving: A logical basis, North Holland (1978).
7) Loveland, D. W. and Stickel, M. E.: A hole in goal trees: Some guidance from resolution theory, Proc. 3rd IJCAI, pp. 153～341 (1973).
8) Bledsoe, W. W.: Non-resolution theorem proving, AI, Vol. 9, No. 1, pp. 1～36 (Aug. 1977).
9) Bobrow, D. and Raphael, B.: New programming languages for AI research, Computing Survey, Vol. 6, No. 3, pp. 153～174 (Sept. 1974).
10) Hewitt, C.: PLANNER: A language for proving theorems in robots, Proc. 1st IJCAI, pp. 295～302 (1969).
11) Winograd, T.: Understanding natural languge, Academic Press (1972).
12) Sussman, G. J. and McDermott, D. V.: From PLANNER to CONNIVER—A genetic approach, Proc. FJCC, pp. 1171～1179 (1972).
13) Hewitt. C., Bishop, P. and Steiger, R.: A universal modular ACTOR formalism for artificial intelligence, Proc. 3rd IJCAI, pp. 235～245 (1973).
14) de Kleer, J., Doyle, J., Steele, Jr., G. L. and Sussman, G. J.: AMORD: Explicit control of reasoning, SIGPLAN Notice, Vol. 12, No. 8/ SIGART Newsletter No. 64, pp. 116～125 (Aug. 1977).
15) Warren, D. H. D. and Pereira, L. M.: PROLOG -The language and its implementation compared with LISP, SIGPLAN Notice, Vol. 12, No. 8/ SIGART Newsletter No. 64, pp. 109～115 (Aug. 1977).
16) Kowalski, R. A.: Predicate logic as programming language, Proc. IFIP 74, pp. 569～574 (1974).
17) van Emden, M. H.: Programming with resolution logic, Machine Intelligence, Vol. 8, pp. 266～299 (1977).
18) Green, C. C.: Application of theorem proving to problem solving, Proc. 1st IJCAI, pp. 219～239 (1969).
19) Waldinger, R. J. and Lee, R. C. T.: PROW: A step toward automatic program writing, Proc. 1st IJCAI, pp. 241～252 (1969).
20) Fikes, R. E. and Nilsson, N. J.: STRIPS: A new approach to the application of theorem proving in problen solving, AI, Vol. 2, No. 3/4, pp. 189～208 (1971).
21) Sacerdoci, E. D.: The non-linear nature of plans, Proc. 4th IJCAI, pp. 206～214 (1975).
22) Tate, A.: INTERPLAN: A plan generation system which can deal with interactions between goals, Univ. Edinburgh MIP-R-109 (Dec 1974).
23) Warren, D. H. D.: WARPLAN: A system for generating plans, Univ. Edinburgh DCL Memo No. 76 (June 1974).
24) Waldinger, R.: Achieving several goals simultaneously, Machine Intelligence Vol. 8, pp. 94～136 (1977).
25) Feigenbaum, E. A.: The art of artificial intelligence: I. Themes and case studies of knowledge engineering, Proc. 5th IJCAI, pp. 1014～1029 (1977).
26) Sussman, G. J.: A computer model of skill aquisition, American Elsevier. (1975).

（昭和53年9月4日受付）