解　説

人工知能向言語（2）*

横井　俊夫**

5. CONNIVER[51),75),76)]

その名前（しめし合せるもの）の通り，PLANNERの盲目的な自動後戻りを問題解決システムの基本アルゴリズムとして，確定することへの批判をこめ，インタプリタの管理部をも，それぞれの目的に合うように利用者が記述できること，すなわち，多階層の動的なシステムが記述できること，Micro-PLANNERでは，データ・ベースの変化は，管理部の中で，制御の流れに一体化されていたが，動的モデルとしては，客観的に観察できるようにすべきであるなどを目的として製作された．先の単位プログラムを用いて，図示すると図-7のようになる．

CONNIVERでは，インタプリタが用いる全てのデータの構造が，明示されており，それを操作する関数群も用意されている．利用者が注意深くプログラムするならば，システムの内部にまで手を入れることが可能である．しかし，知識利用の動的モデルの表現要素としては，あまり興味がない．表現要素として意味のある部分に着目した場合，想定される問題解決システムが図-7である．任意の階層構造を構成することが出来るプログラム（プロセス）群となる．階層をなす単位プログラム間の連絡用の関数として，図示したTRY-NEXT, ADIEU, AU-REVOIRがある．

関数が用いる下位の単位プログラムを管理する特別のデータ構造が，可能性リスト（possibility-list）である．これは，図-5*** の後戻り用スタックと c_4 に見られるパターン照合の結果生ずる，変数と照合する値との対のリストとを一体化し，一般化したものである．すべての単位プログラムに共通の下位プログラムとして，データ・ベースプログラムがある．その解決機構は大きく2つに分けられる．1つはインデックス機構で，Micro-PLANNERより一段強化されたパターン照合，検索機能を果たす．もう1つはコンテクスト機構で，図-5のデータ・ベース用ヒープを一般化し，利用者に解放したもので，多数の状態からなるデータ・ベースを作る事ができる．

図示した連絡用の関数に注目すると連絡の形式に制限が加えられる．つまり，ある程度，記述対象となる解決システムの構造が制限される．それは，下位プログラムとの連絡方法の内で，〈問題〉を与える，〈結果〉を受け取るという部分を重視してTRY-NEXTの機能が設計されているためである．この考え方は，上位

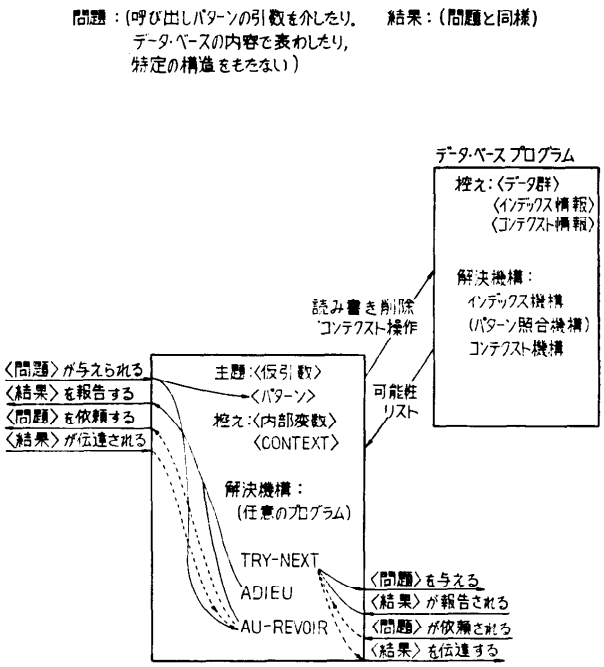


図-7　CONNIVERの構造

* Programming Languages for Artificial Intelligence (2) by Toshio YOKOI (Machine Inference Section, Information Sciences Division, Electrotechnical Laboratory)

** 電子技術総合研究所パターン情報部推論機構研究室

*** 図-1から6，参考文献は，先月号に掲載の“人工知能向言語（1）”を参照されたい．

プログラムへの連絡方法の ADIEU, AU-REVOIR という言葉の選択にも，単位プログラムのタイプを GENERATOR と命名したことにも表われている．この結果，下位プログラムは，問題を与えられ，上位プログラムが納得するまで，いろいろの〈結果〉を返し続けるという性格が強くなり，従って上位プログラムの解決機構のプログラムは，解を捜すのに繰り返しを多用することになる．紙面の都合上，図-5 を若干変更した例 (図-8) を中心に説明する．

CONNIVER の仕様にきちんと従がうと，繁雑になりすぎるので，細かな部分は省略し，記法も分り易いものにかえて利用する．説明上必要な関数の形式を以下に示す．

〈FETCH pattern context〉
〈ADD item context〉
〈TRY-NEXT possibility nomore message〉
〈ADIEU possibility〉
〈AU-REVOIR possibility〉

FETCH, ADD* はデータ・ベースへの読み，書きを表わし，Micro-PLANNER の GOAL, ASSERT に対応する．コンテクストを指定することにより，他に影響を与えることのないようにデータ・ベースの領域を制限することができる．FETCH は pattern に一致する item (リストで表わされる) と方法 (Method) をとり出し，pattern 中の変数と対応する値との対や方法のリストから成る可能性リストを値として返す．方法とは，Micro-PLANNER の定理に対応する．TRY-NEXT は実行されるたびに可能性リストの先頭から，1つずつ“可能性” (possibility) をとり出し，その内容に従って，変数に値を代入したり，方法を起動したりする．nomore は，可能性リストが空になった時に実行される関数で，message は対応する AU-REVOIR の値として伝えられる伝言である．ADIEU, AU-REVOIR は TRY-NEXT によって起動された方法プログラムの中で実行され，“可能性”を可能性リストに入れ，TRY-NEXT を逆起動する．その言葉通り，ADIEU は実行後，単位プロセスは消滅し，AU-REVOIR はそこで次の起動を持つ．よく用いられるコルティンの制御用の関数である RESUME に対応させ説明する．

〈RESUME id val〉

id は，呼び起す相手のコルティンのアイデンティファイア，val は相手の対応する RESUME の値として伝達されるものとする．TRY-NEXT は最初に方法を起動する時は，CREATE & RESUME の役割を果たす．id は可能性リストの先頭にあり，val は，最初はパターン照合による引数への代入として，以後は message による．ADIEU は RESUME & DIE, AU-REVOIR は RESUME に対応する．id は呼び起してくれた相手に固定される．val は，可能性リストに置かれる“可能性”である．

RESUME という関数の対称性が，コルティン間の対等な関係を反映するように，TRY-NEXT と ADIEU, AU-REVOIR の非対称性が単位プログラム間の上位，下位という階層関係を反映する．

図-8 の例に戻る．REMEMBER は FETCH によって取り出される IF-NEEDED タイプの方法である．

〈TRY-NEXT 〈FETCH 〔REMEMBERED PERSON-BEFORE-ME〕〉〉 が実行されると，REMEMBER プログラムが起動される．3つのコンテクストを考える．c_b は眼前の人の特徴を列挙したもの，c_p は自分に既知の人々の記述，CONTEXT はシステムが暗黙に仮定するもので，ここでは一時的なメモとして用いている．REMEMBER は，まず友人を仮定し，全ての友人につき顔の一致する人を見つける．いない場合は，昨日会った人と仮定し，同じ事を繰り返す．FETCH, ADD のデータ・ベースへの操作は，コンテ

```
REMEMBER:
  主題: 〔REMEMBERED    ?X〕
        〈TRY-NEXT 〈FETCH 〔SEX $X ?Y〕 Cb〉〉
FRIEND: P←〈FETCH 〔FRIEND $Y ?Z〕 Cp〉
  LOOP:
        〈TRY-NEXT P 〈GO P-M-Y〉〉
        〈ADD 〔NAME $X $Z〕 〈PUSH CONTEXT〉〉 ……①
        〈LOOK〉
           SUC→〈ADIEU REMEMBERED〉
           FAIL→〈POP CONTEXT〉……②
                 〈GO LOOP〉
P-M-Y: P←〈FETCH 〔RERSON-MET-YESTERDAY
                                  $Y ?Z〕 Cp〉
        LOOP:
                      ⋮
```

```
LOOK:
        〈TRY-NEXT 〈FETCH 〔NAME ?X ?Y〕〉〉
        〈TRY-NEXT 〈FETCH 〔EYE $X ?Z〕 Cb〉〉
        〈TRY-NEXT 〈FETCH 〔EYE $Y $Z〕 Cp〉
                              〈ADIEU FAIL〉〉
                      ⋮
```

図-8　モデル化の例 CONNIVER による

* データ・ベースの操作関数名 (FETCH, ADD, REMOVE など) や方法名 (IF-NEEDED, IF-ADDED, IF-REMOVED など) は Micro-PLANNER の様な特定の解釈に固定化するのを嫌い，ファイルへの入出力に対応する機械的な名前を採用している．

クストが明示されれば，そのコンテクストに，されない時は，CONTEXT に対して行われる．棚下げ(①)により新らしいコンテクスト層 (context lager) が付け加わわり，棚上げ(②)により取りはらわれる．その間のコンテクストに対する書き込み，削除の操作は全てこの層に対してのみ行われ，棚上げによりその操作の影響を一度に取り除くことができる．コンテクスト層の内容は，書き込み，削除の対象となったデータに対し，そのデータが存在する，しないという記述の集まりである．読み操作に対してはコンテクスト層を一段一段辿りながら，最初に出会ったそのデータに対する記述によって有無が決定される．コンテクスト層は任意の木構造に構成することができる．

CONNIVER は，Micro-PLANNER の欠点の克服法として，表現のレベルを下げた一般的な手立てを用意した．制御構造に対しては，最も一般的なモデルとして提案されたスパゲティ・スタック[4),5)]（ヒープをできるだけ線形化し，スタック化しようとしたことに由来）を用いて，多数の実行環境を保持し，処理を行えるようにした．データ・ベースとしては，スパゲティ・スタックのデータ・ベース版ともいうべきコンテクスト機構により，多数のデータ・ベースを実現できるようにした．つまり，モデルの記述手段としては，次の2つを新しく提案したと考えることができる．

(I) **単位プログラムと連絡手段**：自由な解釈を与えることができる単位プログラムを多数作り，相互の連絡手段も自由に設計できる手立てを用意した．ただし，既製品として用意された連絡手段，TRY-NEXT, ADIEU AU-REVOIR を用いるかぎり，先に説明したように階層関係に関するかなり強い制約を受ける．

(II) **データ・ベース単位プログラム群**：データ・ベースを管理するプログラムは，コンテクスト機構により，多数の単位プログラム群としてシミュレートされ，対象の変化の具合を客観視したり，問題解決過程で重要な仮説設定の機構を提供した．この他に，パターン照合機能の向上と整理が行われた．

この2つの機能を活用して，TOPLE[52)], HACKER[77)], BUILD[18)], MYCROFT[30)] 等のシステムが CONNIVER でプログラムが試みられた．

6. その他 (SAIL[20),82)], QLISP[68)])

SAIL は，ALGOL 60 を基本にし，そのサブセットとして，連想3つ組を扱う LEAP[19)] を含むもので，スタンフォード大学の人工知能プロジェクトの基本言語として作られたものである．後に，Micro-PLANNER や，CONNIVER に影響され，類似の機能を持った版が作られたが，あまり成功とはいいがたい．LEAP をサブセットとする ALGOL 60 というのであれば，1つの閉じた言語として，効率の良いプログラムを作る良い道具となる．さらに拡張するとすれば，同じく ALGOL を基本とする SIMULA-67[44)] の class 概念を拡張して導入した方が，より人工知能向言語としても，まとまりのある使い易いものになったであろう．

QA 4[11),69),70)]は，SRI，スタンフォード大学で製作が試みられた言語で，製作上の失敗から再整理され，QLISP という名前でまとめられた．定理証明を基本とした質問応答システムとして，QA 3[32)~34)] が作られ，さらに，定理証明のアルゴリズムの比較に重点を置いて，QA 3.5[29)] が作られた．次の段階として，高階述語論理導入の試みと PLANNER の出現とが相まって，手続きによる表現を中心とした QA 4 が提案され，特に目新しいものとしては，基本データ型として，集合とバグ (BAG) を導入したことである．集合は，順番が意味をもたぬリストで，同一の要素が，自動的に1つに縮退される．バグは，この縮退化が行われぬ集合である．

7. 知識の表現と利用のモデルの基本要素

人工知能向言語は，知識の表現と利用のモデルを記述する上での有用な要素を提供してくれた．

それでは，その使用経験や他の分野からの研究成果から，もう少し，モデルとしての輪郭を定めることができないであろうか．もちろん，非常に一般的で，しかも非常に強力な解決機構やそれと一体をなす，現実の世界との対応のつけ易い一般的な表現法を求め得る段階ではない．しかし，問題解決プログラムは，その性格上，ある程度構造が規定できるのではないか，事物と事物間の関係が，表現の基本になるが，全ての関係は等質ではなく，非常に重要な関係で知識を構成する骨組となるものがあるのではなかろうか．強力とはいいがたいが，解決アルゴリズムの各部で共通に利用できる推論アルゴリズムがあるのではないか．これらの問いに，人工知能向言語や定理証明や意味ネットワークの分野での成果をもとに，1つの解を提案するのが本章の目的である．

まず，全体の輪郭をはっきりさせる手助けとして，知識の表現のタイプに関する，手続き型 (procedural)

	宣言型	手続き型

①
数値計算の例
	$Z=X+Y+5$	LOD R, X
		ADD R, Y
		ADD R, '5'
		STR R, Z

② SHRDLU からの例	THEOREM [GRASP] PURPOSE (#GRASPING X) PRECONDITION INDEPENDENT (#MANIP X) DEPENDENT 1. (#GRASPING Y) →(GET-RIP-OFF Y) 2. (#CLEARTOP X) 3. (#MOVEHAND (TOPCENTER X)) EFFECT ADD (#GRASPING X) (DELETE)	(THEOREM GRASP (CONSE (X, Y) (#GRASPING $?$X) (GOAL (#MANIP $?$X)) (COND ((GOAL (#GRASPING $?$X)) (SUCCESS)) ((GOAL (#GRASPING $-Y)) (GOAL (#GET-RID-OFF $?$Y) (USE GET-RID-OFF))) (T)) (GOAL (#CLEARTOP $?$X) (USE CLEARTOP)) (SETQ $-Y (TOPCENTER $?$X)) (GOAL (#MOVEHAND $?$Y) (USE MOVEHAND)) (ASSERT (#GRASPING $?$X))))

図-9 宣言型，手続き型の対比

が宣言型 (declarative) かの議論[84),85)]を整理することから始める．PLANNER により明示された手続き型知識表現は，知識の見た目の表現より利用する側面に非常に威力を発揮した．しかし，システムが大きくなるにつれ，また能力をより高度にしようとするにつれ，より分り易く，知識の構造を反映する表現が求められるようになった．**図-9** に一例を挙げる．①は甚だ手続き型に不公平な例である．手続き型は，宣言型をコンパイルすることによって得られる．ただ計算値を得るだけならば，宣言型をコンパイルしたり，インタプリートするより，手続き型を実行する方がはるかに効率が良い．②の手続き型は文献 83) から借用した例である．ロボットの手が物体を摑むという知識を表現したもので Micro-PLANNER によって記述されている．しかし，このプログラムをよくよく整理すると左の宣言型としてまとめられる．摑むという動作を規定する項目は，大きく，その動作の目的，動作が実行できるために満されるべき条件，その動作の効果からなる．さらに，条件は場面に独立のものと場面に依存するものに分かれる．効果は，データ・ベースを対象にした場合，書き込み事実と削除する事実に分かれる．①と同様に，宣言型をコンパイルあるいはインタプリートすれば，手続き型の知識が得られる．更に，この知識をデータとして扱う，より高度の機能を考える場合，①において，数式処理を考える場合や，②において動きに関する知識を分析したり合成したりする場合（例えば，HACKER*）は，明らかに宣言型化が必要である．それでは，知識は全て宣言型で表現すべきであると“宣言”できるであろうか．宣言型を手続き化するには，項目間に実行順序を規定する規則を見つけ出さねばならない．適切な規則が見つけだせない場合は，あらゆる組み合わせの実行順序につき試みる．②の例については，図示した順序が最も自然で一般的なものであろう．しかし，条件の場面に依存するものとして3つの事項がある．この条件間の順番を規定する規則となるとあまり明瞭でなくなる．いかに宣言型とはいえ，このような部分は必ず残る．これに対しては，手続き型表現の自由さと豊かさを利用できるようにしておいた方がよい．

より本質的な事は，人間の持っている知識にも，本来，2つの型があるということである．先のホットケーキの作り方を思い出して頂きたい．手順を表わす知識は，当然の事ながら手続きとしてしか表わせない．

この手続き型か宣言型かの議論に，もう一段高い立場から“枠”組みを与えようとしたのが，Minsky のフレーム理論[53)] (Frame System Theory)** である．この理論は，人工知能研究の指導原理としては，非常に優れたもので，ある意味では，当然の事を述べているのであるが，あまりにも一般的すぎるため，具体的な解釈や適用[45)]になると，かなりまちまちの様である．混乱の原因の1つは知識のレベルをはっきりさせずに議論が行われることにある．例えば，積み木という知識といくつかの積み木によって構成される場面という知識，更に，ロボットが登場し，与えられるいろ

* 熟練していく（学習する）という事を新らしい状況に適合する様にプログラムを作り変えていく事と考え，簡単な積木の世界を対象にしたモデルである．

** 定理証明の枠の問題 (Frame Problem) の枠 (Frame) は，この理論のフレームのある特定の場合の限定された議論に対応するものと見ることができる．

積み木
　色　〈色彩〉　　球
　形　〈球〉∩〈立方体〉∩〈ピラミッド〉 ⟶ 色　青
　大きさ〈半径〉∩〈一辺〉∩〈底辺，高さ〉　形　球
　　項目間の関係　　大きさ5
　　(形と大きさの対応)
場面
　登場するもの　〈積み木の集合〉
　位置　　〈積み木の座標の集合〉
　相互関係　　〈相対的位置関係を表わす記述の集合〉
　　項目間の関係
　　(座標と位置関係との整合性)
　　(積み木の形と位置関係の整合性)
　　(積み木の大きさと座標との整合性)
　　(位置関係相互の関係)
計画
　初期状態　〈場面〉
　目標状態　〈場面〉
　解　　〈動作の系列〉
　指針　　〈忠告〉
　動作　　〈ロボットの基本動作の集合〉
　　項目間の関係
　　(目標状態が初期状態から出
　　発して，解の動作系列により
　　到達可能であるという証明)
　　〔解決機構〕

図-10　フレーム（例）

いろな問題を解いて行動計画を作るという知識，それぞれがフレームとして，同じ様な構造として表わされるが，その内容たるや格段の差がある．**図-10** に3つの知識に対するフレームの主要な部分を示す．

フレームの機能は，各項目に具体的な値が与えられたとき，項目の値域と項目間の関係に照合し，正しいか否か，まちがいの場合はその理由を返すとか，項目の一部を指定してすでに値の与えられている項目との関係で，どういう値なら良いかを返すとか，すでに与えられた値の一部が変化したとき，全体としての整合性を保てるようにするとかである．この様な機能を果すには，積み木フレームに対しては，人工知能向言語のリストのパターン照合機能＋α位で実現できそうである．しかし，場面フレームに対しては，定理証明＋α以上の問題解決機能が必要である．さらに，計画フレームに至っては，人工知能向言語で提供される機能を活用してプログラムされた高度な問題解決機能が必要になる．さらに，この3種のフレーム相互間を司どるものも，単純なアルゴリズムを仮定するというわけにはいかない．つまり，同じフレームというものの，全てに共通のプログラムをいくつか用意すれば，事たりるというわけではない．いわんや，具体的にプログラムとして実現しようとすると，更に，大きな差を生ずる．従って，適用する知識のレベルを明確にしなければ，フレームの議論は無意味である．さて人工知能向言語が提案してくれた諸機能を知識の表現と利用のモデルとして，より高度化，より具体化，すなわち，より構造化するにはどのようにすればよいであろうか．以下に構造化の方向をいくつか列挙し，説明する．これらは，現在，筆者が開発中の知識表現システムの中心となる考え方のいくつかである．

単位プログラム

すべての問題解決システムは，図-1の単位プログラム群として表わされる．構造化は，相互間の連絡方法を中心に進められる*．CONNIVER の TRY-NEXT, ADIEU, AU-REVOIR の片寄った連絡手段を矯正することから始める．まず，素直に図-1の外部から(へ)の8本の矢印を連絡方法としてまとめ，相互の関連をつけると，**図-11** のようになる．自分より上位，あるいは下位というように相手の階層関係を意識しない場合，つまり，相手をすべて対等と見る場合は，(a)と(c)，(b)と(d)は，同じものとなる．ここで行われ

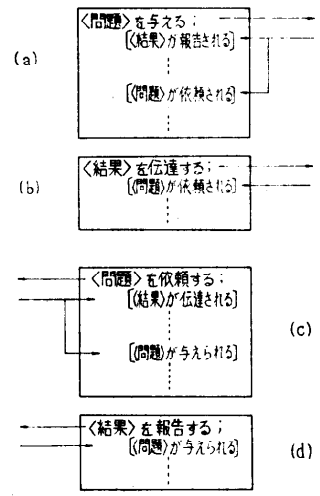


図-11　単位プログラム間の連絡手段

* プログラミング言語の分野でも，単位プログラムの性格と対応する連絡方法は，重要な言語要素である．単位プログラムが，サブルーチンの場合は他のサブルーチンを呼ぶのか，自分を呼んだルーチンに結果を伝えるのかを区別して，CALL と RETURN が用いられる．コルティンの場合は，相互の対等性を強調して RESUME という一つの連絡方法を用いる．並列処理プロセスの場合は，情報を送る場合と受け取る場合を区別する2つの連絡方法，例えば，ACTIVATE と WAIT などを用いる．

コルティンをもとにもう一段構造化を進めた単位プログラムとして SIMULA-67 が提案したクラス[9] (class) がある．このクラスを一般化し，メッセージを送ると受けとるの2つを基本の連絡方法としてプログラミング言語(システム)全体を体系づけようとしているのが，SMALLTALK や PLASMA[42),43)] (PLAnner like System Modeled on Actors) である．現在，筆者は，これらの全貌を明らかにする資料は持ち合わせていない．しかし，窺い知る限りにおいても，(人工知能向)プログラミング言語の基本機構として，なかなか重要な役割をはたす考え方であると思われる．ここでの単位プログラムの構造化も，この考え方を参考にしてのことである．

た連絡方法の構造化とは，

〈情情〉を送る；——→　と

——→〔〈情報〉を受けとる〕

という基本の連絡方法に対し，〈情報〉を〈問題〉と〈結果〉に，“送る”，“受けとる”を上位へのものか，下位へのものかの2種類に区別し，したがって，それぞれを4種類の連絡動作に具体化したものである．さらに，4種類の“送る”という動作と，その動作の返事を受けとる動作として，4種類の“受けとる”という動作が，任意の組み合わせを作るのではない．その妥当な組み合わせをも，図-11 は示している．

(a)について，交信の方法を説明する．“〈問題〉を与える；”が実行されると，相手(ここでは省略してある．)が，“〔〈問題〉が与えられる〕”という状態で，待っているとすると，〈問題〉どうしのパターン照合が行われ，一致すると，自分は停止し，相手は〈問題〉を受けとり，“〔〈問題〉が与えられる〕”以下のプログラムの実行に移る．次に，相手が，“〈結果〉を報告する；”，あるいは，新しく“〈問題〉を依頼する；”という文を実行すると，相手が停まり，先程の“〈問題〉を与える；”の下に並記されている，“〔〈結果〉が報告される〕”か〔〈問題〉が依頼される〕”かが，パターン照合の後に起動される．

```
単位問題解決プログラム：
  〔主題：〈目的〉〈機能〉〕
  〔方法：〈解を得る手助けをしてくれる単位プログラム群〉〕
  〔控え：〈過程の記録〉〕
  〔〈問題〉が与えられる〕
      〈問題〉を分析する；
      〈問題〉を部分〈問題〉に分割する；
      部分〈問題〉は，上に依頼するのか，下に与えるのか，自分で解くか，
          与える——→〈方法〉の中から適したものを選ぶ；
                    その〈方法〉に部分〈問題〉を与える；
                        〔〈結果〉が報告される〕
                              ⋮
                        〔〈問題〉が依頼される〕
                              ⋮
          依頼　——→部分〈問題〉を依頼する；
                        〔〈結果〉が伝達される〕
                              ⋮
                        〔〈問題〉が与えられる〕
                              ⋮
          解く　——→……
      〈結果〉を報告する；
          〔〈問題〉が与えられる〕
                ⋮
  〔〈結果〉が伝達される〕
      〈結果〉を分析する；
          ⋮
  〔〈問題〉が依頼される〕
          ⋮　(6～17 に類似)
  〔〈結果〉が報告される〕
          ⋮
```

図-12　単位プログムラの構造

この交信機構は，そのまま並列処理に拡張できるが，ここでは，特に必要としないのでコルティンとして説明をする．この交信機構を用いて，先の単位プログラム全体をより具体化すると図-12 のようになる．右側の数字は説明のためのものである．主題，方法，控えの共通の変数と，外部からの起動に対応する4つのパターンが最上層に用意される(5, 18, 20, 21)．ここでは“〔〈問題〉を与えられる〕”，〔〈結果〉を伝達される〕”の2つのパターンに対するプログラムを示した．20, 21 に対しても同様の構造が考えられる．これ以上，構造を細かくするには，個々のアルゴリズムを確定しなければ不可能である．

〈問題〉が与えられた場合を例にとり説明する．上位プログラムが“〈問題〉を与える；”を実行すると，〈問題〉どうしのパターン照合が行われ一致したならば，6からのプログラムが起動される．この時，上位プログラムからの〈問題〉で値が指定されない項目については，5の〈問題〉パターンの中で，あるいは，6以後のプログラムの中で，最も適切なものを仮定する機構を持つ．〈問題〉を受け取ると，さらに控えに照し合わせて，より詳細な分析が行われる(6)．5のパターン照合による〈問題〉の検査が，過程の履歴に依存しない，つまり文脈に依存しないものであるのに対し，6では，文脈を考慮した検査が行われる．検査を通過すると，その〈問題〉が一気に解けるものか否かが調べられ，解けぬ場合は，ある分割法を適用し，最初の部分問題を取り出す(7)．次にその部分〈問題〉を，自分で解く事ができるか，手持ちの〈方法〉に指示するか，上位のプログラムに依存しなければならないかを判断し，適切な処置をとる(8～)．何らかの方法で，部分〈問題〉が解けたとなると，次の部分問題に取りかかる．もとの〈問題〉が完全に解けたとなると．上位プログラムに〈結果〉を報告する (16)．報告をした後，再び“〈問題〉が与えられる”，つまり不満が伝えられるのを待つ(17)．再起動されると，〈問題〉の内容に従がい，他の解を捜したより詳細な〈結果〉を報告したり，などのプログラムが17以後に書かれる．

全ての問題解決システムは，相互に連絡し合う単位プログラム群としてモデル化する事ができる．勿論，全ての単位プログラムが，図-12 の全ての機能を持っているわけではな

い．それぞれの役割に応じ，適当な縮退化が行われている．今までに作られた種々のシステムを分析する事により，代表的な縮退化の方法を列挙する事ができるが，紙面の都合上割愛する．システム全体のモデルを作るには，単位プログラム同志の関連の仕方のモデル化が必要である．このモデル，すなわち関連の構造は，抽象一具体，全体一部分，一般一特殊などの関係を反映する．より具体的には，単位プログラムを登場人物（役）と考えた場合，一つの劇を構成するに必要な要素を列挙し，相互の関連を記述したもの，すなわち台本がこのモデルである．何がテーマで，主役，脇役，各役の劇中での役割，舞台の設定，粗筋などの項目を持つ．

対等に交信し合う2つの単位プログラム（図-12 の 20, 21 を取り除いたもの）は，会話・質問応答の簡単なモデルとなる．必要な個所を次のようにいい換えると明らかになる．

5 〔〈問題〉が与えられる〕
——→〔〈質問〉を受けとる〕
13 〈問題〉を依頼する；
——→〈質問〉をする；
14 〔〈結果〉が伝達される〕
——→〔〈答え〉を受けとる〕
18 〈結果〉を報告する；
——→〈答え〉を返す；

更に，〈問題〉すなわち〈質問〉の初期状態と目標状態は，

〈初期状態〉：自分が確かなものと思っている心の状態，質問を発する基盤，手がかりを与える．勿論，文脈を考慮したものである．

〈目標状態〉：相手の答えによって，こうなりたいという状態．

となる．

〈指針〉，〈制限条件〉，〈報告の形式〉などもいろいろの方法で表現される．いずれにしろ，質問は，〈初期状態〉を〈目標状態〉にしてくれるような答えを得られると予想される相手に対してなされる．図-12 に従って，質問応答モデルを詳述する．“方法”は相手に質問するのではなく，自分で考える時の手助けをする，即ち自問自答の相手となる単位プログラム群である．控えには，会話や自問自答の記録が残される．〈質問〉を受け取る(5)とまず文脈に依存しない質問としての適合性を調べる．次に，控えに照合し分析する(6)．この質問に答えられるか，〈初期状態〉と〈目標状態〉に矛盾があるか否か．すぐに答えられるものではないときは，この〈質問〉を部分〈問題〉に分割する(7)．この部分〈問題〉の内で，上に依頼するものが部分〈質問〉となる(13)．その質問に対し，答えを受け取る(14)か，その質問に対し再質問される(15)かである．答えが出来上がると，相手に応答し(16)，答えに再質問が来るのを待つ(17)．分割された部分〈問題〉が，独立のものであればよいが，そうでないものの場合は，その部分〈質問〉に対する〈答え〉を控えに照し合わせ，矛盾のあるときは，再〈質問〉をする．この質問応答モデルによって，システム内部の問題解決機構と，外部との質問応答機構とを，有機的に関連づけることができる．現在，筆者が開発中のシステムでは，単位プログラムは，一般性をもたせるため，もう少し低いレベルで，表記法もプログラミング言語としての体裁をもったものとして設計されている．ここでは，単位プログラムというものの考え方の有用性を示し，今後の構造化の方向を示唆するため，より具体的な単位プログラム像を紹介した．

解決機構

単位プログラムの解決機構は，すべて手続きとしてプログラムしなければならぬのであろうか．すべてに適用できるというのではないが，手続きより，簡便で，場合によってはより強力な機構はないであろうか．現在，このような組み込みの機構として，我々が利用できる最良のものは，定理証明，すなわち一階述語論理とレゾリューション原理 (Resolution Principle) による証明手続き[7] である．定理証明には，枠の問題をはじめとして，多くの未解決の問題がある．しかし，そのような問題が生じない範囲で使用するかぎりは，表現の一般性と安定した強力なアルゴリズムという点で，定理証明に対抗しうるものはない．問題点が生ずるような部分の表現は，人工知能向言語の手続き，データ・ベース，パターン照合などの機能によるべきである．このような考え方を実際に適用した例もいくつかあり，さらに強力に有効性を支持するのが，筆者らの行った定理証明機械の実験[87]である．

この実験は，構造が簡単で効率の良いユニット・レゾリューション・アルゴリズムを，マイクロプログラム可能な小型計算機の上に実現したものである．その結果，定理証明プログラムは非常に小さくなり，アセンブラで書いたのに比べ，マイクロプログラム化により一桁位の高速化が達せられた．この事実を，敷衍すると，もし最新のハードウェアを用い，プログラムを改良すれば，通常，定理証明に関する論文で用いられるような例題は，すべて 1 msec 以下で証明すること

ができるようになる．もはや，定理証明などという大仰なものではなく，一つの強力な信号処理用オペレーションとなる．

ここで述べたのとは逆の立場で，定理証明を利用しようとする方法もある．つまり，証明アルゴリズムを，解決機構の中心アルゴリズムに据え，枠の問題や概念の階層関係の処理や交換則の処理など，現在の定理証明が苦手とする問題を，推論規則を拡張することにより，理論的に解明する，あるいは実際のシステムとして定理証明を利用しようとする方法である．しかし，現時点で見る限り，理論的な面白さは別にし，その苦労の割には，得られるものが少ないように思われる．

データ・ベース

データ・ベース単位プログラムの解決機構は，CONNIVER が採用したインデックス機構とコンテクスト機構が主機能である．人工知能向言語では，データ・ベースに蓄える事実は，最も一般的なリストで，リストの構造や要素には，何ら解釈を与えない任意のものを前提としている．したがって，インデックス，コンテクスト両機構ともに，非常に一般的なものとなる．例えば，**図-13**（a）の例は，6つのリストとしてしか扱われない．しかし，知識は，もう少し具体的な構造をもっているし，読み・書きの操作の対象となるデータにしても，めったに変化しない種類のデータ，頻繁に変化する種類のデータというように変化の仕方も一様ではない．それでは，どのように構造化を進めればよいのであろうか．図-13 の例を用いて，既存の人工知能向言語の問題点と解決策を簡単に列挙する．

（1） 核となる属性と推論

知識の最小の単位は，LEAP が具体化した，対象，属性，値の3つ組である．すべての属性（あるいは関係）は等質ではなく，知識表現にとって核となる重要な属性があるはずである．例えば，意味ネットワークなどで取り上げられる概念の階層関係（IS-A 関係）などである．この様な核となる関係とそれに対する簡単な推論機能は，データ・ベースに組み込むものとすべきである[37),48),55),66)]．IS-A関係に対する推論には，大きく2種類がある．一つは，概念の含包関係に対する推移則である．例えば図示の事実から，（象 IS-A 生物）を推論することである．もう一つは，上位概念が持っている性質は，その下位概念も持つという推論である．例えば，（象 EAT 生物）である．既存の人工知能向言語では，これらを，パターン呼び出しの手続きとして表わすか，あるいは後者の推論に関してはコンテクスト機構で代用するかである．いずれも，非常に効率が悪い．特に，CONNIVER のコンテクスト機構は，頻繁に変更がなされることを前提にして作られたものである．図示の例の様にめったに変更が無いような知識に対しては，もっと簡便な機構を用いるべきである．例えば，属性名を変数名と見なしたときの LISP の変数と値を結びつける機構位で十分である．

```
(動物 IS-A 生物) (植物 IS-A 生物)
   (動物 EAT 生物)
(象 IS-A 動物)
   (象 HAS 長い鼻)
   (象 LIVE 熱帯)
        (a) アイテム

(IF-NEEDED (象 IS 性質)
   THEN (COND (象 LIVE アフリカ)
                ⟶ (象 IS 狂暴)
              (象 LIVE インド)
                ⟶ (象 IS 柔順)))
        (b) 方　法
```

図-13 データ・ベースの内容(例)

（2） 連想の種類

人工知能向言語のインデックス機構は，リストの構成要素をすべて同等に見る．いかなる連想検索にも応じられるように，あらゆる種類のインデックスを用意する．大量のデータを扱うには，インデックス機構の構造化を進め，不必要なインデックスを減らさなければならない．図示の例に対し次の様な連想検索を考えてみよう．

（？ HAS 長い鼻）

つまり，

「何が長い鼻を持っていますか．」

という質問である．

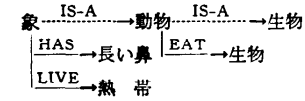
さて，返ってくる答えの範囲を想定しないでなされる質問などあるだろうか．森羅万象に対して，長い鼻を持っているか否かを問いかけても意味があるだろうか．むしろ，答えの予想があって，はじめて質問は意味をもつと考えるべきではなかろうか．つまり，

（動物？ HAS 長い鼻）

「動物の中で，何が長い鼻をもっていますか．」

という質問と考えるべきである．

そこで，インデックス機構は，指定された検索範囲を有効に利用できるようにしなければならない．例えば，図示の（対象，属性，値）の3つ組について，一番簡単な機構は，対象に対するインデックスのみを用意するものである．つまり，



である．対象が明示されている検索（象 HAS ?）は，このインデックスにより，すぐ答えがでる．（動物? HAS 長い鼻）は，“動物”から IS-A をたどり，すべての下位の対象に対して，問いかけを行う．

（3） 単位プログラム化

Micro-PLANNER が考案した手続きのパターンによる呼び出しの考え方は，簡単なリスト同志のパターン照合とさらに照合機能を強化するための手続きとを有機的に関連づけるものである．しかし，入出力操作に対応する3種類に手続きを類別したものの，すべての手続きは，呼び出しパターンをアイテムと見たてて，インデックス機構の一様な管理のもとに置かれ，何に関する手続きであるかを，明示する簡単な方法がない．これは，手続きがふえるに従がい，急激に効率が悪くなるばかりでなく，アイテムや，関係する手続き群，相互の関連を不明瞭にする．例えば，図の(b)は，「象の性質は」という質問に対し，生息場所からいろいろの性質を答えとして返す手続きである．通常のインデックス機構では，手続きのタイプ，“象”，“IS”，“性質”に関するインデックスがすべて用意される．しかし必要なのは，手続きのタイプと“象”に関するものだけである．そこで，“象”に関するアイテムと一体化し，次のようにまとめる．

```
{〔主題: 象〕
 〔控え: IS-A 動物
        HAS 長い鼻
        LIVE 熱帯 〕
  〔質問: (IS 性質) を受けとる〕
        (手続き)
  〔質問: ……〕
        (手続き)
  〔質問: ……〕
        (手続き)          }
```

つまり，先の単位プログラムとして，各対象を表現する．対象は，答えうる質問の集まりとして表現される．この様にして，呼び出しパターンのインデックスを省略し，システム全体を，単位プログラムという一種類の単位で構成される美しい体系にまとめることができる．

8. むすび

自然言語処理をはじめとして，多くの人工知能研究の基盤として，知識の表現と利用システムの研究・開発は，非常に重要である．この研究をさらに支えるものとして，次の3つの大きな柱が考えられる．

① 人工知能向言語に代表されるソフトウェア技術（計算機技術）[2),12),28),72)]

② 言語学・心理学などの成果

③ 述語論理・様相論理・Fuzzy 論理などの形式論理

これらは，三つの代表的アプローチとして，それぞれが自分の長所と他の短所を指摘するという種類の研究が多かったが，最近では相補い合うものとして，有効に利用していこうとする姿勢が広まってきたようである．

ソフトウェア技術は，動くシステムとするためのプログラム作りの道具を与えてくれるという実利的な面は当然のことながら，重要なことは，他の分野が持っていない，動きや変化に対する表現手段を与える．言語学や心理学は，提供された抽象的な表現手段に，より具体的，しかし非常に一般的な解釈を与えてくれるし，言語現象や心理現象との対応づけによって，表現手段の妥当性の論拠を与えてくれる．形式論理は，有効に表現しうる範囲は狭いが，その範囲内では，非常に一般的で整った表現の形式と，推論規制を与えてくれるし，さらに強力な証明手続きを与えてくれるものもある．知識の表現と利用に関する研究は，まだ始まったばかりである．この三つの柱の上に，今までの研究成果を整理して，システムを作り，その上により具体的なかなり大きな質問応答システムなどを作り，その結果をフィードバックするという，地道な，しかも現在の計算機（技術）を最高度に駆使する研究である．筆者のもう一つの期待は，このような着実な人工知能の研究の成果が，逆に三つの柱となる分野に，新しい展開をもたらすであろうということである．

最後に，西野博二部長と渕一博室長に，日頃の御指導を，推論機構研究室の諸氏に日頃の御討論を感謝しむすびとする．

（昭和51年6月7日受付）