A 4　人工知能のプログラミング言語

古川康一　（電子技術総合研究所）

1. はじめに

人工知能の研究を進めるための言語として，近年各所で新たな言語が開発されてきた。それらはPLANNER[1)]，CONNIVER[2)]（MIT），QA4[3)]，QLISP[4)]（SRI），new-SAIL[5)]（SU），POP2[6)]，POPLER[7)]（Ediuburgh 大学）などである。

これらのプログラミング言語は，robot planning，定理の証明，プログラムの自動作製などの研究をその対象としている。つまり，それらの領域に対する"問題向き"言語であるといえよう。"問題向き"の程度は，各プログラミング言語によって多少の差違はあるが，それは手段の相違であって，目的とするところは同じである。ここでは，まずこれらの問題向き言語のもつ諸機能が，どのように"問題向き"であるかを調べることから始めよう。ここで問題をより明確にするために，例をあげて考えよう。それはMonkey and Banana Problem[8)]と呼ばれている問題で，次のように表わされる。「おりの中に猿がいて，天井からバナナがつるされている。猿がバナナを取るためには，箱の上に乗らなければならない。猿，箱およびバナナの位置をa，b，cとするとき，猿がバナナを取るまでの一連の行動を求めよ。」

まず，この問題を計算機上で表現することが必要である。問題を表現するには，おりの中の状態の記述と猿のとる各種の行動の記述が必要となる。一般に，前者をモデルといい，後者を作用素という。作要素は，モデルに作用し，その結果モデルが変更される。問題は，初期モデルから目標モデルに変更するような一連の作用素を求めることである。モデルの記述には，問題を解くために必要な情報が含まれていればよい。いまの例では，それは猿の位置，箱の位置，バナナの位置，あるいはそれらの相対的位置関係，猿が箱の上に乗っているかいないかなどがモデルを構成する。これらの各"事実"は，n組によって表わすことができる。たとえば（AT.MONKEY A），(OFF MONKEY BOX)，(NEXTTO MONKEY BOX) などである。作用素には，GOTO (x)，PUSHBOX (X)，CLIMBBOX，GRASP の4つがある。解に到る一連の作用素を見出す一般的な手法のひとつに，問題をよりやさしい問題におきかえて解く方法がある。初めの問題をgoalとすると，よりやさしい問題におきかえることをsubgoal を設定するという。GPS[9)]　STRIPS[10)] などは，このsubgoalの設定をシステムが自動的に行なう機構をもっている。すなわち，GPSやSTRIPS は，この種の問題の解を与える"一般的な手法"をもとにしたシステムで，その場合，問題の入力は，すべての作用素の記述，および初期モデルの記述のみでよい。しかし，これは利点であると同時に欠点でもある。それは，個々の問題がもっている問題を解くための個別の情報を利用できないからである。

一方，PLANNERやQA4は，一種のプログラミング言語で，システム内には，subgoal を自動的に設定するような機能は全くない。それらはすべてプログラムで指定される。これは，ある意味で低レベル化であるが，その低レベル化によって，より複雑な問題の取り扱いが可能となる。

つぎに，この問題のPLANNER によるプログラムの概観を調べてみよう。

初期モデルの作製は，assert 文によって行なわれる。たとえば

```
(ASSERT' (AT MONKEY A))
(ASSERT' (OFF MONKEY BOX))                    (1)
```

などである。goalの設定はgoal 文

(GOAL ' (MONKEY GETS BANANAS))　　(2)

によって行なう。各作用素の記述はconsequent文によって行なわれる。goal－subgoalの定式化で，goalが設定するsubgoalはnon－deterministicであり，一般には複数個のsubgoalが設定可能で，その中の一部が解を構成する。このような選択過程を許すために，goal文はconsequent文を直接呼ばずに，その呼び出し過程にパターン合わせを使っている。このパターン合わせの過程は，GPSやSTRIPSでのsubgoalの設定のための論理計算に相当している。各consequent文は，そのための結果パターンをもっていて，goal文の引数とパターン合わせが行なわれる。そして合致がとれた場合にのみそのconsequent文が実行される。

PLANNERの各文を実行すると，成功または失敗の値をとる。consequent文が実行された結果が成功であった場合，その結果パターンとパターン合わせが行なわれたgoal文の引数が成り立つことを意味する。この過程は，implication「AならばBである」を手続き的に表わしたものであるといえる。PLANNERでは，これを定理と呼んでいる（PLANNERの定理には，この他にantecedent型，およびerasing型の定理がある）。いま，例として，Operator " grasp "を考えてみよう。それは，PLANNERでは次のように書かれる。

```
(CONSEQUENT (X)
    (MONKEY GETS ?X)                    (3)
(GOAL ' (UNDER BOX ?X))
(GOAL ' (ON MONKEY BOX)))
```

次にプログラムの動きをみてみよう。モデルが(1)によって作られると，goal文(2)を実行することによって，まず，そのgoalがモデルの中で成り立っているかどうかが調べられる。これは，モデルを表わしているデータ・ベースに対して，goalの引数とのパターン合わせによる検索を行なうことに対応している。もしそのモデルがすでに(MONKEY GETS BANANAS)をデータとして含んでいれば，このgoalは達成されたことになる。さもなければ，consequent文の中から，このgoal文の引数と合致する結果パターンをもったものを探し出し，その文を実行する。この探索自身，データ・ベースに対する探索と全く同様に行なわれる。これはいいかえれば，consequent文の集合も，データ・ベースのようにみなされていることを意味する。

いまの例では，(3)の定理が実行される。実行に先だって，パターン合わせによってXにBANANASが拘束（bind）される。(3)の中に，再びgoal文が現われている。このgoal文が初めのgoal文と同様に実行される。ここではgoal文が2つあるが，これは，その両方が成功しなければならないことを示している。ここに現われたgoalは，初めのgoalに対するsubgoalと考えることができる。この2つのsubgoalの順序は重要な意味がある。もし，第2のgoal文を先に実行すると，猿が箱の上に乗ることは保障されるが，一般には箱はバナナの下にはない。一階の述語論理による定式化では，このような順序をシステムに知らせることができない。これは問題のもっている個別の（意味的）情報のひとつである。

以上見てきたように，人工知能用プログラミング言語の特徴は，モデルを表現するための連想的データ・ベースをもっていること，パターン合わせの機能，選択過程を処理するための，新しい型の制御構造，推論の機能，などである。以下の各節では，これらの各機能の具体的な機構について検討を行なう。

2. データ・ベース

パターンを指定してデータを探索する連想データ・ベースとしては，SAIL[11]の連想3つ組データ構造がある。そこでは3つのitemが関連づけられて，1つの事実を表わすようにできている。それは，

item 1 ⊗ item 2 = item 3

と表わされる。通常3つのitemには，それぞれのattribute，objectおよびvalueという意味が割り当てられており，たとえば，

COLOR ⊗ APPLE = RED

のように使われる。データ・ベースに対する連想的な探索は，次のようなFOREACH 文によって行なわれる。

FOREACH× SUCH THAT COLOR ⊗ X = RED

DO statement

この文は，赤い色をもつすべての物に対してDO以下を実行する。ここでははは，item が拘束される拘束変数である。

SAILのデータ・ベースの欠点は，基本的には3つ組しか扱えないことと，各時点で1つのモデルしか表わしえないことである。これに対して，PLANNER，CONNIVER は，任意のn組をも扱える。またQA4，QLISPは，（順序）n組の他に多くのデータ型をもっている。それらは，（QLISPでは）TUPPLE，VECTOR，CLASS，BAGである。TUPPLE とVECTOR は共に順序n組であり，それらは評価のされ方だけが異なる。CLASSは順序づけられていないn組で，重複は取り除かれている。BAGは重複を許す順序づけられていないn組である。

パターンを指定して行なう連想探索を効率良く実行するように設計され，あるいは具現化されている機構には2つある。そのひとつには，QA4 1QLISPで使われているDiscrimination net （D － net）で，他のひとつは，PLANNER で提案されたCoordinate indexing である。

D－net は図1のような木構造によって構成される。この図で○の中の数字は，各データをリストと見たときの先頭から何番目の要素をそのnode からの分岐に使うかを示している。0は型を表わし，以下の成分は順に1,2,・・・・・で参照される。各枝にある情報は，その成分を表わしている。⓪の左下のnodeではいきなり2番目の成分を調べているが，それは1番目の成分が等しいからである。D－net ではすべての型のデータは，その中で置かれる場所が一意的に定まる。ゆえに，LISPのatom と同様に，これらの型のデータにもProperty － list （P－list）を置くことができる。つまり，D－net には，そのデータのP－listのhead cell が置かれることになる。この意味で，これらの型のデータをatom の拡張とみなし，structured atom と呼ばれる。CLASSやBAGはその成分が順序づけられていないので，D－net 内で一意的に表現するためには，標準形に直す必要がある。それらは通常辞書式順序にされる。

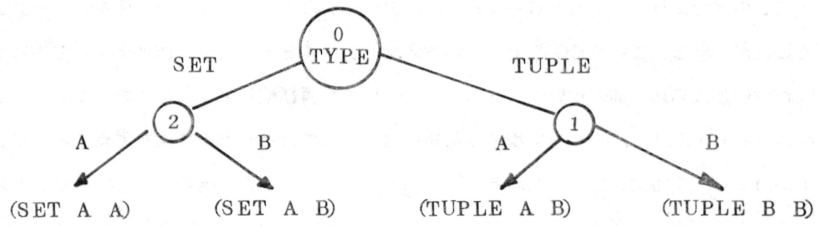

図1 Discrimination net

一方，coordinate indexing は，n組の何番目（cordinate）の成分が何であるかをキーにしたhashing により索引表を作る方法である。この方法は，探索のパターンが単純で，一度の探索で答が得られるような場合には，効率が非常によい（それはデータ・ベースの大きさによらない）が，1つのパターン中に変数が複数個存在する場合には，2回以上の探索を行ない，その結果に対して集合演算をしなければならず，効率は低下してしまう。

また，D－net のように各データがシステム内で一意的に決まる機構をもっていないので，データの外部表現によって決まる属性をP－list としてもつことができない。たとえば，（TUPPLE 1 2 3）に，（CONTAINS－PRIME TRUE）という属性を与えたい場合，一度それを与えても，別のところで（TUPPLE 1 2 3）が現われたとき，そのデータは新たに作られ，初めに与えた属性を参照することができない。

CONNIVER， QA4 ， QLISP は，同時にいくつかの異なったモデルを表現するための機構をもっている。それはcontext mechanismと呼ばれている。まず初めに，context をもったデータ・ベースの概要を説明しよう。それは，context frame （C － frame）と呼ばれるモデルの一部分を表わすデータの集合を節とする木として表わされる。いま，図2のような木を考えてみよう。変数CONTEXT がcf_1 を指しているとすると，この変数の示すcontext は図3のような部分木である。図の矢印は，データの探索の順序を示している。

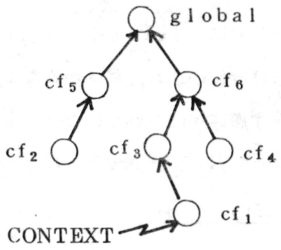

図2 C－fram の木

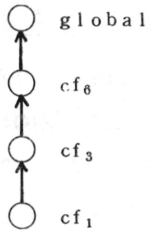

図3 context

C－frameは，モデルに対してなされた変更を示すために用いられる。データ・ベースの初期状態はglobal で与えられ，それが表わすモデルにcf_6で表わされる変更を加えたものが（cf_6，global ）というcontext で表わわされる。それに対して順次cf_3，cf_1の変更を加えて得られたcontext が図3に示されているものである。

各C－frameの中では，データ（n組）は，＂生＂，＂死＂，＂無指定＂のうちのいずれかの状態をとる。そのとき，ある特定のcontext では，すべてのデータは＂存在＂か＂不在＂かのいずれかの状態をとる。それはcontext のC－frame を，そのデータの属性が指定されているところまでたどり，その指定が＂生＂であれば＂存在＂で，＂死＂であれば＂不在＂となる。もしすべてが無指定であれば＂不在＂である。

このようにして，この木でより上位にあるC－frame 中のデータに手をふれないで，任意のデータを＂加え＂たり，＂消し＂たりすることができる。ここでCONTEXT の内容をcf_3にすると，モデルに加えられていた変更cf_1は全く取り除かれて，cf_1が付け加えられる前の状態に戻される。cf_1は後で再び使用するために取っておくこともできる。

このようなcontext mechanismを実現するためには，そこで行なわれる探索を考えてみればよい。その探索は，パターンとcontext を指定して行なわれる。context はC－frameのリストであり，リストの各成分とパターンを探索のキーと考えることができる。その場合，context についての探索は，C－frame の木を上向きにたどる順序で行なわなければならない。その過程で，探索キーのうち，C－frameだけが変更されていく。このような場合には，初めにキーの不変部分であるパターンによって探索し，次にcontext を調べるのがよい。そのためには，各データにcontext 情報をもたせることが必要となる。実際には，C－frameの番号を属性と一緒にしたもの，つまり（C－frame indicator value）をP－list 上におけばよい。1つの structured atom のP－list上には，同じindicatorで，C－frameによってvalueが異なるものが存在してもよい。探索の手間を減らすために，P－list上で，属性はC－frameの降順におかれる。データの生，死を示すindi－

catorはQA4では"MODELVALUE"であり，そのvalueは"TRUE"あるいは"FALSE"である。使用者は，この他に任意のindicatorをもった属性をおくことができる。

contextをもったデータ・ベースでのもうひとつの大事な点は，garbage collectionの機構である。backtrackingをこの機構によって行なうことを考えると，contextがpopされたとき，はずされたC－frameは，もし再び使わなければgarbage としなければならない。このために，システムはデータ・ベース専用のgarbage collectorを必要とする。それはC－frameを参照している変数からmarking を始め,markされなかった属性をP－listから取り除くという作業を行なう。

3. 制御構造

人工知能用プログラミング言語の制御構造の特徴は，backtrackingやcoroutine制御といった，階層的な制御構造からはみ出た複雑な構造を有していることである。階層的な制御構造では，新たなモジュールの活性化とその終了は，call および return でなされる。再帰的呼び出しは，同一のモジュールの全く異なる活性化とみなされる。このような制御はスタックで実現される。この場合，モジュールから制御が離れるのはreturn のときだけで，それによって，そのモジュールは死んでしまう。すなわち，そこでの実行過程はすべて失われてしまい，そのモジュールの実行を後で再開することはできない。そのモジュールを再び呼び出すと，全く別の活性化として扱われ，初めから走り出す。

このような機構のもとでは，backtrackingやcoroutineの制御構造を作り出すことはできない。backtracking を行なうためには，たとえreturnしても，その活性化に伴ってとられたスタック上の情報は捨てるわけにはいかない。また，coroutineでは，return 以外の命令で制御を他のモジュールに移すことが要求される。このような制御構造を作るには，2次元スタックが必要となる。これはheap上に作るのが最も簡単であるが，効率が落ちるので通常のスタック上に作る試みがなされている。[12)]

制御構造の持つもうひとつの側面は，変数のscopeすなわちアクセス構造との関係である。ALGOLのアクセス構造は，プログラムの静的なブロック構造によって決まる。ＬＩＳＰでは，制御構造とアクセス構造は同一である。しかし，いくつかの例外的な場面では，必ずしも上述のとおりになっていない。ALGOLでのcall by name の処理や，ＬＩＳＰのFUNARG処理では，アクセス構造を変えなくてはならない。coroutineやbacktrackingのように制御が必ずしも階層的でない場合には，アクセス構造と制御構造をそれぞれ独立に管理できることが望ましい。

このような柔軟な制御を可能にするモデルとして，BobrowとWegbreit[13)] は次のような機構を提案した。それはモジュールの活性化に必要な情報の構造を与えているもので，その構造を図4に示す。この図でbasic frameは，そのモジュールの活性化に伴って作られるもので，局所変数（ＬＩＳＰでいえばLAMBDA およびＰＲＯＧ変数）のbinding 情報をもっている。その下のframe extension は，そのモジュールが他のモジュールを呼び出

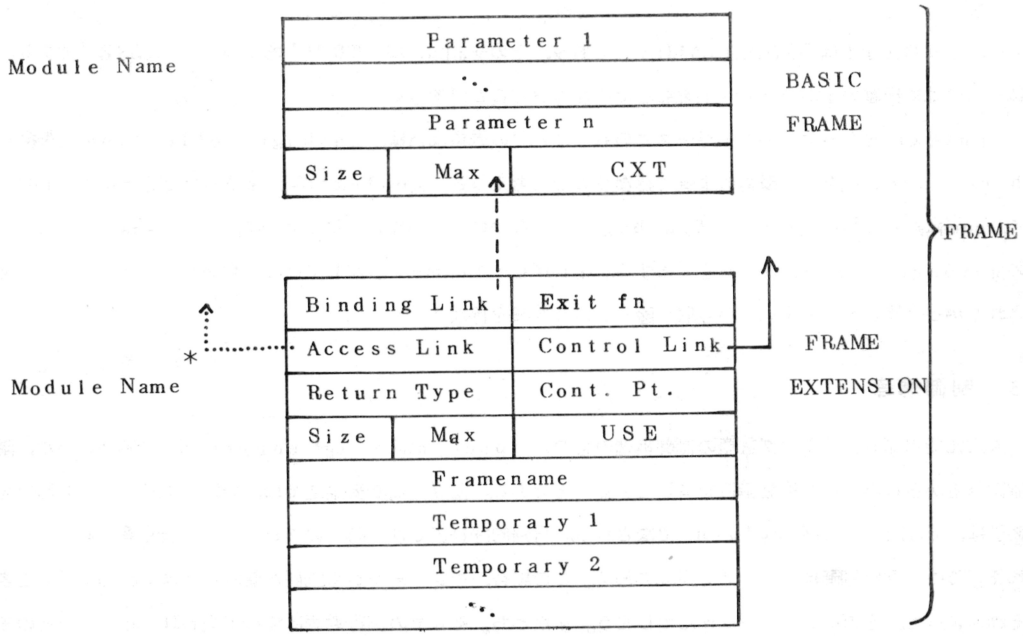

図4　モジュールの活性化に必要な情報の構造

すときに作られる。frame extension 中のAccess Linkは，アクセス構造を作り，Control Link は制御構造を作る。Cont. Pt.は他のモジュールから制御が渡されたときの実行開始点を表わしている。frame をbasic frameとframe extensionに分けたところが大切な点である。こうすることによって，図5に示すように，親モジュールを共有するcoroutineを簡単に作り出すことができる。この場合，Aにある局所変数は，BあるいはQのどちらのプロセスからも変更されうるし，そのような変更はどちらからでも参照できる。Bobrow らは，これらのframeの操作のための7つの基本関数を提案している。これらを用いて，より高度な制御機構を具現化することができる。この機構はCONNIVER, INTERLISP, POPLER にとり入れられている。つぎに複雑な制御機構の例として，CONNIVER で記述されたHANG の例を説明しよう。HANG は一種の割り込みモジュールである。それは，あるパターンのデータがデータ・ベースにつけ加えられた時点で割り込み，ある処理をしてから，再び制御をもとのモジュールに戻す。たとえば，

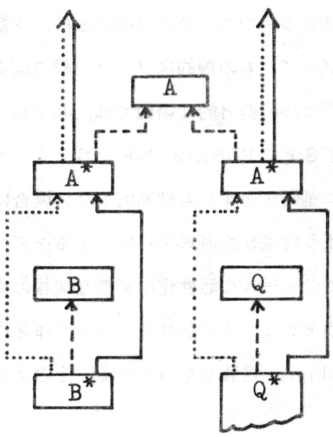

図5　親モジュールを共有するcoroutine

```
(PRO  "AUX"  (X WAYOUT)
    (CSETQ WAYOUT
        (HANG " (IFADDED  (?X BERG) " (GO
            "USEFULWORK) ) )
    (PRINT " (SOME OF MY BEST FRIENDS ARE BERGS) )
    (EXIT T WAYOUT)
```

というプログラムでは，制御の流れは図6のようになる。ここで，HANGは次のように定義される。

```
(CDEFUN HANG (RELEASE EXPRESSION)
            "AUX" (VALRET (C (CONTROL)))
    (ADD (CLOSURE
            (CEVAL (CONS (CAR RELEASE)
                         (CONS (CADR RELEASE)
                               '("AUX" ((F (FRAME)))
                                (CSETQ VALRET F)
                                (GO 'HANGRET))))))))
    (CEVAL EXPRESSION C)
:HANGLET
    (RETURN VARLET))
```

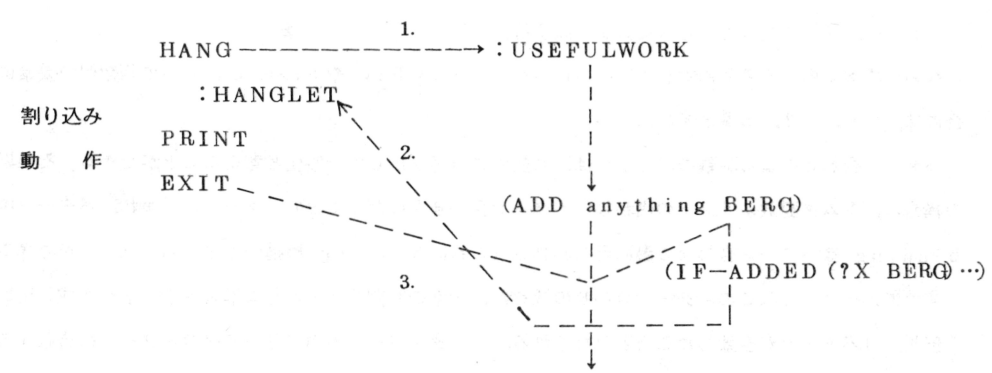

図6　割り込みモジュールをもつプログラムの制御の流れ

ここで，CLOSUREはLISPのFUNARG の拡張で，関数とその中への変数を評価する環境の対で表わされ，データ・ベース中におかれる。そして，PLANNER の定理と同様に，パターン合わせによって呼び出される図6の1のGOは，HANG を呼んでいる環境で実行される。このためにCにControl link の情報を置いて，（CEVAL EXPRESSION C）を実行している。2はHANG の環境が必要であるが，これはCLOSURE によって達成されている（frameが指定されていないときは，current が仮定されている）。3の制御は，HANG が，IF－ADDEDのframeを値として返していることによる。そのframeは（ADD anything BERG）から呼び出されており，したがって，（EXIT T WAYOUT）は割り込まれた（ADD……）の次に制御をもどす。

この種の割り込みモジュールはdemonと呼ばれ，データ・ベースの細かい管理や，debugging，error 処理などに使うことができる。PLANNERのantecedent型とerasing型定理もdemon の一種であるとみなすことができる。QA4／QLISPでは，あるdemon に対して，それが起動されるような関数（データ・ベースへの格納，参照のための関数）の集合を設定することができる。

4.　パターン合わせと推論機構

パターン合わせの機能は目新しいものではない。SNOBOL 4 やCOMITには記号列に対するパターン合わせの機能をもっているし，CONVERT は，対象をリストに拡張している。それらの言語では，パターン合わせは入力記号列の部分記号列への分解および各部分記号列の書きかえによる出力記号列の作製という使い方が主である。しかし，人工

知能用言語の中では，それは主にデータ・ベースの連想的探索に用いられている。探索機構はデータ・ベースの構造に依存する。ここでは，それの詳細にはふれずに，外部仕様の簡単な説明を行なう。

パターン合わせを行なうモードは，関数の評価を行なうモードと異なり，変数と定数の扱いが逆になる。すなわち，atom 自身は何もつけず，変数を表わすためにprefix をつける（評価のときは，atom 自身を表わすときには"QUOTE"をつけ，変数には何もつけない）。

prefix には数種類のものが用意されていて，変数がパターン合わせでどのように処理されるかを区別するのに用いられる。ここで，QA4／QLISPの例を挙げて説明しよう。用いるprefixは←，？，$の3つである。パターン合わせの際，それらは次のような規則に従がう。

←X　は任意の値に割り当てられる。

？X　はもしそれが未だ値を割り当てられていないときには，任意の値が割り当てられ，すでに与えられた値を変更することはしない。

$X　は，すでに割り当てられた値を参照する。

これらはすべて単一の要素に対して用いられるがclassやbay やtupple において複数個の要素に対応させる場合には，←←，？？，$$とすればよい。

パターン合わせによる関数の呼び出しは，名前による呼び出しの一般化と考えることができる。その場合，その関数の構造は，ラムダ変数のリストの代わりにパターン合わせを行なうためのパターンが置かれ，パターン中の変数のbinding はパターン合わせの際に行なわれる（これはbinding 機構の拡張と考えることができる）。

すでに述べたようにこのパターン合わせの機構は，新型の制御構造とともに推論機構の土台を成している。その場合"証明"はパターンが合致したこととみなされる。データ・ベースの中で証明すべきパターンに合致するデータがない場合には，そのパターンに合致するパターンをもった関数が探索され，実行される。この探索は時間がかかるので，それを容易にする種々の工夫がこらされている。PLANNER では，goal 文に呼び出される定理の推薦リストを置くことができる。QA4では，関数はクラス分けされ，goal 文は，クラスを指定できる。

選択過程の処理には，PLANNER の自動後戻り制御による方法と，CONNIVER などのcontext 機構と多重プロセス制御による方法がある。自動後戻り制御はgoalsubgoal による定式化のかなめであるが，Sussman[14]の指摘したように，この構機はしばしば非常に能率の悪いプログラムを作ってしまう。一方，CONNIVER 流のやり方では，制御およびデータの管理はプログラマに任せられており，能率の良いプログラムを作ることができるかわりにプログラミングが難しくなる。

5. むすび

人工知能用プログラミング言語の持つべき機能と，それらが実際にどのように具現化されているかをみてきた。新たに加わった諸機能は，実は目新しいものはほとんどない。それらのひとつひとつの特徴を持ったシステムは存在している。たとえば，パターン合わせはCOMITやSNOBOL 4 にあるし，coroutineなどの制御機能はSIMULA などのシミュレーション言語がもっている。しかしながら，それらを巧妙に組み合わせて，推論機構をもった言語を作り出しているところに価値があるといえよう。それらの諸機能の組み合せは，プログラミング言語としての発展にも寄与していると考えられる。新たに導入された機能の多くは，全く新たな概念としてではなく，これまでの言語の持っていた概念の拡張としてとらえることができた。このことは，プログラミング言語としての連続的発展を可能にする。QLISPがLNTER LISP[15]とQA4の融合体として実現できたのも，このような考察があったからであろうと思われる。

最後に，この調査報告を書くにあたっては，すでに出された調査報告16)，および17）が大変参考になった。興味のある読者は，むしろそちらの調査報告を読むことをおすすめする。16）は，推論に進点をあてた大変含蓄に富んだ，わかりやすい調査報告である。また，17）は，new SAIL, PLANNER/CONNIVER, QLISP/INTER LISP, POPLER についての詳細な比較検討を含んでいる。

参 考 文 献

1. Hewitt, C. Description and theoretical analysis (using schemata) of PLANNER : A language for proving theorems and manipulating models in a robot. AI Memo No. 251, MIT Project MAC, April 1972.
2. McDerwott, D. V. and Sussman, G. J. The CONNIVER reference manual. AI Memo No. 259, MIT Project MAC, May 1972.
3. Rulifson, J. F., Derksen, J. A., and Waldigner, R. W., QA4 : A procedual calculus for intuitive reasoning. Artificial Intelligence Center. Technical Note 73, SRI, 1972.
4. Reboh, R. and Sacerdoti, E. A PRERIMINARY QLISP MANUAL. SRI AI Center Technical Note 81, August 1973.
5. Feldman, J. A. et al. Recent developments in SAIL —— An ALGOL-based language for artificial intelligence, FJCC, 1972.
6. Burstall, R. M., Collins, J. S., and Poppleston, R. J. PROGRAMMING IN POP-2. Edinburgh University Press, 1971.
7. Davies, D. Jurian M. POPLER 1.5 REFERENCE MANUAL. University of Edinburgh, TPU Report No. 1, May 1973.
8. Nilsson, N. J. Problem-solving methods in artificial intelligence. Mcgraw-Hill, 1971.
9. Ernst, G. and Newell, A. GPS : A case study in generality and problem solving, ACM Monograpb Series. Academic Press, 1969.
10. Fikes, R. E. and Nilsson, N. J. STRIPS : A new approach to the application of theorem proving to problem solving. Artificial Intelligence Group Technical Note 43, SRI. 1971.
11. Feldman, J. A. and Rovner, P. D. An ALGOL-based associative language. CACM 12, 8, PP. 439-449, August 1969.
12. Bobrow, D. G. and Wegbreit, B. A model and stack implementation of multiple environments. CACM, 16, 10, October 1973.
13. Bobrow, D. G. and Wegbreit, B. A model for control structures for artificial intelligence programming languages. Proc. of IJCAI, Stanford, California, August 1973.
14. Sussman, G. J. and McDermott, D. V. From PLANNER to CONNIVER —— A

genetic approach, Proc. FJCC, PP. 1171－1179. 1972.

15. Feitelman, W. BBN－LISP――TENEX reference manual, BBN, Inc. 1972.

16. 渕　一博・推論機能をもつプログラム言語　昭和４８年電気四学会連合大会予稿集

17. Bobrow, D. G. and Raphael, B. New programming languages for AI research. Artificial Intelligence Center Technical Note 82 SRI, 1973.

本 PDF ファイルは 1965 年発行の「第 6 回プログラミング―シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html