定理証明用言語 MICRO-PLANNER について†

島　田　俊　夫††

1. まえがき

なんらかの意味で知能を必要とする問題を解く能力をプログラムの形で記述しようという試みは，かなり古くから行なわれてきた．

初期のころは，ある特定の問題，たとえばゲームとか，数学の定理の証明などを目標とし，かなりの成果をあげた．しかし，このように限られた問題で用いられた方法やシステムを他の問題に応用することは大変難しく，もっと汎用性のある方法やシステムを見つけることが次の目標となった．この問題を追求し，システムの形にまとめあげたものに Newel 等の GPS[13] (General Problem Solver) がある．

GPS は goal-orientd な問題の記述，back-up 機構，マッチングによるサブルーチンの呼び出しなど重要な概念を含んでいたが，そのシステムは複雑な問題を解けるほどの実用性はなかった．

一方人工知能の各分野，音声認識や視覚認識，自然言語処理，定理の自動証明などの研究が進むにつれ，もっと強力なシステムあるいはプログラミング言語が要求され，Stanford 大学の SAIL, SPI††† の QA 4, STRIPS, Edinburgh 大学の POP. 2 などがつくられた．

1968 年 MIT の C. Hewitt によって提案された PLANNER[1),2)]も同様の目的を持ったプログラミング言語であるが，そのデータ構造や制御構造は，従来の言語になかった新しい機能をいくつか取り入れており，問題の解法の記述や定理の証明が容易におこなえるよう工夫されている．現在 PLANNER は Project MAC の MULTICS の中に 2 年計画で作成中であるが[7)]，あまりにも機能が大きすぎるため十分実用的なものができるかどうか疑問のあるところである．

ここで紹介する MICRO-PLANNER[5),6)]は MIT の T. Winograd が自然言語を理解するシステムに利用する目的で，PLANNER の一部の機能を抜き出し，G. J. Sussman 等と共同でつくりあげたものである．

MICRO-PLANNER はその構造が LISP に似ており，実際のシステムも LISP で書かれているので，その理解のためには LISP[15),16)]の知識が必要である．

† On a new language MICRO-PLANNER for proving theorems in Artificial Intelligence

†† 電子技術総合研究所　電子計算機部　計算機方式研究室

††† Stanford Research Institute

2. 基本動作

MICRO-PLANNER を理解する一番良い方法は，例を示してその動き方を見ることなので，まず簡単な例をあげて，基本的な機能を説明する．

「ADAM は人間である．」　(1)

「すべての人間は誤まりやすい．」　(2)

故に「ADAM は誤まりやすい．」　(3)

という三段論法による証明を考える．

これを MICRO-PLANNER で表現すると(1)(2)は

```
(THASSERT (ADAM HUMAN))                    (1′)
(DEFPROP THEOREM 1
  (THCONS (Y) (?Y FALLIBLE)                (2′)
    (THGOAL (?Y HUMAN))))
(THASSERT THOEOREM 1)
```

となる．

証明は MICRO-PLANNER に

```
(THGOAL (ADAM FALLIBLE)
        (THTBF THTRUE))                    (3′)
```

を評価 (evaluate) させればよい．各ステートメントは評価されると side-effect をおこし，値 (value) として「成功」(success) か「失敗」(failure) をとる．

(1′)式の THASSERT は評価されると，その引数を assertion データ・ベースに登録し成功値をとる．

この例では (ADAM HUMAN) が登録される．つまり，assertion データ・ベースは「AはBである．」とか「AはBの上にある．」といった断定的な「事実」を登録するところである．

(2′) 式は THEOREM 1 という名前の consequent 形の「定理」を定義してtheorem データ・ベースに登録する．この定理の意味は，「ある変数YがFALLIBLEである．」というゴールを達成したいなら，まず「その変数Yが HUMAN である．」というゴールを達成せ

よ，ということを表わしている．?Y の ? は，Yが変数であることを表わす．また MICRO-PLANNER の関数は，すべて TH ではじまるが，これは MICRO-PLANNER が LISP で書かれているため，LISP の関数と区別するためである．定理の形式は，まず名前とタイプを書き，次にパターンを書く．この例では (?Y FALLIBLE) がパターンにあたり，このパターンにマッチするかどうかでこの定理を使うかどうかが決定される．その次に定理の内容を書く．(1′) 式と (2′) 式から，データ・ベースに登録されるものは「事実」(特定の事に関する知識) と「定理」(手続型の知識，の2種類があることがわかる．

さて (3′) 式の THGOAL は「ADAM は FALLIBLE である．」という主張が正しいかどうか証明するための関数である．(3′) 式が評価されると，まず (ADAM FALLIBLE) という事実がデータ・ベースに登録されているかどうか調べる．もし登録されていれば，直ちに成功して評価を終る．この場合登録されていないので，次の (THTBF THTRUE) を見る．これは該当する事実がなければ (ADAM FALLIBE) というパターンにマッチする定理を探して実行せよという意味である．そこで theorem データ・ベースを探すと，THEOREM 1 の (?Y FALLIBLE) がマッチする (変数はすべての atom とマッチする．) ので，?Y の値を ADAM として THEOREM 1 の中を評価する．そうすると

(THGOAL (ADAM HUMAN))

というサブゴールがつくられ (ADAM HUMAN) という事実が assertion データ・ベースに登録されているか探しに行く．この事実は登録されているので，このゴールは成功し，その結果 (3′) 式も成功である，ということになる．

MICRO-PLANNER の制御構造の特徴は automatic backtracking である．先の例では backtrack を必要としなかった．そこで，もう少し複雑な例を考えてみよう．先の例のデータ・ベースに更に次の事実を登録する．

(THASSERT (EVE HUMBN))
(THASSERT (EVE SINNER))

従って，事実は

(ADAM HUMAN)
(EVE HUMAN)
(EVE SINNER)

であり，定理は THEOREM 1 がある．ここで

「誤まりやすい人で罪を犯したものはいるか？」

という質問をする．これは論理学で existential quantifier を含む問題である．MICRO-PLANNER で表わすと

(THPROG (X)
(THGOAL (?X FALLIBLE) (THTBF THTRUE))
(THGOAL (?X SINNER)))

となる．THPROG は LISP の PROG と同じ働きをする関数で，これが existential quantifier を導入する．

このプログラムを評価すると，最初の THGOAL は先の例と同じように働き，変数 X の値は ADAM になり，このゴールは達成される．THPROG の中では評価は上から順に行なわれ，各 THGOAL は and の関係になる．

そこで評価は2番目の THGOAL にうつるが，このとき ?X は既に値 ADAM を持っているので，達成すべきゴールは

(THGOAL (ADAM SINNER))

になる．(ADAM SINNER) という事実は登録されていないので，このゴールは失敗する．失敗が起きると MICRO-PLANNER は自動的に backtrack をはじめ，一番近くの別の選択が可能なところまで戻る．この場合は THEOREM 1 の中の (THGOAL (?Y HUMAN)) で ?Y の値として ADAM を選択した所に戻り，この値を捨てて別の可能性を探す．すると (EVE HUMAN) という事実が見つかり ?Y は EVE にマッチし，この値が THPROG の ?X に伝わり，THPROG 内の最初のゴールは (EVE FALLIBLE) で成功し，再び2番目の THGOAL を評価する．今度は

(THGOAL (EVE SINNER))

となるが，これは事実として，データ・ベースに登録されているので成功し，その結果 THPROG も成功して ?X の値は EVE が返ってくる．(図1)

このように MICRO-PLANNER はステートメント

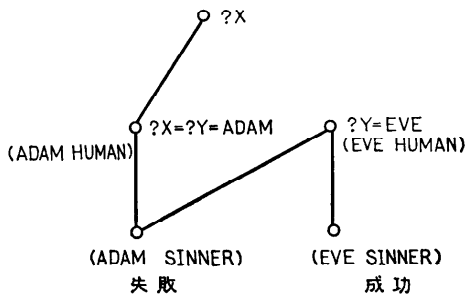

図1 MICRO-RLANNER の backtracking

を評価して失敗すると，もっとも近い選択点へ自動的に戻り，そこで新しい選択を行なう．新しい選択ができなくなったら，更にもう一つの前の選択点へ自動的に戻る．戻る途中に side-effect を持った関数があると，その effect をすべて元通りにする．たとえば THASSERT の side-effect は，データ・ベースに登録すること，なので後戻りのときは，登録したものを取り消してしまう．以上で MICRO-PLANNER の主な機能が出たのでこれを少しまとめてみよう．

3. MICRO-PLANNER の特徴

3.1 goal-oriented な問題の記述

問題を解こうとするとき，まず最終ゴールを定め，そのゴールを達成するための必要条件となるサブゴールをたて，更にその必要条件となるサブゴールをたてるという方法は，人間の思考過程として自然である．

MICRO-PLANNFR はこのような思考過程を THGOAL を用いることによってそのまま表現することができる．

また THGOAL の中のパターンは任意の長さが許されるので，普通の言葉に近い表現をすることができる．

このため複雑な問題の記述も容易になる．

また MICRO-PLANNER には「成功」と「失敗」の概念がある．MICRO-PLANNER は LISP と同じように，各ステートメントを評価し，side-effect を起こすと同時に「値」を持つ．この値が成功か失敗である．通常の言語では「失敗」というとアルゴリズムをまちがえたことになるが，MICRO-PLANNER はそうではない．MICRO-PLANNER の場合，失敗は automatic backtracking を引き起こすことになり，現在の選択を放棄して新しい選択を探す試行錯誤を開始する．

3.2 automatic backtracking

backtrack のしかたは先の例で述べたように，ステートメントを評価し，値が失敗の時，自動的におこる．この際，システムは各ステートメントの side-effect をすべて元に戻しながら backtrack する．この方法は理論的には Floyd の nondeterministic algorithm[14] として知られているが，それを完全な形で言語の中に取り入れたのはおそらく MICRO-PLANNER がはじめてであろう．この方法では，システムはプログラムを評価しながら前進していくときに backtrack のための準備をしておかなければならない．たとえば，「現在の値が1である変数Aの値を2にせよ.」を評価すると，backtrack のための制御用スタックに「Aを2から1にせよ.」と書いておかねばならない．もしこの部分が backtrack しなければ，この準備は全く無駄になるわけで，それだけ余分の時間を消費したことになる．また backtrack の準備は実行時にしか決められないことが多く，従ってコンパイルはほとんど不可能であり，この点でも実行時間を長びかせる．このように automatic backtracking は実用的な面で不利な点が多く，現在 MICRO-PLANNER に対する不満もこの点に集中している．この点については，また後でふれることにし，automatic backtracking の良い点を挙げておこう．それは利用者にとって，プログラムを書く手続が簡単になるということである．探索し，試行錯誤を行なわねばならない問題では，必ず backtracking が必要である．その方法は automatic backtracking 以外にもあるわけだが，この方法が概念として単純であり 利用者も楽である．しかし，その分だけシステム側に負担がかかっている．別の方法（後述）では，システムの負担は軽くなっているが，利用者の負担は増している．

3.3 データ・ベース

3.3.1 連想機能

データ・ベースは2種類のデータを登録することができる．一つは，特定の事柄に関する事実であり，もう一つは，一般的な手続を記述した定理である．

そして，これらのデータは，その一部を指定すると，連想機能によって全体が取り出せる．もし，

(ADAM LIKES POETRY BUT DOESN'T LIKE EVE)

という事実が登録されていると，ADAM, LIKES, POETRY, BUT, …のどれか1つがわかっていれば，このデータが引き出せる．

3.3.2 命令形の定理

他の証明用言語でも定理を持っているものはある．しかし，それらは普通，平叙文「AならばBである.」という形をしている．MICRO-PLANNER の場合は定理は命令形で，

（a）「Bを証明しようとするゴールがあるならば，Aをサブゴールとして設定せよ.」

（6）「Aが登録されたなら，Bを登録せよ.」

（C）「Aが取り消されたなら，Bを取り消せ.」

の3つの形がある．（a）は先の例にでてきた consequent 形の定理である．（b）（c）はそれぞれ，antecedent 形，erasing 形と呼ばれ，データ・ベースの整理

をするために用いる．たとえば先の例で THEOREM 1を持つかわりに (ADAM HUMAN), (EVE HUMAN) という事実が与えられたら，必ず (ADAM FALLIBLE), (EVE FALLIBLE) という事実も登録すれば，THEOREM 1 は必要でなくなる．論理的には問題はないが，これを乱用すると，たちまちデータ・ベースは一杯になるし，つまらない事実をたくさん登録することになる．どちらの形を用いるかは，利用者にまかされている．

3.3.3 状態

公理は時間的に不変であるが，事実は時間とともに変化する．そこで他の theorem-prover では(A ON B) というデータを受け入れることはできない．なぜなら「AがBの上にある．」というのは公理ではなく，ある状態における事実でしかないからである．

そこで普通は，「状態」という変数を導入して，(A ON B S) という形で表わす．また定理を用いるときも状態を考慮せねばならない．「Yの上にあるXをZの上に置けば，XはZの上にあり，YとZが同じでなければXはYの上にない．」という定理を考えると

```
(FOR-ALL X Y Z S)
(AND (X ON Y S)
  (OR (EQUAL Y Z)
    (NOT (X ON Y (PUT X Z S))))))
```

と表わされるであろう．PUT というのは，状態Sのときに，XをZの上に置くことによって得られる状態を値とする関数である．(X ON Y (PUT X Z S)) はXをZの上に置いたあと，XはYの上にあるかをきいてみるわけだが，このためには今までの行動の中でXにかかわったものを，すべて思い出して，それをYの上に置いたかどうか調べなければならない．これは普通長い手続を必要とする．MICRO-PLANNERでは，状態という変数を導入せず，かわりにデータ・ベースが現在の状態を表わすようにする．上の定理は

```
(DEFPROP THEOREM 2
  (THCONSE (X Y Z)
    (PUT ?X ?Z)
      (THGOAL (?X ON ?Y))
      (THERASE (?X ON ?Y))
      (THASSEST (?X ON ?Z)))
```

と表わされる．この意味は「XをZの上に置く．」ときは，Xが上にあるYを探して「XはYの上にある．」という事実を取り消して，「XはZの上にある．」という事実をつけ加える，ということである．このようにMICRO-PLANNER は状態について，一々述べる必要を取り除き，世界の移りかわる状態をデータ・ベースの移りかわる状態に反映させることができる．

3.4 パターンによるデータおよびサブルーチンの呼び出し

MICRO-PLANNER のデータ・ベースは，問題がかかわる世界を表わしており，問題が複雑になればその量は膨大なものになる．一方制御構造は automatic backtrackingを持っており，ある選択の結果が失敗したとき，自動的にデータ・ベースから別の選択を引き出さねばならない．この役割を効果的に行なうのがパターン・マッチングである．普通のプログラミング言語では，データの呼び出しは場所を指定して行なうし，サブルーチンの呼び出しは名前で行なう．

MICRO-PLANNER の場合は，これらはすべてパターン・マッチングで行なわれる．

(THGOAL (?Y HUMAN)) が (ADAM HUMAN) を引き出したのも，(THGOAL (ADAM FALLIBLE) (THTBF THTRUE)) が THEOREM 1を引き出したのも，すべてパターンによって自動的に行なわれた．MICRO-PLANNER のプログラムは，データ・ベースに蓄えた事実と定理（サブルーチン）を適当にパターンで選びわける．これらの操作は，利用者には表向きは見えない．逆に言えば，利用者は，わずらわしい解の探索をシステムまかせにすることができる．

また MICRO-PLANNER は定理を名前で呼び出すこともできる．これは recommendation と呼ばれているもので，たとえば(3′)式で

```
(THGOAL (ADAM FALLIBLE) (THUSE
  THEOREM 1))
```

と書くと，定理を探しに行くとき名前がわかっているので直ちに呼び出すことができる．もし利用者が，使うべき定理がわかっているなら，それをパターンで呼び出すのは能率が悪くなる．何故ならパターンによる呼び出しは，手続が名前による呼び出しより複雑であるし，THEOREM 1 以外の定理をたくさん呼び出すかもしれない．それでも結局は THEOREM 1 に到達するわけだが，それまでに無駄な時間を消費する．

パターンによる事実と定理の呼び出しと，名前による定理の呼び出しを使いわけて，大規模な問題を能率よく処理していくことが可能になる．

4. 主な関数

MICRO-PLANNERの関数は全部で約 40 ある．そ

の中から良く使われるものを抜き出して説明しょう.

4.1 データ・ベースを操作する関数

(1) (THGOAL pattern recommendations…)

データ・ベースから pattern にマッチする事実を取り出す. もし事実が見つかったら, Pattern の中の変数に値を結合し, 成功値をとる.

recommendation には次のものが書ける.

(THNODB) 事実を探すな.

(THDBF filter) filter を満足するような事実のみを探せ.

(THTBF filter) filter を満足するような consequent 形の定理のみを探せ. すべての定理とパターン・マッチを試みるなら (THTBF THTRUE) と書く.

(THUSE theorem-names) 指定した名前の consequent 形の定理を用いよ.

もし事実を見つけることができなくて, recommendation に (THTBF filter) や THUSE theorem-names) が与えられていれば, 条件を満足する consequent 形の定理を探し, あれば評価する. THGOAL の値は定理の値になる. 事実も定理も見つからないとき THGOAL は失敗する.

(2) (THFIND mode macro variable-list steps …)

steps の部分に条件を表わすプログラムを書くと, それを満たす答の数に制限をつけることができる. THGOAL は答を1つ見つける depth-first search であるが, THFIND は breadth-first search を可能にする. たとえば,「赤いブロックをすべて見つけよ.」は

```
(THFIND ALL ?X (X)
    (THGOAL (BLOCK ?X))
    (THGOAL (COLOR ?X RED)))
```

と書ける, mode は ALL の他に (min max result) という形式が使え, 見つけた答の数が「少なくとも min」で「多くとも max」の範囲にあるとき, THFIND の値として result を返す.

(3) (THASSERT pattern recommendations…)

データ・ベースに事実または定理を登録する. recommendation は次の通り.

(THPSEUDO) 事実をデータ・ベースに登録しないで, 値は成功となる.

(THPROP expression) 事実に特性をつけて登録する.

(THTBF filter) filter を満足する antecedent 形の定理をすべて評価する. この定理の値は THASSERT の値に影響しない.

(THUSE theorem-names) 指定した名前の antecedent 形定理を用いよ.

(4) (THERASE pattern recommendations…)

登録した事実や定理を取り消す. recommendation は THASSERT と同じだが, 関係する定理は erasing 形である.

4.2 THPROG とその中で用いる関数

(1) (THPROG variable-list steps…)

LISP の PROG と同様で, ステートメントを上から順に評価するための関数である. 評価の値が失敗の時は backtrack して side-effect も元に戻される.

(2) (THSUCCEED node expressions)

指定した場所あるいは関数や定理を強制的に成功させる. 次の形式がある.

(THSUCCEED THTAG tag) 成功値を持って指定した tag の所へ行け.

(THSUCCEED THPROG expressions) expressions を評価した値を THPROG の値とする.

(THSUCCEED THEOREM expressions) expressions を評価した値を THEOREM の値とする.

(THSUCCEED THEOREM) THEOREMを強制的に成功させる.

(THSUCCEED) 成功値をとる.

(3) (THFAIL node expressions)

指定した場所または関数や定理を強制的に失敗させる. 形式は THSUCCEED と同じものの他に (THFAIL THMESSAGE message) がある. これは評価されると強制的に失敗して THMESSAGE ステートメントのパターンにマッチするところまで backtrack する.

(4) (THMESSAGE variable-list pattern steps …)

上の (THFAIL THMESSAGE message) と組み合わせて用い, backtracking の及ぶ範囲を制御する. THFAIL の message を評価した値と THMESSAGE のパターンが一致すると steps を評価する.

(5) (THUNIQUE variables)

プログラムが無限ループに入ることを防ぐ関数である．実際の働きは変数に同じ値が結合されると失敗し，その他の時は成功する．

(6) (THFINALIZE node) or (THFINALIZE THTAG tag)

プログラム内の指定した部分を backtrack 制御用スタックから取り除く．従ってこの部分は backtrack できなくなる．

この他に THGO, THRETURN があるがbacktrack する以外は LISP の GO, RETURN と同じである．

4.3 プール関数

THCOND, THOR, THAND, THNOT などがあるが，いずれも同名の LISP の関数と同様の機能を持つ．

MICRO-PLANNER に特有のものをあげておく．

(THAMONG variable choice-list)

変数に値を結合するための関数である．もし変数が値を持っていないなら，choice-list の先頭の値を結合する．既に値を持っているときは，その値が choice-list の中になければ失敗する．もし backtrack してきたら，変数に choice-list の次の値を結合して成功する．choice-list が尽きたら失敗する．

これで MICRO-PLANNER の説明を終り，少し複雑な例を示そう．

5. 例題

「3人の宣教師と3人の人喰い土人.」

問題は“3人の宣教師と3人の土人が，1せきのボートを使って川を渡ろうとしているが，ボートの定員は2名なので何回かに分けて運ばねばならない．その途中のどんな時でも，土人の数が宣教師の数を越えると土人は宣教師を食べてしまう．食べられないように渡るにはどうすればよいか．ボートは全員が漕げ，1人で操作することができる.”というものである．これを MICRO-PLANNER で書いた1つの例が図2である．

このプログラムの方針は，最初全員が左岸におり，ボートも左岸にあるという状態を (STATE 宣教師の数 土人の数 ボートのある岸) というふうに表わしている．(ステートメント 0080) 次にボートの可能な乗り方も登録しておく．(0110～0150)

定理 LEFT-BOAT の方針は

```
0010*      THREE MISSIONARIES AND THREE CANIVALS
0020
0030* PLANNER VI.
0040
0050* ASSERTIONS
0060
0070*  INITIAL STATE
0080 (THASSERT (STATE 3 3 1))
0090
0100* OPTIONS
0110 (THASSERT (BOAT 0 1))
0120 (THASSERT (BOAT 1 0))
0130 (THASSERT (BOAT 1 1))
0140 (THASSERT (BOAT 2 0))
0150 (THASSERT (BOAT 0 2))
0160
0170* THEOREMS
0180
0190 (THCONSE LEFT-BOAT (X1 Y1 X2 Y2) (STATE (THV X1) (THV Y1) 0)
0200*  CHECK LOOP
0210 (THUNIQUE X1 Y1)
0220
0230*  SELECT BOAT
0240 (THGOAL (BOAT (THV X) (THV Y)))
0250
0260*  CHECK (MISSIONARIES > 0 = CANIVALS)
0270*        (MISSIONARIES = 0)
0280*        (CANIVALS = 0)
0290 (THEITH (THV X) (ALL (THV X1) (THV Y1)))
0300 (THEITH (THV Y) (ALL (THV Y1) (THV Y)))
0310 (THANY (GREATER X (THV X))
0320         (GREATER X (THV Y)))
0330         (THOR (EQUAL (THV X) 0)
0340               (EQUAL (THV X) 3)
0350               (EQUAL (THV X) (THV Y)))
0360
0370*  CALL RIGHT-BOAT
0380         (THGOAL (STATE (THV X) (THV Y) 1) (THTBF THTRUE)))
0390 )
0400
0410 (THCONSE RIGHT-BOAT (X1 Y1 X2 Y2) (STATE (THV X1) (THV Y1) 1)
0420*  CHECK LOOP
0430 (THUNIQUE X1 Y1)
0440
0450*  SELECT BOAT
0460 (THGOAL (BOAT (THV X) (THV Y)))
0470
0480*  CHECK CONDITIONS
0490 (THSETQ (THV X) (DIFFERENCE (THV X1) (THV X)))
0500 (THSETQ (THV Y) (DIFFERENCE (THV Y1) (THV Y)))
0510 (THANY (GREATER (THV X) -1)
0520         (GREATER (THV Y) -1)
0530         (THOR (EQUAL (THV X) 0)
0540               (EQUAL (THV X) 3)
0550               (EQUAL (THV X) (THV Y)))
0560
0570*  CALL LEFT-BOAT
0580         (THGOAL (STATE (THV X) (THV Y) 0) (THTBF THTRUE)))
0590)
0600
0610 (THGOAL (STATE 0 0 0) (THTBF THTRUE))
```

図2 3人の宣教師と3人の人喰い土人のプログラム

1. THUNIQUE で同じ STATE が現われるのを禁止してループを防ぐ．(0210)
2. THGOAL でボートの乗り方を決める．(0240)
3. 左岸，右岸，ボートのどこでも土人の数が宣教師の数を越えないことをチェックする．(0260～0350)

   ここでもし条件を満たしていなければ，失敗して2の THGOAL へ backtrack する．満足していれば
4. 反対側の岸のボートの乗り方を決めるため定理 RIGHT-BOAT を呼ぶ．

RIGHT-BOAT は最後に LEFT-BOAT を呼ぶところが LEFT-BOAT と異なるだけである．

最後の THGOAL (0610) が達成すべきゴールで，左岸の STATE が宣教師も土人も0になり，ボートも左岸にない，ということを意味している．この THGOAL によってまず定理 LEFT-BOAT が呼ばれ，次いで RIGHT-BOAT, LEFT-BOAT, …というように繰り返し定理を呼び，途中失敗を繰り返しながら解に達する．(図3)

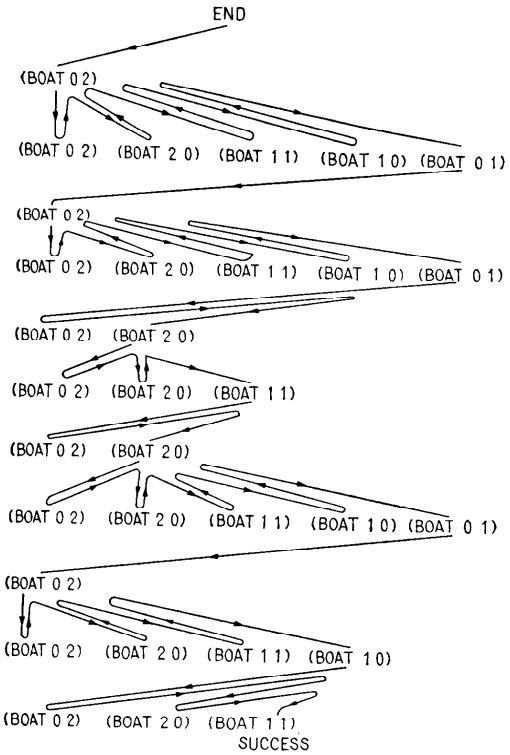

図 3 解の探索の様子

なお図2のプログラムで (THV X) とあるのは ?X と同じものである．また定理の定義と登録の形式が本文と少し異なり簡単になっている．†

6. 問題点——PLANNER から CONNIVER へ

高級言語は利用者にかわってシステムが多くの処理をするため，問題の記述が容易になるが，同時に言語の持つ個有の構造を利用者に押しつける．ある定まったデータ構造と制御構造を持つことは，利用者にそのスタイルに従って問題を解くことを強制する．

従って言語が，対象とした分野で一般に用いられているアルゴリズムの構造を，そのまま記述できないときに問題が生じる．MICRO-PLANNER の持ついくつかの特徴は，実際の人工知能の分野でどれほど有用であろうか？この点については既に MIT でかなりの使用経験が積まれており，その報告も幾つかでている[8),9)]．

† 本文は STANFORD 大学の MICROPRANNER[6)] をもとにしたがこの例は電子技術総合研究所のものを使ったため若干の相異が生じた．

その中で指摘されている最大の問題点は automatic backtracking である．この機能は，一見具合が良いように見えるが，実際にはそうではなかった．

その理由は

(1) automatic backtracking は，その準備のためにかなりの時間とメモリーを必要とする．もし失敗が起こらなければ，それらの準備は全く無駄になる．ところが現実のプログラムでは対象とする問題の知識がかなりあるので，失敗する回数が少なく，準備にかける手間が引き合わない．Sussman は「失敗をしばしば引き起こすようなプログラムは悪いアルゴリズムであり，それをシステムが救済するのは良くない．」と主張している．

(2) backtracking を引き起こす大部分のプログラムは，もっと効率の良い recursive なプログラムに置きかえることができる．

(3) 解の探索のための方法が，depth-first search に限られる．良いプログラムは事前に depth-first search か breadth-first search かを決めないで，探索の状況に応じて両者を使いわけるものである．

(4) automatic backtracking は失敗したら，単に side-effect を元に戻して，一番近い選択点へ戻る．失敗の理由によっては，もっと前の選択点へ戻る方が良い場合がある．このため (THFAIL THMESSAGE message) があるが，十分でない．

である．

これらの欠点は MICRO-PLANNER の設計思想にかかわるもので，簡単な改良で克服されるものではない．

そこで制作者の1人 G. J. Sussman は階層的なデータ構造と，より柔軟な制御構造を持った CONNIVER[10),11),12)] という言語を考え出した．CONNIVER は大部分の基本的考えを PLANNER から受け継いでいるが，制御構造に automatic backtracking を採用せず，階層的なデータ構造に基づく多重プロセスの機能を取り入れている．この方法によれば上に述べた欠点はすべて解消するがプログラミングは少し難しくなるであろう．大切な事は人工知能の問題の構造をどちらが正しく把握しているかである．CONNIVER については MIT でもまだ使用経験が乏しく評価は定まっていないが，やがては MICRO-PLANNER に取ってか

わるものと思われる.

なお MICRO-PLANNER のマッチングの能力，すなわち変数が atom としかマッチできないことも欠点の一つであるが，この拡張は容易である.

7. 結び

定理証明の分野では従来 predicate calculus が最も良く使われていた．この方法はしっかりした数学的背景を持つが，現実の複雑な問題を解くのは非常に困難である[17]．MICRO-PLANNER はプログラミング言語であるから数学的な正しさとは無関係であるが，推論過程を記述するための様々な便利な機能を持っており，実用規模の問題を解く定理証明システム実現への足がかりを与えてくれたと言えよう．こうした推論過程を記述するための言語は他に Stanford 大学の new SAIL や SRI の QA 4 が著名であり，これらは CONNIVER と同等あるいはそれ以上の機能を有している．こうした言語が日本でも広く普及し，人工知能の各分野に良い影響を与えることを期待したい.

参考文献

MICRO-PLANNER 及び PLANNER の言語使用に関するもの

1) C. Hewitt: PLANNER: A Language for Manipulating Models and Proving Theorems in a Robot, MIT AI Memo No. 168. (1968, Revised 1970).
2) C. Hewitt : PLANNER: A Language for Proving Theorems in Robot, Proc. of IJCAI (International Joint Conf. on Artificia lIntelligence) (1969).
3) C. Hewitt: Description and Theoretical Analysis (Using Schemata)of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot, MIT Technical Report AI TR-258 (1972).
4) C. Hewitt: Procedual Embedding of Knowldge in RLANNER, Proc. of IJCAI(1972).
5) G. J. Sussman, T. Winograd & E. Charniak: MICRO-PLANNER Reference Manual, MIT AI Memo No. 203A (1971).
6) B. G. Baumgart: MICRO-PLANNER Alternate Reference Manual, Stanford Univ. SAILON No. 67 (1972).

MICRO-PLANNER および PLANNER の応用に関するもの

7) C. Hewitt: PLANNER Implementation Proposal to ARPA 1972-1973, MIT AI Memo No. 250 (1971).
8) T. Winograd: Procedures as a Representation for Data in a Computer Program for Understanding Natural Language, MIT MAC TR-84 (1971).
9) P. Winston: The MIT Robot, Machine Intelligence, Vol. 7 (1972).

CONNIVER に関するもの

10) G. J. Sussman: Why Conniving is Better than Planning?, MIT AI Memo No. 255 (1972).
11) D. V. McDermott & G. J. Sussman: The CONNIVER Refernce Manual, MIT Al Memo No. 259(1972).
12) G. J. Sussman & D. V. McDermott: From PLANNER to CONNIVER ……A genetic approach, Proc. FJCC, pp. 1171-1179(1972).

その他

13) G. W. Ernst & A. Newell: GPS: A Case Study in Generality and Problem Solving, Academic Press (1969).
14) R. W. Floyd: Nondeterministic Algorithms, JACM, Vol. 14, No. 4, pp. 636-644 (1967).
15) J. McCarthy, et al.: LISP 1.5 Programmer's Manual, The MIT Press (1962).
16) C. Weissman: LISP 1.5 Primer, Dickenson Publishing Co. (1967).〔ダイヤモンド社より邦訳が出ている.〕
17) J. Derksen, J. Rulifson & R. Waldinger: The QA 4 Language Applied to Robot Planning, Proc. FJCC, pp. 1181-1192 (1972).

(昭和48年3月3日受付)