人工知能と対話技法 17－5
（1980. 9. 17）

PROLOGによるゲームプログラムの構成

田島守彦
（電子技術総合研究所）

0．あらまし

プログラム言語PROLOGによってゲームプレイプログラムOTL.PLGを開発中である。柔軟性を持たせ学習が可能なよう，知識および知識ベースを改良するためのメタ知識を節形式で表現し実行することができる。序盤について実例を示す。

1．はじめに

ゲームは人工知能研究のテーマとしては古くから研究されてきており，最近では人間に劣らぬ能力を持つゲームプログラムも出現している［1］。しかしながら，チェス，碁等の完全情報ゲームにおいては，その複雑さのため人智に匹敵するものはまだ実現していない。のみならず，その思考方法も，mini-max手続きを主な手段とするカづくの場合がほとんどであり，人工知能と称するには単純すぎると思われるものが多い［2］。

筆者もOthelloゲームを例にとってゲームプログラムの研究を行なってきたが［3］，従来からの言語で容易にプログラムができるような形のプログラムでは，余りにも構造が固定化し，人間が行なっているような柔軟かつ自由で発展性のある思考は実現困難であることを経験してきた。特に，場面場面に応じて戦略を選び，必要な枝のみを集中的に検討し，敵の意図をも考慮する，といった柔軟な思考は，知識ベースを扱えるような言語によって初めて実現できると考えられる。また，知識ベースを自から組織化してゆくことを考えても同様である。

一方人工知能向き言語の分野では，最近，述語論理形言語であるPROLOGが注目されている。PROLOGでは，述語論理の証明過程がプログラム実行に対応しており，形式的に明解で美しくかつ強力である。すでにいくつかの応用が試みられている［4］。論理は節形式で表わされ，基本的にデータとアルゴリズムの区別が無く，各節が知識片としての機能を持っている。また自身でデータベースを変更できる機能を持っているため，メタ言語としての記述が同時に可能である[5]。これらは学習機能を表現する上で非常に強力である。また，必要な節を必要な場合に作り出すことができ，固定したプログラム全体を予め用意することなく，柔軟なシステムを作成できることが期待できる。加えて，PROLOGに備わっている自動後戻り（backtrack）の機能は，ゲームのような木探索を必要とするシステムの記述に好都合である。

これらの理由により，本論文では，ゲームプログラムをPROLOGで記述することを試みる。ゲームの例としては引続きOthelloを用いる。初めに全体の構成について述べる。次に局面および序盤の戦略，定石等の表現法について記し，学習メカニズムの記述例および実行例を示す。最後にPROLOGによるプログラミング上の問題点あるいは強化すべき機能について考察する。

2．PROLOG概説

2.1 記法 [C1,C7]

1階述語論理を節形式（clause form）で表現する。節とは次の形のものである。

$$Q_1 \vee Q_2 \vee \cdots \vee Q_m \leftarrow P_1 \wedge P_2 \wedge \cdots \wedge P_n$$

ここで $m \leq 1$ の場合，節はHorn節と称される。PROLOGでは節をHorn節に限る。よって次の3タイプの節が存在する。

(1) $Q \leftarrow P_1 \wedge P_2 \wedge \cdots \wedge P_n$

(2) Q

(3) $\leftarrow P_1 \wedge P_2 \wedge \cdots \wedge P_n$ （$\neg(P_1 \wedge P_2 \wedge \cdots \wedge P_n)$ を意味する）

これらをPROLOGではそれぞれ次のように表記する。

(1) $+Q-P_1-P_2-\cdots\cdots-P_n.$

(2) $+Q.$

(3) $-P_1-P_2-\cdots\cdots-P_n.$

ここでQを節の頭（head）と呼び，残りを節の本体（body）と呼ぶ。Skolem関数として許されるデータ型はリストのみである。リストは次のようにドットで示される。

<リストの例>　　A.B.C.NIL

2.2 動作

PROLOGの処理系の目的は，Horn節集合に対して分解証明（resolution）を行なうことである。(1)および(2)の形式の節の集合をプログラムとし，(3)の形式の節をそれに対する実行文とする。証明法は入力分解証明（input resolution）であり，同一頭の節の入力は上から下，本体内の述語統合は左から右の順である。木探索的には深さ優先の展開である。統合（unification）が失敗すると自動的に後戻りを行なう。

問題解決（problem solving）の立場からは次のように解釈される。すなわち，目的を否定したものを(3)の形式で表わし，(1)の形の演算，(2)の形の言明（assertion）を用いて反証（refutation）を行なう。反証の過程で得られる反証例が解となる。

ゲームプログラムでは，反証例として打着手を得る。その際最適手が最初に例証されるようにしておく。ゲーム終了後の学習過程では反証例として次のゲームのための新たな公理系を得るのが目的となる。両者ともPROLOGの証明過程に帰される。後者の証明のための節集合は前者の証明のためのメタ知識である。

2.3 機能 [C1,C7]

使用者の便宜のために次のような機能がある。

表 2-1　組込述語と制御リテラル

-LU(*C)	1文字読込み	-BLANC(*C)	空白
-LUB(*C)	〃	-ETOILE(*C)	*
-ECRIT(*C)	1文字書出し	-VIRG(*C)	,
-LIGNE	1行印字	-PARG(*C)	(
-TTY	TTY↔ファイル	-PARD(*C)	)
-LIREFICHIER	節→データベース	-GUILLEMET(*C)	"
-BOOLISTE	スイッチON⇔OFF		
-IMPRIME	入力流れの現行出力	-AJOUT(*C)	節→データベース
-SORCHA(*S)	文字列書出し	-AJOUTB(*C)	〃〃(終りに)
-SORTER(*X)	項の書出し	-AJOUTC(*C)	同上(重ね書きはなし)
-SAUVE	現状態→ファイル	-SUPP(*C)	データベースから除去
-STOP	実行終了	-*X	項がアトムと解釈
		-UNIV(*X,*Y)	項*Xの分解と*Y
-PLUS(*N1,*N2,*N3)	+	-ATOME(*X,*Y)	〃
-MOINS(*N1,*N2,*N3)	−	-FGALF(*X,*Y)	構文の一致
-MULT(*N1,*N2,*N3)	×		
-DIV(*N1,*N2,*N3)	÷	-/	打切り
-RESTE(*N1,*N2,*N3)	余り	-/(*L)	打切り*Lを失敗へ
-INF(*X,*Y)	<	-ANCETRE(*L)	最新の祖先を*L
-LETTER(*C)	アルファベット	-ETAT(*S)	現状態を*Sに
-CHIFFRE(*C)	数字		

(1) 組込述語　算術，入出力に関する標準的な述語に加え，データベース操作のための述語が用意されている。
- データベースへの付加　-AJOUT, -AJOUTB, -AJOUTC
- データベースからの削除　-SUPP
- 文字列とリスト間の変換　-UNIV, -ATOME

(2) 制御　PROLOG では統合が失敗すると自動的に後戻りが行なわれるが，プログラムによっては全ての可能性について試みる必要がない，あるいは行なってはいけない場合がある。証明の打切りのために制御用リテラルがある。
- -/, -/(項), -ANCETRE

表 2-1 に組込述語および制御用リテラルを要約する。

3. ゲームプログラムの構成

Othello ゲームプログラムの全体構成図を図 3-1 に示す。PROLOG では述語の全てのデータが直接にパラメータとして渡される。各節が procedure とみなせる。実行が TOP-DOWN で行なわれるので，プログラミングも自然に TOP-DOWN で行うことになる。同一頭を持つ節を隣接させるという点以外に制限はないが，関連のある節は見易さのためにまとめてある。

3.1 指令レパートリ

次の 6 種の実行文を用意する。

(1) -O.　プレイプログラムの開始。必要な初期設定を行ない，先手後手を入力して停止。

(2) -M(i.j). または -M(PASS).　人間側の手入力。i, j で横縦座標を入力する。PASS ならば後の形。入力の合法性を検査し石を置き反転する。表示を行ない棋譜をとる。

(3) -C.　計算機側の着手。最適手を見出し石を置き反転し表示し棋譜をとる。

(4) -R.　結果の表示。石数および棋譜の表示。

(5) -L.　学習プログラムの実行。

(6) -P.　知識ベースの表示。

このプログラムは 1 手毎に停止するため，その都度 (2) または (3) を発行する必要がある。これを回避することも可能で，再帰的に書けばよい（図 3-2）。しかし，この方式は巨大なメモリを消費するため実用的でない。筆者のシステムでは，最も単純な，盤面のマスの全く固定された優先順位に従って打つのみ，というプログラムでさえ，50 手目で仮想メモリからあふれて実行が打切られた。

3.2 状態の記憶

プログラムは 1 手毎に停止するため，局面，

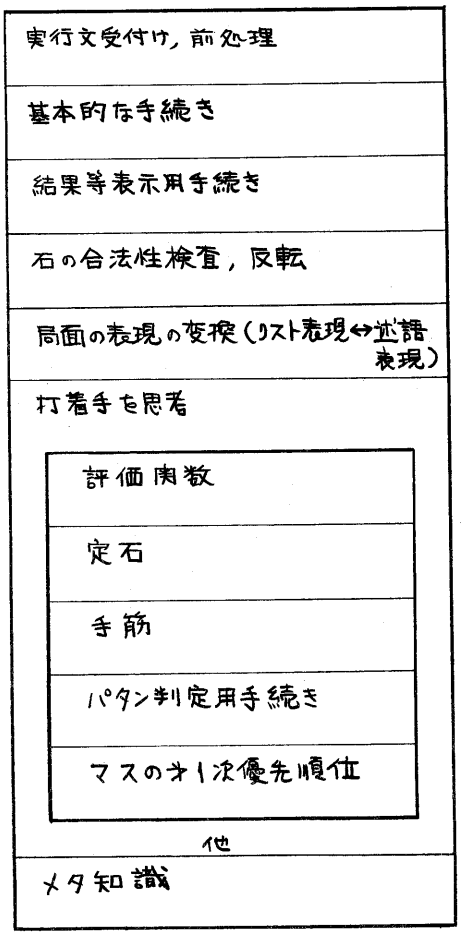


図 3-1　ゲームプログラムの構成

棋譜，手番号等の情報をデータベースを操作して保存しておく必要がある。

```
+GAME(*T,*KOM,*K,*KL,*H,*HL,*N,*NL)
  -UCH(*T,*KOM,*K,*KN,*H,*HN,*N)-PLUS(*N,1,*NN)
  -WKM(*KN,*HN,*NN)-GAME1(*T,*KOM,*KN,*KL,*HN,*HL,*NN,*NL).
+GAME1(*T,*KOM,*KN,*KN,*HN,*HN,*N,*N)-CEND(*KN,*HN,*N).
+GAME1(*T,*KOM,*KN,*KL,*HN,*HL,*N,*NL)
  -KOTAI(*T,*TN)-GAME(*TN,*KOM,*KN,*KL,*HN,*HL,*N,*NL).
```

図 3-2 再帰的に書いた場合

1手打つと最後の処理として次のものをデータベースに書込む。PROLOGでは変数であることを＊を付けて示す。

+INT(*N,*K,*H). ------ (3.1)

ここで

n：ゲームがn手進んだことを示す。
k_n：局面，n手終了した時の盤状態 k_n
l_n：棋譜

（本文中では変数を通常の表記通り小文字で示す。）

(3.1) の形の節が1手毎に1個データベースに加えられる。途中からやり直しが可能なように，前に作られた節を消去することはしない。やり直し（このための命令文は用意してある）のために前の局面 k_m に戻った時は，$n>m$ を満たすすべての +INT(*N,*K,*H). を消去するようにする。

+INT の他にも必要に応じて局面の状態を示す知識を書込んでおき，次の打着時の高速化を図っている。

3.3 使用システム

使用システムは VAX-11/780 で VAX/VMS をOSとしている。仮想メモリ4Mバイトをフルに使っている。PROLOG インタプリタが FORTRAN で書かれており，節および展開時用の記憶域はそれぞれ 250000 語，600000語（1語4バイト）である。速度はかなり速い。

4. 表現とプレイ

4.1 局面の表現

図4-1に示す盤上の情報は図4-2のようなリストで表わすのが自然である。この表現法の欠点は，リストを前からたどって行くため，各マスへのアクセスに時間がかかることである。特に書込みにはリスト全体の複製が伴うので大変である。このため必要な場合には次のような表現法を併用している。

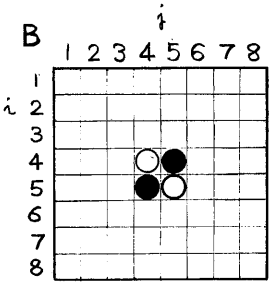


図 4-1 初期形

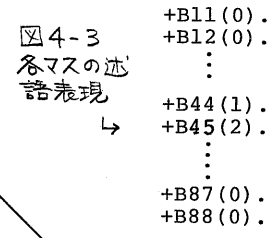


図4-3 各マスの述語表現

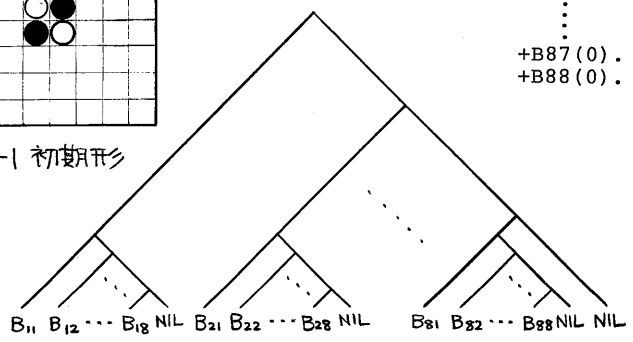


図 4-2 盤情報のリスト表現

○述語によるマスの表現

図4-3に1マスを1述語で表わす表現法を示す。マスの状態をパラメータで表わす。ここでは空白，白，黒をそれぞれ0，1，2で示している。座標 i,j から各述語を作り出すために組込述語 UNIV が利用される。データベースを扱うことはプログラムを不透明にし虫を作り易い。入念な注意が必要である。

4.2 定石の表現

Othello においても序盤における定石が知られている [9]。定石を次のような節で表現する。

+GM(*N,*K,*M)

ここで n, k, m はそれぞれ手番号，局面，次の手を示す。序盤なので，k は盤の中心部 6×6 の部分のみを表わすことにし，能率を上げている。図4-4は最初および2番目の手の指定と2個の定石を示したものである。

```
+GM(1,*K,5.6).
+GM(2,(0.0.0.0.0.0.NIL).(0.0.0.0.0.0.NIL).(0.0.1.2.0.0.NIL).
  (0.0.2.2.2.0.NIL).(0.0.0.0.0.0.NIL).(0.0.0.0.0.0.NIL).NIL,6.4).
+GM(4,(0.0.0.0.0.0.NIL).(0.*V.*W.0.0.0.NIL).(0.*X.1.2.0.0.NIL).
  (0.*Y.2.2.2.0.NIL).(0.*Z.1.0.0.0.NIL).(0.0.0.0.0.0.NIL).NIL,4.6).
+GM(6,(0.0.0.0.0.0.NIL).(0.*V.*W.0.0.0.NIL).(0.*X.1.2.0.0.NIL).
  (0.*Y.2.2.2.0.NIL).(0.*Z.1.0.0.0.NIL).(0.0.0.0.0.0.NIL).NIL,4.6).
```

図 4-4 定石の表現

4.3 序盤のプレイ

序盤は基本的戦略に定石を組合わせてプレイを行なう。戦略は次のようである。

◦戦略 序盤では自分の石が少なく，しかも団子状に固まるように打つ [9]

この目標の達成度を評価するため，次のような評価関数を近似として用いる。プレイヤを＋，－として k を局面とする。

◦関数 J 局面 k における＋側の評価点を次の式で計算する。

$$J_+(k) = N_-(k) - N_+(k)$$

ここで $N_+(k)$：局面 k における＋側の石数

$N_-(k)$：局面 k における－側の石数

但し N_+, N_- とも斜方向を除く四方を石で囲まれている石は数えない。

すると，序盤の手選択は基本的に次のようになる。ここでの議論に不要なパラメータは省略する。

手(m) ← 定石がある (m) (4.1)

手(m) ← 候補手 (m) ∧ 仮打ち (m, $k_{temp.}$) ∧ $J_+(k_{temp})$
∧ 最良点更新 (m) ∧ 満足な手 (m) (4.2)

手(m) ← 最良点をとる手の呼出し (m) (4.3)

最良点の更新 (m) ← 先の最良手呼出し (m') ∧ {m の点 > m' の点}
∧ データベースに点と m を更新 (4.4)

最良点の更新 (m) ←

候補手 (3.3) ← 打着可 (3.3) } 全マスについてオ1次優先順に並べておく (4.5) (4.6)

候補手 (6.6) ← 打着可 (6.6)

⋮

手(m) が統合されると，まず定石を探す。なければ最初の候補手を選び仮に打つ。局面評価を J で行ない，それまでの最良手と評価点をデータベースに記録する。満足すべき手ならばそれが打着手となる。不満足であれば自動的に後戻りが行なわれ，次々と候補手が調べられる。全ての候補手について不満足であれば，データベースに記録された最良手を採用する。

ゲームプレイプログラムで特徴的なのは，満足すべきものが見つからなくても，その時点で最良の手を選ぶ必要がある点で，この点定理証明の場合と事情が異なる。このためデータベース操作による最大値登録を行なわねばならない。この機能の言語への導入が文献 [10] の集録機能で示唆されている。

関数Jはかなりの率で最適手を示すのがみられる（[9]を参考として）。しかし，20%ぐらいの率で最適でない手を示すので，これを定石あるいは先読みで補正する必要がある。ここでは定石によっている。

5. 学習法の表現と序盤への適用例

5.1 方針

知識が完全ならばゲームに敗れることはない。敗れた際にそのようなより良い知識ベース（即ち節集合）を自分で組織化してゆくための知識（メタ知識）を与えることを試みる。図5-1にそのようなシステムの概念図を示す。

Othello ゲームの序盤においては，既述の形式の定石の集合を改良してゆくことでほぼ十分であり，中盤に要求されるようなパタン認識的能力はほとんど必要ない。

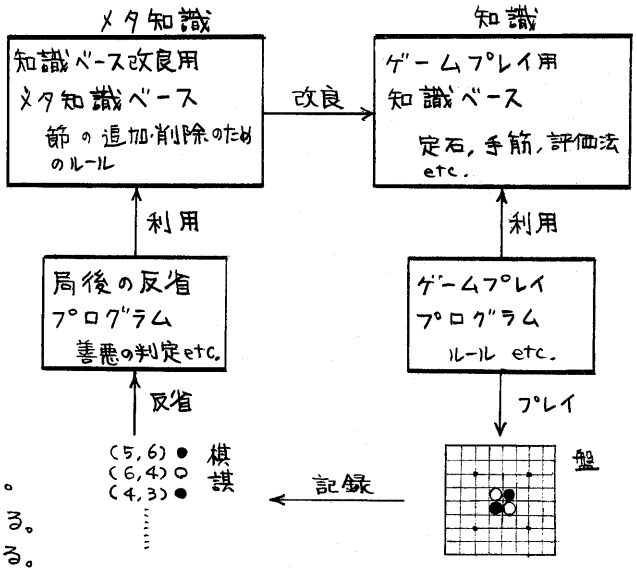


図5-1

良い手の定石に加え，同時に悪い手も記録し学習能率を高める。これを負の定石と呼ぶことにする。次のような節で示す。

+BM(*N,*K,*M).

変数はGMのものと同様である。

5.2 定石改良用メタ知識

4種の述語を次のように定義する。

(1) GOODM(n)　n手目は良い手である。

(2) BADM(n)　〃　悪い手である。

(3) GOODP(n)　n手終了した時の局面は，n+1手目の打着者の立場で良い局面である。

(4) BADP(n)　n手終了した時の局面は，n+1手目の打着者の立場で悪い局面。

すると次の関係は全く自然であり通常のゲームにおいてても使われる。

$$BADP(n) \leftarrow GOODM(n) \wedge GOODP(n-1) \quad (5.1)$$

$$GOODP(n) \leftarrow BADM(n) \vee BADP(n-1) \quad (5.2)$$

式(5.2)は次の2個の節で表わせる。

$$GOODP(n) \leftarrow BADM(n) \quad (5.3)$$

$$GOODP(n) \leftarrow BADP(n-1) \quad (5.4)$$

5.3 学習のための準備

ゲームプレイ中に悪い局面に印をつけておく。4-3の(4,3)において，悪い局面の場合には+BP(n).という節を記録する。良悪は関数Jに適当な基準を設けて判断する。中盤からの情報を利用する場合もある。

5.4 知識の改良

ゲーム終了後，-L.を発行すると反省プログラムが実行される。これは次の動作を行なう。

(1) +BP(n)があれば -BADP(n)を実行する。これはメタ知識(5,1)の頭と統合される。

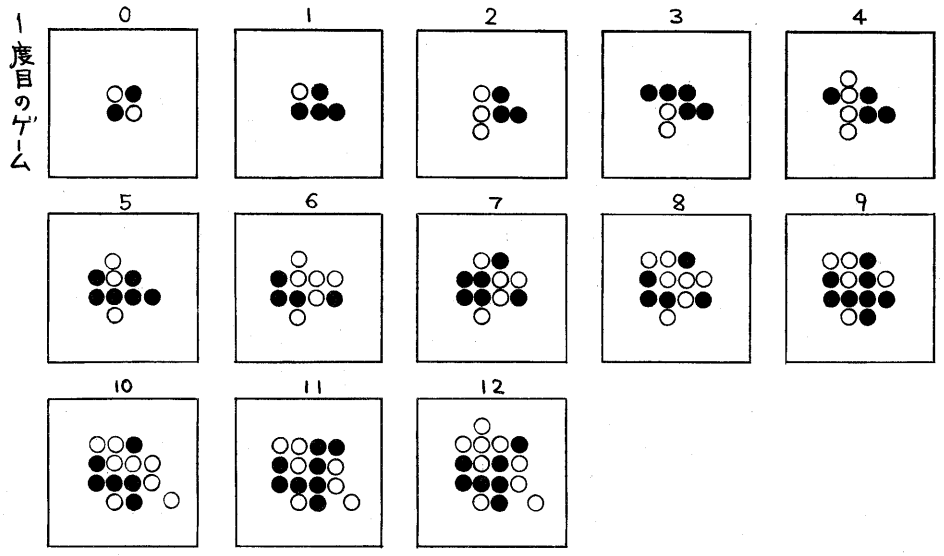
(2) -GOODM(n) 敵の手である。予想外の手であれば定石とする。

(3) -GOODP(n-1) メタ知識(5,3)の頭と統合される。場面が1手逆のぼる。

(4) -BADM(n) Jを用い次の候補を調べる。基準以上であれば特に打着手が悪かったのである。負の定石とし，実行終了。でなければ局面が悪かったので(5)。

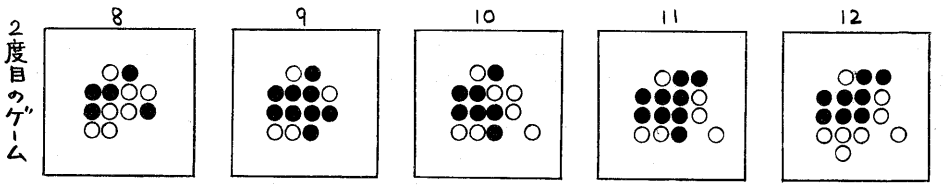
(5) -BADP(n) この局面からの負の定石があれば消す。(5,1)と統合され(2)へ。

このメタ知識は簡単であるが，必要な回数反復して使用できる有用なものである。図5-2に実行例を示す。計算機は白番である。12手目で白番が局面11が悪かったことを見つけた(100行目)。-L.を発行した。負の定石を作成した(500行目)。




```
100   +(BP(11)) . NIL ADDED
200   -L.
300
400   +(BP(9)) . NIL ADDED
500   +(BM(8,(0 . 0 . 0 . 0 . 0 . 0 . NIL) . (0 . 0 . 1 . 2 . 0 . 0 . NIL)
600   . (0 . 2 . 2 . 1 . 1 . 0 . NIL) . (0 . 2 . 2 . 1 . 2 . 0 . NIL) . (0
700   . 0 . 1 . 0 . 0 . 0 . NIL) . (0 . 0 . 0 . 0 . 0 . 0 . NIL) . NIL,3 .
800   3)) . NIL ADDED
900
```

2度目のゲームでも局面11が悪く，定石(1700行目)




```
1200  +(BP(11)) . NIL ADDED
1300  -L.
1400
1500  +(BP(9)) . NIL ADDED
1600  +(BP(7)) . NIL ADDED
1700  +(GM(5,(0 . 0 . 0 . 0 . 0 . 0 . NIL) . (0 . 0 . 1 . 0 . 0 . 0 . NIL)
1800  . (0 . 2 . 1 . 2 . 0 . 0 . NIL) . (0 . 0 . 1 . 2 . 2 . 0 . NIL) . (0
1900  . 0 . 1 . 0 . 0 . 0 . NIL) . (0 . 0 . 0 . 0 . 0 . 0 . NIL) . NIL,5 .
2000  3)) . NIL ADDED
2100  +(BM(4,(0 . 0 . 0 . 0 . 0 . 0 . NIL) . (0 . 0 . 0 . 0 . 0 . 0 . NIL)
2200  . (0 . 2 . 2 . 2 . 0 . 0 . NIL) . (0 . 0 . 1 . 2 . 2 . 0 . NIL) . (0
2300  . 0 . 1 . 0 . 0 . 0 . NIL) . (0 . 0 . 0 . 0 . 0 . 0 . NIL) . NIL,3 .
2400  4)) . NIL ADDED
```

図5-2 学習例

および負の定石（2100行目）が得られた。

この学習プログラムは人間の行なっている学習に近い動作を示している。4手目，5手目等で新知識が得られたが，これは文献[9]に照らしても妥当なものであった。学習プログラムでは，メタ知識の作成法が知能の良悪の鍵となる。

6. PROLOGについての考察

ゲームのためには既述の(1)集録機能に加え，以下の点が付加されると良い。

(2) 実行順と後戻り順を独立させる。これは右方にあるリテラルを左方のものの後戻りよりも後で後戻りさせることができるためで，式(4,2)の満足の基準を可変にすることが容易になる。

(3) データベース操作の機能の充実。特に任意の位置の削消去が必要である。

(4) 逆インタプリタ。データベースの内容を表示する。

(5) データ型の充実。アレイ，集合があると能率がかなり上がる。

(6) ユーザ定義の組込述語。固定した評価関数等は高速な言語で書いて使いたい。

7. 結び

ゲームプログラムの記述にPROLOGを用い，有効性を確認した。得られた知識をうまく蓄積できる点特に優れている。本稿では序盤について例を示した。中盤以後については次の機会に譲る。より高度な知識が要求される。

現在はFORTRANで記述されたインタプリタによっていることもあり，速度が特に遅い。また全てをPROLOGで書いているので膨大なメモリを使用している。後者は大容量メモリにも期待するが，言語の併用を考えたい。前者は文献[11]のような並列処理系が解決の鍵となろう。最後に，本研究の機会を与えて下さった当所佐藤孝平制御部長，PROLOGの応用を勧めて下さった田村浩一郎論理システム研究室長，インタプリタの便宜を計って下さった梅山伸二技官，並びに有益な御討論を頂いた同研究室の皆様に感謝する。

文献

[1] H. バーリナー：コンピューター・バックギャモン，サイエンス，オ10巻8号，pp.28-38（1980年8月）.

[2] 実近：ゲームプレイングプログラムの近年の成果，情報処理，Vol.20, No.7, pp.601-611（1979年7月）.

[3] 実近，田島．Relationship between Evaluation Function of Moves and Positions in Othello Game，電総研彙報オ43巻オ4号，pp.1-17（1979年4月）.

[4] R. Kowalski：Logic for Problem Solving, North Holland (1979).

[5] H. Gallaire, C. Lasserre：Controlling Knowledge Deduction in a Declarative Approach, Proc. of 6th International Joint Conference on A.I., pp. S-1～S-6 (Aug. 1979).

[6] 佐藤：プロローグ利用の手引，(株)エスジー.

[7] 佐藤：プロローグインタープリタ内部仕様書，(株)エスジー，これは[8]をもとにする

[8] G. Battani, H. Meloni：Interpreteur du Langage de Programation Prolog, Repport de D.E.A., d'Informatique Appliquee Universite d'Aix-Marseille (1973).

[9] 井上博：逆転の発見，(株)企画室ネコ（1977）.

[10] 田村，梅山：DIALOG-I：図式プログラミング言語，情報処理学会記号処理研究会資料12-8，pp.53-59（1980年6月）.

[11] 田村他：述語論理型プログラムの分散処理型アーキテクチャによる実現，情報処理学会記号処理研究会11-1（1980）.