

計算機による Reversi Game の実験 及び、ある種の Language に関する Idea

東大 理 竹内 郁雄

§.1 序

大上段に振りかぶって言えば、計算機を用いて Game をやってみる意義として、次の様子をものが列挙されるであろう。

(1) Sequential な Process によるある種の Pattern 認識の問題。

Game (又は Puzzle) での Rule や Strategy を計算機にのせるにあたって最も重要なものがある。例えば、囲碁の Rule を如何に計算機的に記述するかという問題を考えてみていただきたい。

(2) 計算機の Non-Numerical な応用としての Programming Technique の問題及び、Programming Language の問題。

これは上述の (1) と重複する意味もあるが、一般に Game や Puzzle が、図形的なものを処理する問題に与ることが多いことを考えると、この様なものを効果的に記述することの出来る Language の必要性が生ずる。よって、これ

が List 処理 (も、と局面を限って言えば、Linear Graph の処理) の言語に、刺戟と与える可能性が充分あると考えられる。(1)との関連で言えば、Game の Rule 等を、Formal に表現することが出来れば良いのである。

(3) Tree Searching の最適化の問題。

これは個々の Game にも依るが、ある意味では解決しているともいえる。しかし、Tree Search がより dynamic に進むにつれ、Game 自身の解析が絡んできて、話は容易ではなくなる。(参考文献 [1])

(4) Game 自身の解析。

Size の小さい Game では、完全な解析も可能である。又、部分的ではあっても、Strategy の評価と繰り返すことで Game の構造が解かることもある。

(5) Learning の問題。

計算機を人工知能への足掛りと心得ている人々は、学習機能の実験を Game で行うことが多い。Checker 有名な Samuel ([2],[3]) も、Game をやるのは、人工知能へのアプロ-4への手段であるとはっきり言っている。この Learning の問題は、余程理論的に堅固な基盤が与いと、人間が学習するだけに終る無残な Case も多いようである。

(6) Oasis!

解説の必要は無いです。

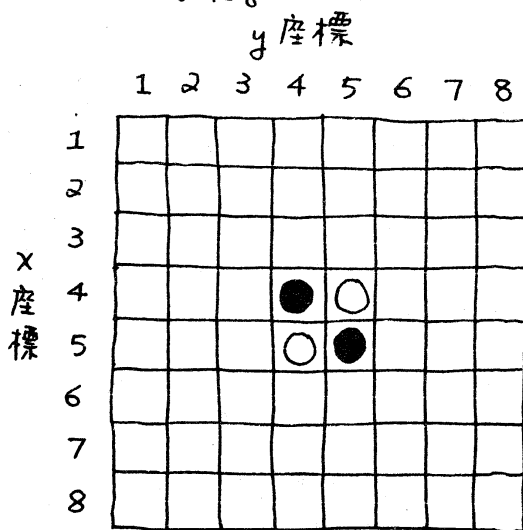
この小文で述べることは、'70年3月頃から正身3ヶ月位かけて行った実験の報告及び(2)で述べた様子ある種の言語に関するIdeaである。(研究会でL⁶という言語を教えていただいたので、以下の報告はそちらに供って加筆補正させていただきます。) 実験のデータの整理等、粗漏り点が多いのは御容赦下さい。

§.2 Reversi (源平碁ともいう。)のRule

M印登録商標の製品の裏に書いてある通りに記す。(この記述は、品物がボール紙からプラスチックに替った今も変わっていません。)

- 打込 ● 先づ中央の四区画に公平に二目づつ打込みます。次に先手から交互に任意の場所に打進みますが、必ず敵の石を何コかはさすねばなりません。はさめぬ時は相手が続けます。(石は片面が赤、片面が白になってくる。)
- 捕獲 ● はさんだ敵の石は何コでも裏返して味方の石にします。
- 勝負 ● こうして打ち終ったから石の多い方が勝です。”

このRule解説で先づ疑問矣であったのは、斜めにはさんだ敵石も捕獲してよいかどうかと、捕獲の連鎖反応を認めるかどうかであった。後者は、思考実験の段階で否定された。前者は、K氏との人間Simulationで3回連続32:32という信じられないような結果でもって、棄却することにした。(その後E氏からは、これは棄却の正当な理由にはならぬとの異議もあった。)尚、はっきりにさせるため、先手は赤石を持つことにした。



左図は、盤面と、その初期状態を表わしている。横の数字は計算機との対話に使う座標を表わしている。(x, yの順序) 例えば赤は、3-5, 4-6, 5-3, 6-4に石を打ち得る。

5.3. RVSの構造

表題のRVSとは、Reversiを計算機でやるために基礎として作った一種のSystem Programの呼称である。使用した計算機はTOSBAC-3000(8K語, 1語16bit, 基本的な演算時間は $2.4\mu s \sim 3.6\mu s$)である。RVSはAssemblerで書かれ、使用

語数は約3K (内 working space は. 300語) 弱である。

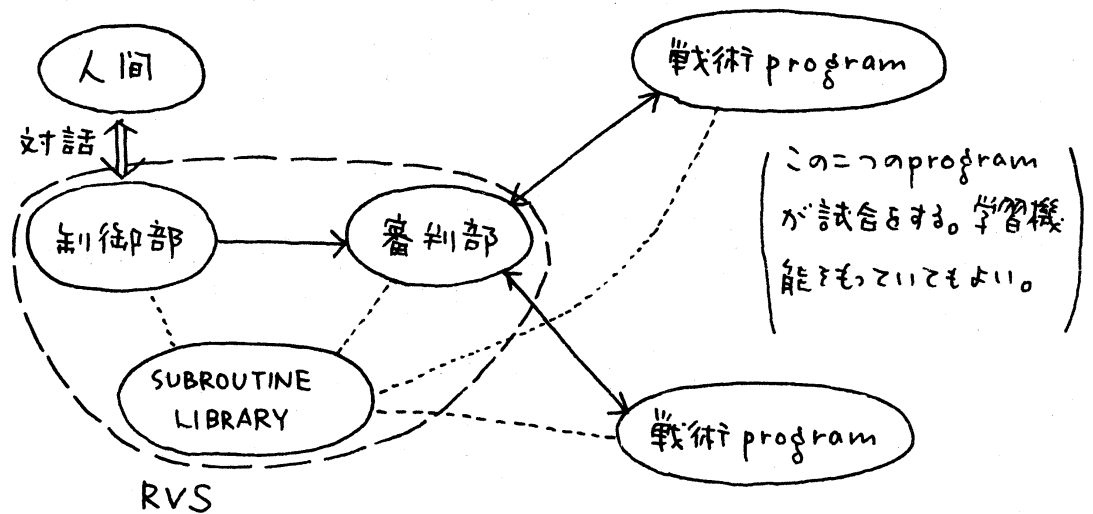
RVS は戦術 program とは別物であって、次の構造をもつ。

(1) 制御部 ----- 審判部の挙動を制御する。また人間に種々の情報と伝える。RVS と人間との対話は全てここを通し
て行われる。

(2) 審判部 ----- 実際の試合の進行を管理する。

(3) SUBROUTINE LIBRARY 及び DATA 部 ----- (1), (2) の使用する sub-
routine や data の部分で、その大半は戦術 program を使
用出来るようにしている。

試合を行なうには、別に作った戦術 program (RVS の sub-
routine が使用出来るので、この coding には余り負担はかか
らぬ。) を原則として2つ計算機に load してやるべき。
人間用としては、ただ「手」を入力するだけでよいので、RVS に
組み込んである。



審判の仕事と概括的に述べる。

- (イ) まず審判の持つ、ている各種の Table を初期化する。(実際には制御部で行う。)
- (ロ) 先手の戦術 program に control を渡し、そこから着手の報告(定められた形式で Register を通してやる。)があるまで待つ。この報告によつて、審判に control が戻ったとき、審判は、その着手の可否 (i.e. Rule 違反かどうか) を調べて可なりば、自分の持つ、ている各種の main table にその旨の書き込みをする。
- (ハ) 今度は、後手の戦術 program に control を渡す。このとき直前の敵の着手に関する情報等を伝える。以下、(ロ)と同様にして試合を続行させていく。尚、盤面、着手、計時の印刷は、必要があれば(← 制御部)行う。他に、待つタや人間による割り込み perturbation 等が可能である。
- (ニ) 試合終了と判定されると、結果の印刷をした後、RVS に control を戻す。(i.e. 対話モードになる) もし、学習モードの指定がしてあったときには、必要な情報を持って、各戦術 program の学習用 subroutine に飛ぶ。(学習機能を持たなくとも、一応空の subroutine を書式として書いておけばよろしい。) 各 routine からの帰りにば、手番交代要求が来たかみて、もしあれば、先手後手を入れ替える。

の論. この入替は、両方の戦術に伝えられる。この一連の仕事が終れば、再度試合を始める。

§.4 戦術 program のいくつかの実験について。

(1) 盤面の評価

計算機で Game をやる時、一番問題にするのは、今の評価をどうやればよいかということである。今の評価には、大まかに言って、Dynamic とものと、Static とものがある。大体 Dynamic とものは、今頃に depend するような評価である。高級子なのである。それに対して、Static とものは、着手された盤面のみに depend するもので、program を作り易く、後述の α - β method が使用出来るので時間の点からいって有利である。(以後、私は Static と評価だけを使っただが、S 氏の御注意にある通り、pass するやち何も打たないという着手は dynamic と評価にしかひっつからないうし、又、それが重要な point にもなっている様である。)

又、盤面の評価の方法にも荒っぽく分けて、2つの方法があると言えよう。一つは qualitative ともので、実はこれは前述の Dynamic と今の評価とほとんど同義といっているやものである。ある種の重大な状況があるため、先読みを続けなければならぬとか、絶対にその手を選ばないとかいっ

た風は、一般に第一は a quantitative を方法に優先さるもの
 ある。しかし、Reversi の場合、この方法の必要性が、そのほ
 ど感じられなかったので、専ら次の方法のみによった。第二
 としては、quantitative を評価が挙げられる。これは、一般に
 盤面の細分化された状況（これを認知する機構を Atomic Pre-
 dicate と呼ぼう。Minsky 等による Perceptron 思想（浮かべられ
 た）に重みをつけて加算したものを評価の依り所とするも
 のである。これは評価が scalar 量になるので、非常に取り扱
 いやすいし、学習の問題もこの辺が一番楽ではある。

すなわち、評価を最終的に scalar 量にして、あとは
 Min-Max Back Up を行い、その選択を行えばよい。

さて、quantitative を盤面の評価で最初に考えられるものは
 Order 1 の Atomic Predicate だけからなる評価関数 (RVS
 の subroutine library に用意がしてあって WSM と呼ばれる。
 Order 1 という言葉は、Minsky からの流用) である。すな
 わち、盤上の各目には重みを与えて、atomic predicate の各目
 の上に何色の石があるかを調べる。味方石ならその重みを、
 敵石ならその符号反転値を、空なら 0 を加算する。（敵石の
 ときは、符号反転値の 2 倍を加算したほうがよいという説も
 ある。）Reversi の場合、対角線に関する反転、二等分線に関

する反転で盤面の意味は変わらないうで重みとして10個のパラメータがあければよいのである。

0	1	2	3	
	4	5	6	
		7	8	
			9	

右図は、上に述べた parameter の座標を表わしている。これを加座標と呼ぶ。以後 $W(\alpha_0, \dots, \alpha_9)$ とかけば、0 の位置に α_0 という重み、 \dots 、9 の位置に α_9 の重

みが与えられていることにする。簡単な考察でわかる様に、0 の位置は絶対にひっくり返らないうで、かなり高い重みが与えられ、第一線 (edge) も一般に高く、第二線は低い重みでよいことがわかるであろう。経験者は、7 の位置もかなり高い重みを持つべきことを主張している。

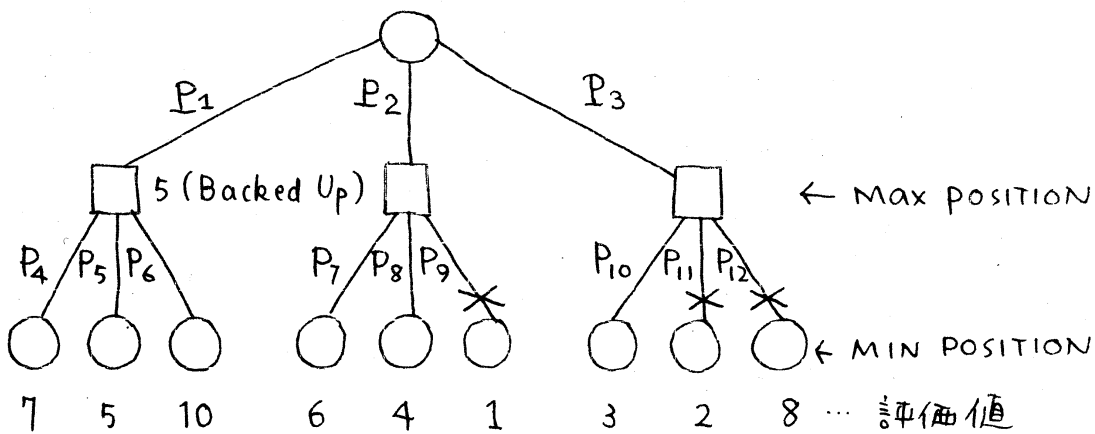
次に、Order 1 の評価関数が形にほとんど depend しないうで、実をもちたために、Atomic Predicate a Order を上げる事が考えられるが、少なくとも思考実験だけでは、意味が定かでない。parameter が飛躍的に増えるので、この等の実験を行なう、こみるには至らないうた。勿論、思考実験で不明確を故より、学習の実験として適当なものであるが、話が非常に大掛りとなり、時間の問題もあるので、今後の余裕を得つより他ないうたであろう。こういった手続きに関しては、Samuel を見るとよい。

しかし、ある意味で WSM でも Order を上げた様子効果を持つ

たせる細工は出来る。すなわち、石が打たれた時に、その近傍の目の重みと適当に変更して、あたかも形の認識が重みに影響する様にするのである。特に、このとや、で終盤で、各目の重みが平均化されるならば、評価は正否に石の数では負けるような稽事（実際をんを記録はある。）が無くなることも期待される。この種の戦術 program としては、VAL2付き（打たれた位置の近傍ではなく、その位置だけを一定値にかえる。）と、CHNW付き（打たれた位置の近傍を平均化の方法で地をらしめるもの。）を作った。

(2) α - β method

4の評価が static なときには、この α - β method の Min-Max Back Up によるのと同じ4の選択を、約数分の1の手間で得ることが出来る。原理は図で説明する。



図でX印のしたある枝は、もはや探索する必要がないので

ある。例えば P_9 は P_8 の評価値が、Max Player が P_1 を選ぶことにより保証される5を下まわすため、もはや P_2 を選ぶ可能性をなくしてしまつたため、当然、探さぬことは正しいのである。一般に先読みの手数が増えても、この考え方を拡張すればよい。(具体的事例は、付録を見らねたい。)

尚、ここで注意すべきことは、この枝刈り(図でX印をつけること)の個数がかなり遅に依ることである。例えば、図で、枝の探索が P_3, P_2, P_1 の順で行なわれたら、枝刈りは一回も出来ぬ。これを避けるため、枝刈りが最も効果的に行なわれる様、 P_1, P_2, P_3 を並べ変える方法を考えねばならぬ。この Reordering は Game の種類にもよるが、先読みの途中の level での評価が良ければ、先読みの先端の level でも評価は良いという予想のもとに行なうのが一般的であろう。そこで先読みの level を固定してこれを Reordering と行なう方法を Fixed Ordering (FO) と呼ぶ。この level は、先読みの先端 level から2〜3度、たどることが Optimal であろうと言われている。(勿論、評価関数や Tree Search の手間がどうかによつて変わる。) 他に、level が Dynamic に変わる Dynamic Ordering という方法もある。この辺の事情に関しては [1] を見らねたい。

私の現在までの戦術 program では、副産物的に FO がついで

11) だけで効果の薄いものがある。

α - β method の効果の例を次に挙げる。この Data は、7
手先読み戦略 program に細工をして打ち出したものがある。
見易さのために多少手を加えてある。

RVS T, TIMER MODE 3, BOTH

RVS W, WEIGHT 300,30,50,40,3,2,3,20,10,1,

INTO 15242,

RVS A, SET

RVS R, RUN

			(後列した 後の Tree の末端の数)	(前列した ときの Tree の末端の数)
* 3-5 (12; 40)	6SEC	203	832	
5-6 (0; 5)	5SEC	131	642	
* 6-5 (12; 53)	5SEC	147	560	
3-4 (0; ?)	2SEC	0	1435	
* 3-3 (40; 74)	9SEC	368	5201	
4-6 (-28; -47)	9SEC	371	4278	
* 5-3 (40; 98)	27SEC	1268	8660	
2-4 (-17; -52)	30SEC	1887	12627	
* 1-4 (85; 56)	26SEC	1455	9160	
4-3 (-73; -2)	20SEC	1275	13627	
* 5-7 (96; ?)	2SEC	0	8032	
6-6 (-56; 37)	37SEC	2812	48188	
* 6-7 (98; ?)	1SEC	0	15855	

*印は赤の打った手である。左端は手の x, y 型の表示で、

次の () 内の α 二つの数値は、その手を打ったときの盤面の評
価と、7手先を読んだ Backed Up Value と、謂わば予想であ
る。その後には書いてある α は α - β method によったときの
時間である。これからも、如何に α - β method が効果的であ

3 かは一目瞭然である。尚、Backed Up Value が? になっているのは、この手が唯一の可能な手なので先読みを行なわなかったためである。従って、Tree の末端も 0 になる。又、この場合でも、1~2 秒かかっているのは、印刷時間のためで正味の計算時間はもっと短い。ところで、一般に中盤になると、Tree は莫大になつて、 α - β method でも 200 秒近くかかることがある。人間との対局のために、FO をもっと効果的につかつて、少くとも半分の時間に出まると思われる。(ちなみに、この勝負は、3 対 61 で後手が圧勝する。総計所要時間は、約 20 分)

(3) 各種戦術 program の実験結果

特に断りなしの限り、重みは $W(300, 30, 50, 40, 3, 2, 3, 20, 10, 1)$ で実験した。この数値には何ら、理論的背景はない。識者の意見によつたものである。 n 手読みとは、先読みの先端の level が n であることを示す。

(a) 3 手読み (一手にほとんど時間はかからず)

先手であれば、初心者 + α の程度の人間といい勝負。後手をやらせると 初心 - α の人間にも完勝する。

(b) 5 手読み (α - β で一手平均 10 秒以内)

中級者以上の実力をもち、先手、後手と両方にやらせ

ると、先手が勝つ。後手でぼろしまぎい手とうつことがある。

(c) 7手読み (α - β で平均30秒位)

かなり強い。名人級ではあるが、やはり熟考した名人には勝てない。この同志対戦させると、後手が勝つ。

(d) 9手読み (かなり長大)

このでも、同志戦ちをやらせると、後手が勝つ。初盤で、perturbationを加えても同じ。この等の結果をみると、このゲームは後手必勝?とも思えてくるが、これが人間同志の対戦から出る結論と逆なのが不思議である。

(e) 3手読み学習付 (VAL2付きとVAL2無しのものがある。)

ここという学習とは、学習と呼ぶより、反省強化といった単純な類のものである。過去のGameで各手には石が打たれたから何回反転したかをreferして、手の評価に適当なperturbationを与えるものである。すなわち、5回前までの反転数と記録しておき、3手先読みBacked Up Valueと、次の手の反転重み(例えば、過ぎずっと偶数回の反転であれば、より好ましく、逆に奇数回であれば、よくなる)。これを、反転数に対応した数値の和で評価する。)を加算して、その総合値で、手を選択するものである。反転数重みと、重み(位置 α)、VAL2の相関を調べ尽したわけではないうが、VAL2付きのものは期待に反して効果がないうようである。これに比して、VAL2の付い

こゝのものは、後手で3手読み、5手読み、7手読みには、数回で勝てるようになってきた。この時使用した反転重みは、10, -10, 20, -20, 30, -30, 30 (すなわち、反転0回には10点、反転1回には-10点という具合)であり、重みは特別に、

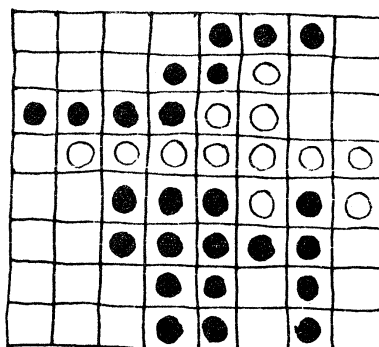
$W(1000, 200, 400, 300, 3, 2, 3, 30, 20, 1)$ を使用した。第一線(edge)を強調するのは、反転重みによるperturbationをあまり隈のすでは強く受けたいようにするためである。すなわち、この学習(反省強化)は、初盤のperturbationを行なうだけである。(反転重みのparameterを見てもそれは明らか) 逆に言えば、初盤、中盤が如何に大事な物語っているともいえよう。勿論、このような単純な方法が意外に効果的であったのは、相手が決定論的playerであったからに相違ないであろう。

尚、この学習programは先手になると、後手でやったほど冴えないと思われよう。

例として、(このGameが如何に微妙であるかという例でもある)、このprogramが7手読みで9回目で勝利したときの勝負を再現しよう。終盤で信じられないようなことが起っている。傍線を引いた部分が赤の手番で、必要があれば()の中に、現在局面の評価と7手先のBacked Up Valueを記す。

3-5 3-4 3-3 4-6 6-4 6-5 6-6 5-3 4-7 2-5
1-5 5-6 PASS 3-6 2-6 5-7 6-7 5-8 PASS 7-7
8-7 7-5 8-5 7-4 8-4 1-6 1-7 3-2 2-4 4-8
3-1 4-3 6-3 4-2

ここで盤面の状態は右図の
 様になっている。この辺で
 最早、本は勝てない局面で
 あるらしい。



5-2 5-1 6-2 2-3 1-3 2-2 1-2 7-3 8-3 7-6

次のあたりから、暴落が始まる。

8-6 (502; 689) 3-7
1-4 (506; 790) 2-7
2-8 (545; 842) 4-1
6-1 (689; 382) 7-2
8-2 (728; -398) PASS
3-8 (842; ?) 1-8

PASS 1-1 PASS 2-1 PASS 7-1 PASS 6-8 PASS 7-8
PASS PASS

結局、44:18で後半白の圧勝となる。上図の局面で、い
 る3を perturbation を与えて、9手読み同志でやらせても
 きれどいとここで反転劇が起る。

(f). 5手読み CHNW付き。

Weight の与え方によって、微妙な差があらわれる様であ

る。標準 weight では、普通の 5 手読みとやって、先手で勝ち後手で負け。 $W(1000, 200, 400, 300, -30, -20, -40, 40, 30, 1)$ では、先手で負け、後手で勝ちと出る。つまり、戦術の今後の方向は、この CHNW を如何に強くするかにあるようである。試合終了時には、weight は全面的により平均化をしている。現在の CHNW ではどんなことをしているかその例を示す。説明で w はその位置の weight と示している。

	6			
7	2	3	4	5
	7			

もし 1 の位置に石が打たれると

$$\text{mean1} := \{w(1) + w(2) + w(3) + w(4)\} / 8$$

$$w(1) := w(1) / 2 + \text{mean1}$$

($w(2), \dots, w(5)$ について同様)

$$\text{mean2} := \{w(6) + w(2) + w(7)\} / 8$$

$$w(6) := w(6) / 2 + \text{mean2}$$

($w(2), w(7)$ についても同様)

又、もし、2 の位置に石が打たれると $w(2) := w(3)$

... etc

(♯) WEIGHT を学習させる問題。

これは、非常に興味ある問題であるが、困難である上、Order 1 の Atomic Predicate が使用しきれない盤面評価関数自身にも疑問があるのだ。簡単を考察にのみとどめる。

WEIGHT を変えていく動機としては、次のものがある。

(イ) 石の反転数

(ロ) pivot (敵石をはさむときの相棒の石のこと)として、

如何に使用されたか?

(ハ) 強いはずの敵の着手と自分の予想との食い違い。

(ニ) 同じ結果を得るのに必要の WEIGHT の最小限のバラツキへの抑制。(parameters の爆発防止)

§.5 Game の Rule と形式的に表現すること。

Game の Rule と Automaton の類似性については、以前から気がつかれていることであるが、ここは抽象性よりも、記述の具体性を重んじるつもりで、話を進めよう。

最初に Game (Finite, Imperfect Informational) を次の枠組で考える。

DEFINITION Game とは $\{P, S, \delta, p, F, \{f\}_P, I\}$ のことである。ここで

P : players の集合

S : state の集合 (例えば盤の状態、その他 Game の進行に関わる種々の flag 等) - 一般に $S = S_1 \times S_2 \times \dots \times S_n$ という形にしている。

δ : $P \times S \rightarrow \mathcal{P}(P \times S)$ (Rule にあたって (i.e. admissible) の next state と player の順番に関与する非

決定的な transition map)

$F \subseteq P \times S$ (Game の最終状態。ここにくると Game は終了する。)

$I \subseteq P \times S$ (Initial 状態。)

$\{f\}_P$ (一般に $\#(P)$ 個の函数) $\{f\}_P \ni f : F \rightarrow P \times S$ (これは勝負や pay-off を表現する函数である。)

$\rho : P \times S \rightarrow \{S \text{ の restriction (projection) } \}$: の意味は Imperfect Informational Game に対応するものである。 $(p, s) \in P \times S$ なる状況では player p は知らなかった state s としては $(\rho(p, s))(s)$ しか知らないことを示す。この ρ と δ と α の間には自然な compatibility のあることは仮定する。(可なり、或る player にとって利用しうる情報だけでの手の選択が出来るということ) //

さらに、計算機との関連を考える時には、上の定義のような δ よりも、いわゆる Input alphabet に対応できるように τ の $\{T, T\}$ を付け加えて、 δ を取り扱やすくしてやった方が便利であろう。

T : 着手に Name を付けたものの集合 (例えば 3-5 石打ちとか、四歩打とか)。一度、入力記号に相当する。

$\tau : P \times S \rightarrow \mathcal{P}(T)$ admissible な着手の集合。

$\delta: T \times P \times S \longrightarrow P \times S$ (ただし, (t, p, s) が
 $t \in T(p, S)$ のときに define される)

この枠組から, 審判の行動を記述することは, 普通のプログラミング言語で十分容易である。しかし, 実際には問題となるのは, 大きき枠組の記述よりも, 個々の S , δ , τ 等を如何に記述するかである。例えば, S の表現一つにしても, 非常に沢山の方法が考えられる。しかし, 大きく分けて, 二つの種類があると思われる。

1. (個々の Game の特性を十分に吟味して), 計算機で容易に取扱え, 計算出来るような表現。Assembler 段階で作れば, 計算時間において抜群の効果を持つような表現が考えられる。しかし, あきらかに Machine independent にあり, 一般性が高い。又, もう一つの面として, ほとんど大体 array 的に処理するような方法 — ある程度の計算時間の経済は保証されるだろうが — がある。これは, ある意味で Language Dependent な方法で, 苦しむところである。

2. 人間の直観により強くなり, たえるような表現。

今後の Software の進歩の一部は, 2. のような表現を最適

5. DATA に関する基本的な述語 & action.

他に、記憶容量、時間など、programmingの楽々から、次のものも加えらるべきだろう。

6. List (ringed もよい) が program の中で dynamic に使われるか、static に使われるかを宣言することによって static の場合には 113 113 を節約 (内部的) が出来る。

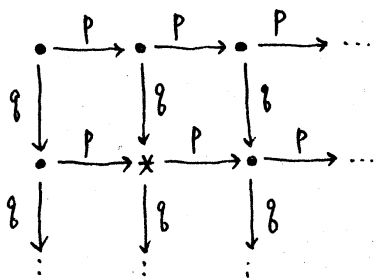
7. BRANCH に coloring が出来る。

8. NODE, BRANCH に mask (一時的にその存在を無視すること) や, unmask が 113 113 を Level (深さ) で出来ること。

9. ARRAY もつかえる。

次に幾かの Program 例 (Language Specification もしき) で例を出すのは、それがまだ Idea にとどまり、113 からである。) を列挙して、上述のことを具体化してみよう。記述は L⁶ にかなり似ている。

a. Reversi である位置に石がうてるかという述語。



左図の様に、盤を表現して、右横方向の branch には p, 下縦方向の branch には g という '色' をつける。逆方向は接頭辞 rev- をつける。

今の場合、NODE上の情報は、そこに何色の石があるかだけであるから、NODE上のDATAは1 string での。('VOID', 'FRIEND', 'ENEMY')。さて、*の位置に石が打てるかどうかの述語は次の様にする。 (|| は comment の意。 ~ は省略可能なもの)

$$\underbrace{[\bigvee (\text{for } \pi \in \text{set}(p, q, \text{rev-}p, \text{rev-}q)) \text{ PRED}_1(\pi, *)]}$$

|| branch を $p, q, \text{rev-}p, \text{rev-}q$ にして、各々で PRED_1 を検し

|| て、その Or をとる。ここを

$\text{PRED}_1(B, N)$: predicate type (branch, node)

v-[[$\rightarrow \text{BUG1} = N$]

[$\text{BUG1} = \text{'VOID'}$]

[$\rightarrow \text{BUG1} = \text{successor}(B : \text{BUG1})$]

[$\text{BUG1} = \text{'ENEMY'}$]

L1: [[$\text{BUG1} = \text{successor}(B : \text{BUG1})$] \rightarrow

[$\text{BUG1} = \text{'ENEMY'}$] \rightarrow goto L1,

[$\text{BUG1} = \text{'FRIEND'}$] \rightarrow true,

\rightarrow false],

\rightarrow false]]


|| BUG の意味は L⁶ a ∈ a と同様。→ は then の意。BUG1

|| の後の a period は、data a 構造に対応して いる。L1 は

|| label。recursive call が される。→ の片側には ない

- || も書かなかったのは, locally true の意。locall の意。
- || 味は \exists が, \exists の囲む [] の中での意味をたずね
- || ということであり, \exists に反して, true, false は \exists の
- || 値をもつて, 指定された block の外へ飛び出す力をもつ
- || ているのである。

b. Pentomino の例。

次に, pentomino の便器 ( 形のもの, この名称は私が
 中学時代から愛用しているものにつき断固使用する。ちなみに
 他の優れた名前を紹介すると, P は田吾作, L は曲者, T
 は殿, F は馬の首 etc.) が, 盤上におけるかどうかの述語を
 記してみよう。(反転は, 今考えよう。)

$PRED_2 (*, \mathcal{D})$: predicate type (node, branch)

v- [$\rightarrow BUG2 = *$]

[$BUG2. = 'VOID'$]

[$\rightarrow BUG2 = \text{successor}(\mathcal{D}; \text{BUG2})$]

[$BUG2. = 'VOID'$]

[$\rightarrow BUG3 = BUG2$]

[$\rightarrow BUG2 = \text{successor}(f_1(\mathcal{D}); \text{BUG2})$]


[$BUG2. = 'VOID'$]

[$\rightarrow BUG3 = \text{successor}(\mathcal{D}; \text{BUG3})$]

[BUG3 = 'VOID']

[\rightarrow BUG3 = successor ($f_2(\mathcal{D})$: BUG3)]

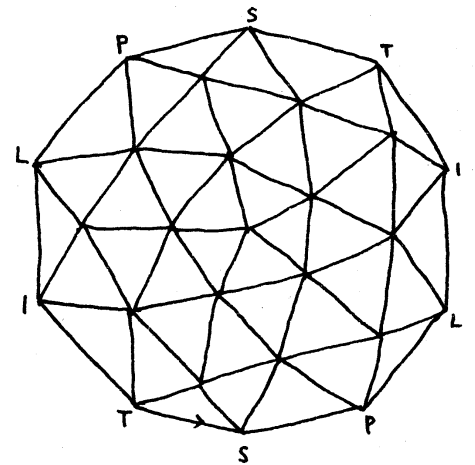
[BUG3 = 'VOID']]

||  \mathcal{D} は direction 位の意。 $f_1(\mathcal{D})$ は ($p \rightarrow q$, $q \rightarrow \text{rev-}p$, $\text{rev-}p \rightarrow \text{rev-}q$, $\text{rev-}q \rightarrow p$) 及び BRANCH から BRANCH 1 の函数。 $f_2 = f_1 \circ f_1 \circ f_1$ 。

c. Graph の connectedness を調べようとする program.

次に SPLIT と呼ばれる Game を紹介する。盤面は、図のよ

うにまわっていて、Rule は、一手交代で自分の石を置き、先に境界の対称点同士を連結させた方が勝である。尚、境界に限って対称点に一度 = 個の石を置いてよい。この盤面で、二つの NODE が自分の石で連結しているかどうかを調べる述語を書いてみよう。



(branch は q と同じ色付け)

PRED₃(POS₁, POS₂) : predicate type (node; node)

[for $N_1 \in \text{NODES}$ [$\underbrace{N_1}_{\text{node}} \neq \text{'FRIEND'}$] \rightarrow mask $\underbrace{N_1}_{\text{node}}$ level (α)]

\forall [$\rightarrow N_2 = \text{POS}_1$] [$\rightarrow \text{init}(\beta)$]

L2: [$\rightarrow \text{pushdown}(\beta)$ successor ($q \vee \text{rev-}q : N_2$) with $m(\alpha)$]

[$\rightarrow N_2 = \text{pop-up}(\beta)$]

■ [$N_2 = \text{pos}_2 \rightarrow T, \rightarrow \text{mask prebranch}(N_2) \text{ in } (\beta) \text{ level } (\alpha)$]

[$\rightarrow \text{goto L2}$]]

[[$\rightarrow \text{maskfree node all level } (\alpha)$]

[$\rightarrow \text{maskfree branch all level } (\alpha)$]]

- || まず味方の石 α の部分と graph から一時的に抹殺する。
- || その level を α と指定する。次に β という、あらかじめの用意された Pushdown List を初期化する。 pos_1 の successor で α -level の mask がかかっているものも、 β に書き込む。尚、successor が ϕ であるものは、この [] の中の値は locally false である。(他にも、この種の predicate 付きの action は多い。) prebranch と mask するというのは、この program が同じところを堂々巡りさせているのである。|| 忘れたが、[] の前の v -は、それがこの PREDICATE の値に与ることと示す。

以上の example は、完全なものではない。もう少し || 表現は考えらると思う。私の現在の考えでは、この等の言語の Processor は L^6 の拡張(言語と機能と)と、簡単な Macro Processor があるかは何か処理できるのではないかと思ひ、非力ながらそれを少し試してみようと思つて || いる次第である。

Game に話と限って考えても、例えば碁の取り扱いは Dynamic と Graph の操作などでかなり最適時間どかつ。理論的に興味のあるものが出まると思われる。

参考文献

[1] J. R. Slagle & J. K. Dixon: Experiments with some Programs that search Game Trees (1969) J. A. C. M vol 16 no. 2

[2] Advance in Computer, vol 1

A. L. Samuel: Programming Computers that Play Games

[3] A. L. Samuel: Some Studies in Machine Learning using the Game of Checkers II. Recent Progress (1969)

Annual Review in Automatic Programming vol. 6.

[4] K. C. Knowlton: A Programmer's Description of L⁶.

Com. A. C. M vol. 9 (616~625) (1966)

最近、次の様子ちよと面白い本が出た。

[5] R. Banerji, M. D. Mesarovic, ed.: Theoretical Approaches to Non-Numerical Problem Solving (1970)

(Lecture Note in Operation Research and Mathematical Systems, No. 28. Springer Verlag)