数理解析研究所講究録
第98巻 1970年 30-56

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、次の様な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である。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な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（もっと局面を限って言えば，Linear Graphの処理）の言語に，刺戟を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になるにつれ，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の評価を繰り返すことで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 で行なうことが多い。Checker で有名なSamuel（［2］,［3］）も，Game をやるのは，人工知能へのアプローチ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ないと，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あろう。

この小文で述べることは、'70年3月頃から正身3ヶ月位かけて行なった実験の報告及び(2)で述べた様なある種の言語に関するIdeaである。(研究会でL^6という言語を教えていただいたので、以下の報告はそれに従って加筆補正されています。) 実験のデータの整理等、粗漏な点が多いのは御容赦下さい。

§.2 Reversi (源平碁ともいう。)のRule

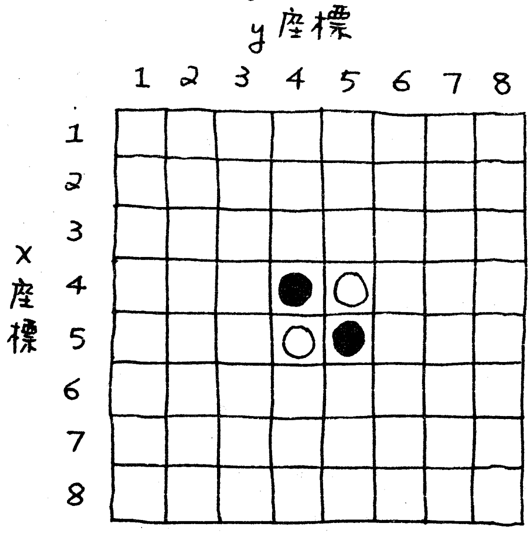
M印登録商標の製品の裏に書いてある通りに記す。(この記述は、品物がボール紙からプラスチックに替った今も変わっていない。)

"●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まねばなりません。はさめぬ時は相手が続けます。(石は片面が赤、片面が白になっている。)

●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● こうして打ち終ってから石の多い方が勝です。"

このRule解説で先ず疑問点であったのは、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ないとの異議もあった。）尚、はっきりさせるため、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われる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。

§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Sytem 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2.4μs〜3.6μs）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。

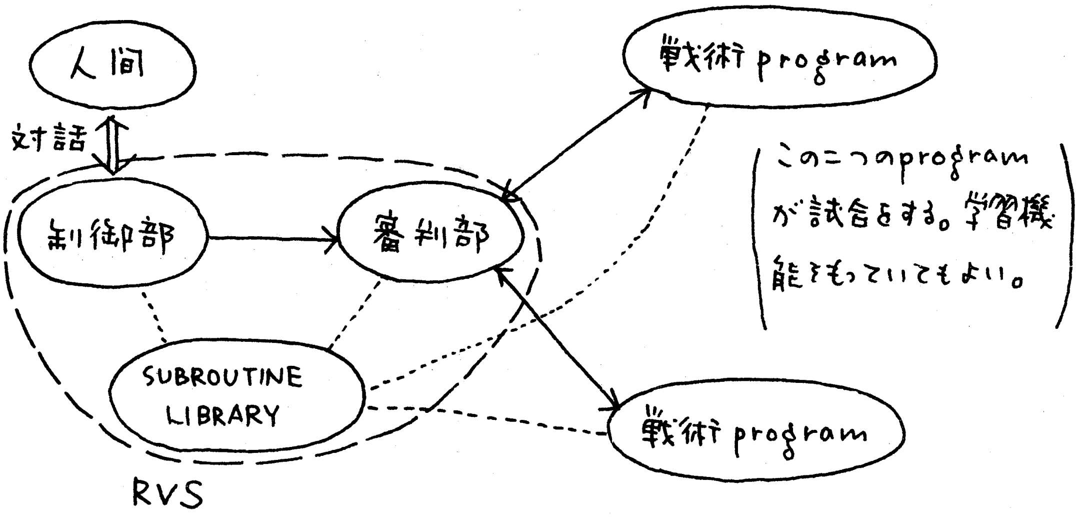
RVSは戦術programとは別物であって、次の構造をもつ。

(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を伝える。RVSと人間との対話は全てここを通して行なわれる。

(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subroutineやdataの部分で、その大半は戦術programも使用出来るようになっている。

試合を行なうには、別に作った戦術program(RVSのsubroutineが使用出来るので、これのcodingには余り負担はかからない。)を原則として2つ計算機にloadしてやればよい。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。



審判の仕事を概括的に述べる。

(イ) まず審判の持っている各種のTableを初期化する。(実際には制御部で行なう。)

(ロ) 先手の戦術programにcontrolを渡し、それから着手の報告(定められた形式でRegisterを通してやる。)があるまで待つ。この報告によって、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)を調べて可なれば、自分の持っている各種の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)行なう。他に、待ッ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ても、一応空のsubroutineを書式として書いてなければならない。) 各routineからの帰りには、手番交代要求がなかったかみて、もしあれば、先手後手を入れ替える。

勿論、この入れ替えは、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、一番问題になるのは、手の評価をどうやればよいかということである。手の評価には、大まかに言って、Dynamicなものと、Staticなものがある。大体、Dynamicなものは、手順にdependするような評価である。高級なのである。それに対して、Staticなものは、着手された盤面のみにdependするので、programも作り易く、後述のα-β methodが使用出来るので時间の点からいっても有利である。(以後、私はStaticな評価だけを使ったが、S氏の御注意にある通り、passすなわち何も打たないという着手はdynamicな評価にしかひっかからないし、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なもので、実はこれは前述のDynamicな手の評価とほとんど同義といっていいようなものである。ある種の重大な情況があるため、先読みを続けなければならないとか、絶対にその手を選ばないとかいっ

た風に、一般に第二のquantativeな方法に優先されるものである。しかし、Reversiの場合、この方法の必要性が、それ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、quantativeな評価が挙げられる。これは、一般に盤面の細分化された情況（これを認知する機構をAtomic Predicateと呼ぼう。Minsky等によるPerceptronを想い浮かべられたい。）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scalor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ろう。

いずれにせよ、評価を最終的にscolor量にして、あとはいわゆるMin-Max Back Upを行なって手の撰択を行なえばよい。

さて、quantativeな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、盤上の各目に重みを与えて、atomic Predicateで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ので重みとして10個のパラメーターがあれば充分である。

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。これを枡座標と呼ぶ。以後 $W(\alpha_0, \cdots\cdots, \alpha_9)$ とかけば、0の位置に α_0 という重み、……、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ので、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。

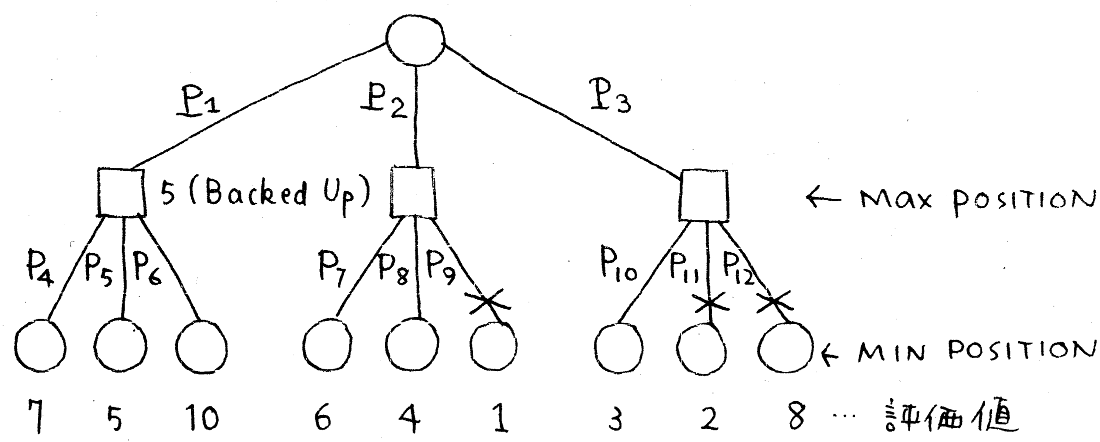
次に、Order 1の評価関数が形にほとんどdependしない欠点をもつために、Atomic Predicate のOrderを上げる事が考えられるが、少なくとも思考実験だけでは、意味が定かではなく、parameterが飛躍的に増えるので、これ等の実験を行なってみるには至らなかった。勿論、思考実験で不明確な故より、学習の実験として適当なのであるが、話が非常に大掛りとなり、時間の問題もあるので、今後の余裕を待つより他ないであろう。こういった手続きに関しては、Samuelを見られたい。

しかし、ある意味でWSMでもOrderを上げた様な効果を持

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を適当に変更して、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、これをやって終盤で、各目の重みが平均化されるならば、評価は正なのに石の数では負けるような惨事（実際そんな記録はある。）が無くなるとも期待される。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ならしするもの。）を作ってみた。

(2) α－β method

手の評価が static なときには、この α－β method で Min-Max Back Up によるのと同じ手の選択を、約数分の一の手間で得ることが出来る。原理は図で説明する。



図でX印のしてある枝は、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4点が，Max PlayerがP_1を選ぶことによって保証される5点を下まわるため，もはやP_2を選ぶ可能性をなくしてしまったため，当然，探されることはないのである。一般に先読みの手数が増えても，これの考え方を拡張すればよい。（具体的な例は，付録を見られたい。）

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）の個数がかなり運に依ることである。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ない。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ない。このReorderingはGameの種類にもよるが，先読の途中の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。そこである先読みのlevelを固定してそこでReorderingを行なう方法をFixed Ordering（FO）と呼ぶ。このlevelは，先読みの先端levelから2〜3戻ったところがOptimalであろうと言われている。（勿論，評価関数やTree Searchの手間がどれくらいかによって変わる。）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]を見られたい。

私の現在までの戦術programでは，副産物的なFOがついて

いるだけで効果の薄いものである。

α−β method の効果の例を次に挙げる。このDataは、7手先読みの戦術programに細工をして打ち出したものである。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
```

				(枝刈りした後の、Treeの末端の数)	(枝刈りしないときの、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で、謂わば予想である。その後に書いてあるのは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

るかは一目瞭然である。尚、Backed Up Value が？になっているのは、そ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かい。ところで、一般に中盤になると、Treeは莫大になって、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[redacted]、FOをもっと効果的につかって、少なくとも半分の時間に出来ると思われる。（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

特に断わらぬ限り、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。[redacted]識者の意見によったものである。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ない）

先手でやれば、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の人間にも完敗する。

(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つ。先手、後手を両方これでやらせ

ると、先手が勝つ。後手では少しまずい手をうつことがある。

(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。これ同志対戦させると、後手が勝つ。

(d) 9手読み（かなり長大）

これでも、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。これ等の結果をみると、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。）

ここでいう学習とは、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で各目に石が打たれて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回前までの反転数を記録しておき、3手先読みのBacked Up Valueと、次の手の反転重み（例えば、過去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、よくない。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、手を選択するのである。反転数重みと、重み（位置の）、VAL2の相関を調べ尽したわけではないが、VAL2付きのものは期待に反して効果がないようである。これに比して、VAL2の付い

ているものは、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，

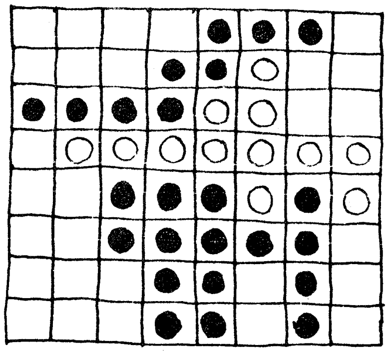
W(1000, 200, 400, 300, 3, 2, 3, 30, 20, 1)を使用した。第一線(edge)を強調するのは、反転重みによるperturbationをあまり隅の方では強く受けないようにするためである。すなわち，この学習（反省強化）は，初盤でperturbationを行なうだけである。（反転重みのparameterを見てもそれは明らか）逆に言えば，初盤，中盤が如何に大事かを物語っているともいえよう。勿論，このような単純な方法が意外に効果的であったのは，相手が決定論的playerであったからに相違ないであろう。

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る。

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手読みに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、赤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

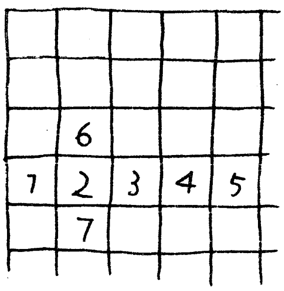
PASS 1-1 PASS 2-1 PASS 7-1 PASS 6-8 PASS 7-8
PASS PASS

結局、44：18で後手白の圧勝となる。上図の局面で、いろいろなperturbationを与えて、9手読み同志でやらせてもきわどいところで反転劇が起る。

(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。いずれにせよ、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は全面的によい平均化をしている。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を示している。



もし1の位置に石が打たれると

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

$$w(1) := w(1)/2 + mean1$$

($w(2), \cdots, w(5)$ について同様)

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

$$w(6) := w(6)/2 + mean2$$

($w(2), w(7)$ 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

(g) 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しか使用しない盤面評価関数自身にも疑問があるので、簡単な考察にのみとどめる。

WEIGHT を変えていく動機としては、次のものがあろう。

(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なWEIGHT の最小限のバラツキへの抑制。(parameters の爆発防止)

§.5 GameのRuleを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、ここでは抽象性よりも、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, Inperfect Informational)を次の枠組で考える。

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

$\mathbf{P}$: playersの集合

S: stateの集合(例えば盤の状態、その他Gameの進行に関する種々のflag等)一般に $S = S_1 \times S_2 \times \cdots \times S_n$ という形をしている。

δ: $\mathbf{P} \times S \longrightarrow \mathfrak{P}(\mathbf{P} \times S)$ (Ruleにかなった(i.e. admissible)なnext state とplayerの順番に関する非

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 な状態。)

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

$\rho : P \times S \longrightarrow \{S$ の restriction (projection)$\}$ この意味は Imperfect Informational Game に対応するものであって $(p, s) \in P \times S$ なる情況では player p に知らされた state としては $(\rho(p, s))(s)$ しかないことを示す。この ρ と δ との間に自然な compatibility のあることは仮定する。(すなわち、或る player にとって利用しうる情報だけで手の選択が出来るということ) //

さらに、計算機との関連で考える時には、上の定義のような δ よりも、いわゆる Input alphabet に対応するようなもの $\{\tau, T\}$ を付け加えて、δ を取り扱いやすくしておいた方が便利であろう。

T : 着手に Name と付けたものの集合 (例えば 3-5 石打ちとか、四8歩打とか)。丁度、入力記号に相当する。

τ : $P \times S \longrightarrow \mathfrak{P}(T)$ admissible な着手の集合。

$\delta : \mathcal{T} \times P \times S \longrightarrow P \times S$ （ただし、(t, p, s)で$t \in \tau(p, s)$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ろう。しかし、実際に問題となるのは、大きな枠組の記述よりも、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よう。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の特性を充分に吟味して）、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よう。しかし、あきらかにMachine dependentになり、一般性がない。又、もう一つの面として、なんでも大体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、苦しいともいえる。
2. 人間の直観により強くうったえるような表現。

今後のsoftwareの進歩の一部は、2.のような表現を最適

コストで処理出来るようなLanguage Processorにかかっているといえよう。ここでは、問題をはっきりさせるために、目的とするLanguageとして、Linear Graph上の種々のManupulationが効果的に取扱えるものを考えよう。これは、単にゲームの記述、処理だけではなく、Finite Mathematics への種々の応用にも強力であると思われる。

一般なList処理の言語として、Machine IndependentぎりぎりのL^6（[4]）があるが、これから述べるIdeaはある意味でL^6のMacro的な拡張であるともいえる。私は、今後、L^6そのものをList処理だけに限らない一種のMachine IndependentなAssemblerに拡張して（実際にImplementするつもり）、それをもってhigh levelなLanguageを記述するのに使おうと思っている。

さて、Linear Graphを扱うLanguageが必要最小限持たねばならぬ機能には次のものがあると思われる。

1. SET NOTATION ……… $\begin{cases} \text{unordered型のset} \\ \text{push-down pop-up型のset.} \end{cases}$
2. NODEやBRANCH上にいろいろなdata structureがあること。
3. NODEやBRANCHのcreation, deletion.
4. あるNODEからのSUCCESSORのLIST UP.

5. DATAに関する基本的な述語及びaction。

他に、記憶容量、時間の点、programmingの楽などから、次のものも加えられるべきであろう。

6. List(ringedでもよい)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の場合にいろいろな節約(内部的)が出来る。

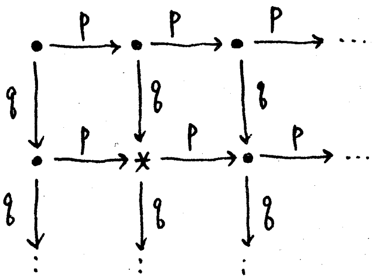
7. BRANCHにcoloringが出来る。

8. NODE, BRANCHにmask(一時的にその存在を無視すること)や、unmaskがいろいろな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もしないで例を出すのは、それがまだIdeaにとどまっているからである。)を列挙して、上述のことを具体化してみよう。記述法はL^6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q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でよい。('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なろう。(‖はcommentの意。~~は省略可能なもの)

[⋁(for π∈set(p, q, rev-p, rev-q)) PRED₁(π, *)]

‖ branchをp, q, rev-p, rev-qにして、各々でPRED₁を験し
‖ て、そのOrをとる。ここで

PRED₁(B, N) ; predicate type(branch, node)

```
v-[ [→BUG1=N]
    [BUG1.='VOID']
    [→BUG1=successor(B: BUG1)]
    [BUG1.='ENEMY']
  L1:[ [BUG1=successor(B: BUG1)]→
      [ [BUG1.='ENEMY']→goto L1,
        [BUG1.='FRIEND']→true,
                          →false],
      →false] ]
```

‖ BUGの意味はL^6のものと同様。→はthenの意。BUG1
‖ の後のperiodは、dataの構造に対応している。L1は
‖ label。recursive callがされている。→の片側になに

‖ も書かなかったのは、locally true の意。locall の意
‖ 味はそれが、それ囲む[ ]の中でしか意味をもたない
‖ ということであり、それに反して、true、false はその
‖ 値でもって、指定されたblockの外へ飛び出す力をもっ
‖ ているのである。

b. Pentomino での例。

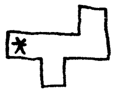
次に、pentomino の便器（T┘ 形のもの、この名称は私が中学時代から愛用しているものにつき断固使用する。ちなみに他の優れた名前を紹介すると、卩は田舎作、凵は曲者、├ は殿、乙は馬の首 etc.）が、盤上におけるかどうかの述語を記してみよう。（反転は、今考えない。）

```
PRED2 ( *, Ð )  :  predicate type (node, branch)
v-[ [→ BUG2 = *]
    [ BUG2. = 'VOID' ]
    [→ BUG2 = successor ( Ð : BUG2 ) ]
    [ BUG2. = 'VOID' ]
    [→ BUG3 = BUG2 ]
    [→ BUG2 = successor ( f1 ( Ð ) : BUG2 ) ]
    [ BUG2. = 'VOID' ]
    [→ BUG3 = successor ( Ð : BUG3 ) ]
```

[BUG3. = 'VOID']

[→ BUG3 = successor ($f_2(D)$: BUG3)]

[BUG3. = 'VOID']]

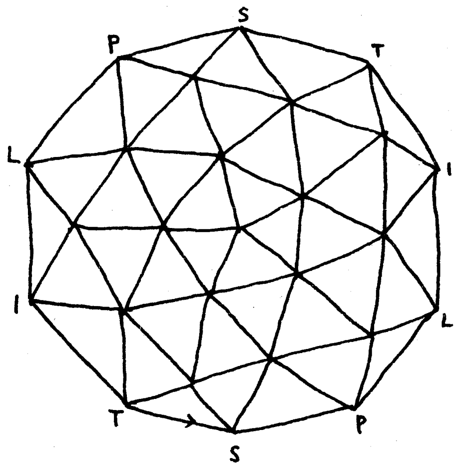
‖  で、Dは direction 位の意。$f_1(D)$ は (p → q,

‖ q → rev-p, rev-p → rev-q, rev-q → p) なる BRANCH から

‖ BRANCH への函数。$f_2 = f_1 \circ f_1 \circ f_1$。

c. Graph の connectedness を調べるような program.

次に SPLIT と呼ばれる Game を紹介する。盤面は、図のようになっていて、Rule は、一手交代で自分の石を置き、先に境界の対称点同志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二個の石を置いてもよい。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 という色付け)

PRED$_3$ (POS$_1$, POS$_2$) : predicate type (node ; node)

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

∨-[[→ N_2 = POS$_1$] [→ init (β)]

L2 : [→ pushdown (β) successor (q ∨ rev-q : N_2) with m(α)]

[→ N_2 = pop-up(β)]

[N_2 = pos_2 → T, → mask prebranch(N_2) in (β) level(α)]

[→ goto L2]]

[[→ maskfree node all level(α)]

[→ maskfree branch all level(α)]]

‖ まず味方の石のない部分を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
‖ 用意されているPushdown Listを初期化する。pos_1のsuc-
‖ cessorでα-levelのmaskのかかっていないものを、β
‖ に書き込む。尚、successorがφであれば、この[ ]の中
‖ の値はlocally falseである。(他にも、この種のpredica-
‖ te付きのactionは多い。) prebranchをmaskするという
‖ のは、このprogramが同じところを堂々巡りさせないた
‖ めである。いい忘れたが、[ ]の前のv-は、それがこの
‖ PREDICATEの値になることを示す。

以上のexampleは、完全なものではない。もう少しいい表現は考えられると思う。私の現在の考えでは、これ等の言語のProcessorはL^6の拡張(言語上、機能上)と、簡単なMacro Processorがあれば何とか処理できるのではないかと思い、非力ながらそれを少し試みてみようと思っている次第である。

Game に話を限って考えても、例えば碁の取り扱いなどは Dynamic な Graph の操作などでかなり最適時間でかつ、理論的に興味のあるものが出来ると思われる。

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)
[redacted] Annual Review in Automatic Programming vol. 6.

[4] K. C. Knowlton : A Programmer's Description of L^6.
Com. A. C. M vol. 9 (616～625) (1966)

最近、次の様なちょっと面白い本が出た。

[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)