ソフトウェア工学　52 — 8
（１９８７・２・１３）

囲碁のルールの代数的記述について

井上　克郎　　鳥居　宏次

大阪大学基礎工学部情報工学科

囲碁プログラムの基礎となるゲームのルールを厳密に規定するために、代数的記述言語ＡＳＬ／１を用いて囲碁ルールを記述してみた。基底データ型及びそれに関する基本関数としては整数、ブール値、そして集合のみを前提とした。そして、現在や過去の盤面、打手、アゲ石等をすべて記憶する抽象的碁盤と呼ぶ’状態’を定義し、その状態が対局者の各一手によってどのように変化するかを記述した。代数的記述の公理は、自然語で書かれたルールの各条文にほぼ対応して作成され、全体で９６個になった。この記述は、比較的容易にかつ形式的に、効率的実行可能な関数型言語や手続き型言語のプログラムに変換可能で、変換されたプログラムは、禁じ手や勝負の判定を厳密にルールに従って行なうルールチェックプログラムになる。同様に囲碁で用いられる種々の用語、戦略、定石等を代数的に記述し、それを効率的に実行可能なプログラムに変換すれば、機械対人間の対局を行なう囲碁プログラム作成の一部となりうる。

Algebraic Specification of the Go Rule

Katsuro Inoue,　　and　Koji Torii

Department of Information and Computer Sciences,
Faculty of Engineering Science, Osaka University
1-1 Machikaneyama, Toyonaka, Osaka 560, Japan

To construct a game program, we have to clarify the rule of the game. In this paper, we show an algebraic specification of the GO game, witten in an algebraic specification language ASL/1. We used integer, boolean, and set as primitive data types, and functions on them as primitive functions. In this specification, we defined an abstract GO board which holds the current and past GO boards, the plays already made, and captured stones. We wrote the changes of the states of the board for each play. In the specification, there are 96 axioms, each of which consists of several lines. This specification can be transformed into an efficient functional or procedural program, which tests legality of each play given to this program and judges the winner at the end of the game. In the same manner, if we would write the specifications of the strategies of GO game and transform it into efficiently executable programs, the progarams obtained would be major parts of a practical GO program.

1．まえがき

ゲームのプログラムを作成するためには、その基本となるゲームのルールが厳密に規定されている必要がある。最近、人間相手に囲碁の対局を行なう囲碁プログラムの作成が試みられているが[8]、その囲碁のルールについては詳しく研究されていない。現在、人間どうしの囲碁の対局ルールについては、歴史的な習慣を成文化した日本棋院の囲碁規約に基づいているが、2．で述べるような理由から、そのままでは囲碁プログラムのルールとして用いづらい。ここでは比較的厳密に記述されており、囲碁プログラムのルールとしてふさわしいと思われた池田が提案するルール[7]を選び、これをより形式的に規定するために代数的仕様記述言語で記述してみた。

用いた仕様記述言語は、代数的記述言語ＡＳＬ／１[1][3]に準じた言語で、基底データ型としては整数、ブール値、そして集合のみを用いて記述した。記述の大きさは数行から成る公理が全部で９６個であった。自然語で書かれたルールの各条文にほぼ対応して、代数的記述の公理を作成していった。ただ、陽には述べられておらず、かつ記述の必要があると思われること(例えば盤面上のすでに石が置いてある点には石を打てない等)は補って記述した。現在や過去の盤面、打手、アゲ石等をすべて記憶する抽象的碁盤と呼ばれる状態を定義し、その状態が対局者が一手打つごとにどのように変化するかを記述する。この方法による記述は、比較的容易かつ形式的に、関数型言語や手続き的言語の効率的実行可能なプログラムに変換可能である。

代数的記述言語で、スタックやソーティング、プロトコル等を記述した例は多く見られるがこの種の問題を記述した例はあまりない。この記述を効率的に実行可能なプログラムに変換することにより、禁じ手や勝負の判定をルールに従って厳密に行なうルールチェッカを得ることができる。また、囲碁は、他のオセロやチェス等のゲームに比べその戦略が複雑であり、単なる思い付きの打ち方や人間が用いる種々の作戦の一部を計算機上に実現したとしてもあまり強いプログラムになるとは思えない。我々は、強い囲碁プログラムを作るには、通常人間が打つ打ち方を整理、分類して、それぞれの作戦に応じた手続きを作成する必要があると考えている。従ってここで試みようとする手法と同様に、分類された作戦、定石等を代数的に記述し、効率の良い（例えば）関数型プログラムに変換することによりそれらの手続きを得ることができる。

2．囲碁ルールについて

現在、日本国内で行なわれている囲碁は、主に日本棋院が昭和２４年に定めた囲碁規約[10]に基づいている（以降現行のルールと呼ぶ）。この現行ルールの他、世界的、歴史的に見れば、中国ルール、台湾ルール、朝鮮ルール、チベットルール等種々のルールがある。また、ルールの研究家たちは、現行のルールの種々の問題点を改善すべく新しい囲碁ルールの提案を行なってきている[9][10]。ここではそれらのなかからゲームとしておもしろいものを選ぶのではなく、囲碁プログラムのルールとしてふさわしく、そして、容易にかつ形式的に代数的仕様記述言語で表せるものを選ぶ。

中国、台湾、朝鮮、チベット等のルールは、陽に成文化したものが見当たらないため用いなかった。また、現行ルールは、過去の慣習をそのまま成文化したため、ゲームの定義のみならず、マナー等不必要なものを含む[9]。さらに、ゲームの基本となる勝負の定義があいまいである。

例えば現行ルールの活（いき）の定義（第３１条）は、「前条第１項の眼を２個以上有し、または交互着手してもこれを２個以上確保することができる形の一連の石は、これを活という。」とある。”眼を２個以上有する”ということは容易に記述できようが、”交互着手してもこれを２個以上確保することができる”の部分は、将来の最良の戦略を前提にしており、容易に記述できないと思われた。また、一般にゲームの進行中、現行ルールの条文に該当しない奇形、珍形が生じる場合があり、これに関しては現行ルールでは以前の判例や日本棋院の判定に頼るようになっているが（第４５，４６条）、これを囲碁プログラムのルールとして取込みにくい。

そこでここでは表２－１に示す池田が提案したルール（池田ルール、または単にルールと呼ぶ）[9]を用いることにする。池田ルールは簡潔に成文化されており、現行ルールのようなあいまい性もないので記述しやすいと思われた。池田ルールの定義１～５及びルール１～３，５，８等は現行のルールと違いない。ルール４は現行ルールにある“却（こう）立て”をより一般化したもので、これにより、あらゆる種類の無限循環を防ぐ。ルール６，７は、石の活死を定義せずに勝敗を決めるために導入されたルールで、これにより終局時に白，黒のそれぞれの地が容易に確定される。

3．代数的記述言語

ここでは代数的言語ＡＳＬ／１[1][3]の記述方法に準じた言語を用いてルールの記述を行った。（ＡＳＬ／１は、代数的言語ＡＳＬ／＊[1][2]における文法及び公理の記述法を具体的に定めた言語である。）一つのテキストは、すでに定義してある仕様を取り込むinclude文、各関数の引数と関数値の型（データタイプ）を「関数名：引数１の型，引数２の型，・・・，引数ｎの型　→　関数値の型（値域）」の形で指定する型指定（ＡＳＬ／１における表現式を指定する生成規則の記述に対応する）、そして項の合同関係を項１＝＝項２で表現する公理の集まりから構成される。

代数的言語で仕様を書く時のプログラミングスタイルとしては、いわゆる”抽象的順序機械”と呼ばれるものを主体とした[1][6]（すべての公理がこの形ではないが、容易に変換しうる）。これは、仕様中で、基底となる型（例えば整数，配列等）以外に一つの型が定義されており（これを抽象的状態、stateと呼ぶ）、次の条件を満たしている。

表２－１　囲碁ルール（池田ルール）

定義１
n路の碁盤　n本の等間隔の平行線と、それに直交するn本の平行線にてなる格子図形をn路の碁盤という。
格子線　線上の直交する平行線を格子線という。
格子点　格子線の交点を格子点という。
空格子点　石の置かれていない格子点を空格子点という。
置石格子点　石の置かれた格子点を置石格子点という。
置石　格子点上におかれた石を置石という。
隣接点　ある格子点の、それを通る格子線上のそれに最も近い格子点を、その格子点の隣接点という。

定義２
打着　石を空格子点上に置くことを打着という。
除去　格子点上の置石を取除くことを除去という。
着手　打着又はパスすることを着手という。
パス　打着することなく、順を次に渡すことをパスという。

定義３
除去される状態　同色の置石の集合が、その置石格子点のすべての隣接点のうち、その置石格子点を除く隣接点のすべてが、他の色の置石で占められている時、前者の置石の集合は除去される状態にあるという。

定義４
形　盤面上に置かれたすべての置石の状態を形という。

定義５
地　空格子点の集合でそのうちのどの格子点から格子線上を次々と移動するも、必ず同色の置石に達するとき、その空格子点の集合をその色の地といい、その置石を囲石という。

ルール１　黒石白石
２組の競技者の一方は黒石を、他の一方は白石を所有する、競技者は自己の所有の石を打着する。

ルール２　交互着手ルール
競技者は交互に着手する。着手せざるときは敗とする。

ルール３　除去のルール
ある石が打着されて、他の色の石の集合が除去される状態となったときは、次の着手の前にそれらの置石の集合は除去される。

ルール４　同形再現打着禁止ルール
ある格子点への打着は、もしそれによって他の色の石を除去する状態にすることができた場合は、それを除去した形が、その対局中にすでに現われた形と同じ形になってはならない。

ルール５　自殺打着禁止ルール
ある格子点への打着は、それによって他の色の石を除去し得る状態にすることができず、しかも自己の石が除去される状態となってはならない。

ルール６　終局
６・１　仮終局　双方のパスが連続したとき仮終局とする。仮終局後も着手を継続する。
６・２　ペナルティ　仮終局後のパスは、そのつどペナルティとして自己の石を一つ相手に渡す。
６・３　終局　仮終局後に双方のパスが連続したとき終局とする。
６・４　同数着手ルール　終局の最後のパスが、仮終局後の先着者が行ったときは、そのパスのみペナルティは不要とする。

ルール７　得点ルール
夫々の地の格子点数より除去された石及びペナルティとして渡した石を減じた数を得点とし、夫々の得点を比較して勝敗を決める。

ルール８　条件設定ルール
先着の条件（手割）、勝敗の規定（コミ）は別途にこれを設定するものとする。

（１）引数にstateを二つ以上もつ関数記号は存在しない。便宜上、stateは関数の第１引数に現われるとする。
（２）公理の左辺の形は$f(x_1,x_2,\cdots)$または$f'(g(x_1\cdots x_1),x_{1+1},\cdots)$である。ただし、gの値域はstateである（fの第１引数の定義域はstateであってもよい）。また変数x_1，・・・はすべて異なる。
（３）同じ左辺をもつ公理は複数個存在しない。また、左辺が$f(g(x_1,\cdots),\cdots)$の形の公理がある時、左辺が$f(x_1,\cdots)$や$g(x_1,\cdots)$の形の公理はない（公理系が重なりをもたない）。

抽象的順序機械の形の記述は、関数型言語（例えばＡＳＬ／＊の具体的な部分言語であるＡＳＬ／Ｆ[7]など）や手続き的言語の効率的なプログラムに比較的容易に変換可能である[6]。

４．ルールの記述

４．１　データタイプ

ここでは、池田ルールを記述するために用いるデータ型について述べる。基底データ型としては整数（integer）（基本関数としては、＋，－，＝，＞等を用いる）、ブール（boolean）（&，or）、及び集合（set）（∪，－（差集合）、∈，∈，｜　｜（要素数））のみを用いる。これらの仕様の記述はinclude文によって取込まれる。記述中では、color，grid，play，board，そして　state　を定義している。

（１）color：これは、碁盤上の一つの点（縦横の平行線の交点、池田ルールでは格子点と呼ばれる）の状態を示すデータ型で、’黒’、’白’、’空’、そして’外’の４つのいずれかの値を持つ。池田ルールでは陽に記述されていないが、盤面内の各点が、黒石が置かれている（黒）、白石が置かれている（白）、なにも置かれていない（空）のいずれかの状態であることを、公理1-1で表す。また、ここでは後の記述の都合上、盤面を、実際ゲームの行われるｎ×ｎから無限の平面に拡張しておく。そしてゲームにとっては不必要な拡張された部分の各点は外という値を常に持つとする。関数点の色は（後に述べる）抽象的碁盤(state)と点の指定(grid)の二つを引数として持ち、現在の碁盤上の１つの点の状態を返す。公理1-2は、ゲームの開始時には盤面内の各点は空、それ以外の点は外であることを表す（ハンディをつけるための置石はないものとしている。必要ならば容易に記述できる）。
（２）grid：盤面上の一つの点を指定するデータ型である。盤面を一つの二次元座標系と見て、ｘ座標とｙ座標を指定する二つの整数の組である（公理1-4,1-5）。関数座標化は２つの整数をx座標、y座標として１つのgrid型の値を作る。関数座標Ｘ，座標Ｙはgrid型の値よりそれぞれｘ座標，ｙ座標を取出す。
（３）play：対局者が行なう着手（池田ルール定義２）を表すデータ型。値として、打着せずパスを行うことを表す’パスする’、そして、実際にある点に打着を行うこと

表４－１　囲碁ルールの代数的記述（その１）

```
ＴＥＸＴ　囲碁ルール；

include  integer；
include  boolean；
include  set(grid)；
include  set(color)；

＃１．開始盤面
    点の色：state, grid  →  color；
    開始状態  :        →  state；
    黒，白，空，外：      →  color；
1-1: {盤面内か？(xy)  &  点の色（Ｓ，xy）＝黒}   or
     {盤面内か？(xy)  &  点の色（Ｓ，xy）＝白}   or
     {盤面内か？(xy)  &  点の色（Ｓ，xy）＝空}   or
   {not（盤面内か？(xy)) & 点の色（Ｓ，xy）＝外}
      ＝＝真；
1-2:点の色（開始状態，xy）＝＝
      if  盤面内か？(xy)  then  空
      else  外  ；

    盤面内か？：grid  →  boolean；
    最大の線ｎ：      →  integer；
1-3:盤面内か？（xy）＝＝
      座標Ｘ(xy)≧１  &  座標Ｘ(xy)≦最大の線ｎ；
      &  座標Ｙ(xy)≧１  &  座標Ｙ(xy)≦最大の線ｎ；

    座標化：integer, integer  →  grid；
    座標Ｘ  ：grid  →  integer；
    座標Ｙ  ：grid  →  integer；
1-4:座標Ｘ（座標化(x,y)）＝＝ｘ；
1-5:座標Ｙ（座標化(x,y)）＝＝ｙ；

＃２．着手
    パスか？：play  →  boolean；
    パスする：      →  play；
    打着    ：grid  →  play；
    点      ：play  →  grid；
2-1:点（打着（xy））＝＝ xy  ；
2-2:パスか？（パスする）＝＝  真；
2-3:パスか？（打着(xy)）＝＝  偽；

＃３．交互着手ルール
    不正着手か？：state  →  boolean；
3-1:不正着手か？（白番（開始状態，TE））＝＝真；
3-2:不正着手か？（白番（白番（Ｓ，TE1），TE2））＝＝
      真；
3-3:不正着手か？（黒番（黒番（Ｓ，TE1），TE2））＝＝
      真；
3-4:不正着手か？（黒番（開始状態，TE））＝＝偽；
3-5:不正着手か？（黒番（白番（Ｓ，TE1），TE2））＝＝
      偽；
3-6:不正着手か？（白番（黒番（Ｓ，TE1），TE2））＝＝
      偽；

＃４．打石
    黒番：state, play  →  state；
4-1:点の色（黒番（Ｓ，パスする），xy）＝＝
      点の色（Ｓ，xy）；
4-2:点の色（黒番（Ｓ，打着（xy1）），xy）＝＝
      if  xy1＝xy  then  黒
      else  if  白が除去される状態か？（Ｓ，xy1，xy）
            then 空
      else  点の色（Ｓ，xy）；

    白番：state, play  →  state；
4-1':点の色（白番（Ｓ，パスする），xy）＝＝
      点の色（Ｓ，xy）；
4-2':点の色（白番（Ｓ，打着（xy2）），xy）＝＝
      if  xy2＝xy  then  白
      else  if  黒が除去される状態か？（Ｓ，xy2，xy）
            then 空
      else  点の色（Ｓ，xy）；
```

を表す値（関数打着によってgrid型から得られる）のいずれかである。関数点は逆にplay型の値からgrid型の値を得る。

（４）state：ゲーム開始から現在までのすべての着手と（拡張された）盤面、アゲ石、ペナルティ石等を覚える抽象的碁盤。過去の状態を覚えておくのは、たとえば同形再現打着禁止ルール（ルール４）により、ある打着によって得られた盤面がゲーム開始時から今まで現われたいずれの盤面とも等しくないことを調べる必要があるからである。関数点の色はこのstateから各点の現在の状態を与える。また終局時に勝負の判定に必要な白、黒のおのおののアゲ石の数やペナルティの数を取出す関数アゲ石白、アゲ石黒や、ペナルティ白、ペナルティ黒等を用意した。

４・２　各ルールの記述

池田ルールの定義１，２にある碁盤、打着、着手及びパスは、すでに公理1-1～2-3を用いて記述した。その他の定義は、必要に応じて記述する。池田ルールの各ルールを以下のように表す。

（１）ルール１　白石黒石

黒石・白石の所有については陽に記述しない。黒の着手及び白の着手に対応して、抽象的碁盤（state）の状態をそれぞれ変化させ状態遷移関数黒番と白番を用意する。これらの関数は、第１引数として抽象的碁盤、第２引数として着手する手（play型で、パスか打着のいずれか）を持つ。この二つの関数を一つにまとめ、着手が黒である、または白であるということをパラメータ化することも可能であろうが、ここでは記述の読み易さを考慮し、分離した。以降、黒にとって必要な公理を先に記述し、それに対応する白の公理の番号には ’ を付す。

（２）ルール２　交互着手ルール

不正な着手であることを述語関数不正着手か？によって表す(3-1～3-6)。ルールでは陽に述べられていないが、先着者は常に黒であると仮定した。黒白黒白・・・の順に着手が進むかぎりこの述語は偽であるが、・・・白白・・・、または・・・黒黒・・・の順の着手が行われると真になる。ルールにある”着手せざる時”とは”ある一定時間内に着手せざる時”等の意味であろうと推察されるが、この文だけでは明らかではなかったので、ここでは記述しなかった。

（３）ルール３　除去のルール

公理4-1～4-2’は、ルールでは陽に述べていない以下のことを記述している。すなわち黒番で（または白番で）パスを行うと、各点の状態は前と同じ状態である（4-1～4-1’）。また、ある点に黒（白）を打つような着手を行うと盤面の状態は黒（白）に変わる。そしてそれ以外の点で、黒（白）を打つことにより除去される状態になる白（黒）の点は空になる。それ以外は前と同じ状態である（4-2～4-2’）。

述語白が除去される状態か？は、ある盤面で（第１引数で指定）、黒をある点（第２引数）に打つことによってあ

表４－２　囲碁ルールの代数的記述（その２）

```
＃５．除去のルール
    白が除去される状態か？：
                state, grid, grid  →  boolean；
5-1:白が除去される状態か？（S，xy1,xy2)＝＝
    点の色（S，xy2）＝白　&
    白が包囲されたか？（黒を置く（S，xy1），xy2）；

    黒を置く：state, grid  →  state;
5-2:点の色（黒を置く（S，xy1），xy）＝＝
      if  xy1=xy  then  黒
      else  点の色（S，xy）　；

    白が包囲されたか？：state, grid  →  boolean；
5-3:白が包囲されたか？（S，xy2）＝＝
    空∈石の種類（S，
      隣接点の集合（S,白のグループ(S,xy2))
          －　白のグループ(S,xy2)　）　&
    黒∈石の種類（S，
      隣接点の集合（S,白のグループ(S,xy2))
          －　白のグループ(S,xy2)　）

    石の種類：state, grid_set  →  color_set；
5-4:石の種類（S，φ）＝＝φ；
5-5:石の種類（S，{xy}∪X）＝＝
      点の色（S，xy)　∪　石の種類（S，X）；

    隣接点の集合：state, grid_set  →  grid_set；
5-6:隣接点の集合（S，φ）＝＝φ；
5-7:隣接点の集合（S，{xy}∪X）＝＝
      {上（xy)}∪{下(xy)}∪{左(xy)}∪{右(xy)}
      ∪　隣接点の集合（X）

    上，下，左，右：grid  →  grid；
5-8:上(xy)＝＝座標化（座標X(xy)，座標Y(xy)＋1）；
5-9:下(xy)＝＝座標化（座標X(xy)，座標Y(xy)－1）；
5-10:左(xy)＝＝座標化（座標X(xy)－1，座標Y(xy)）；
5-11:右(xy)＝＝座標化（座標X(xy)＋1，座標Y(xy)）；

    白のグループ：state, grid  →  grid_set；
5-12:白のグループ（S，xy2）＝＝
       白のグループ１（S，xy2，φ）；

    白のグループ１：
            state, grid, grid_set  →  grid_set；
5-13:白のグループ１（S，xy，X）＝＝
      if  点の色（S，xy）≠白　or　xy∈X　then　X
      else
      白のグループ１（S，上(xy)，
        白のグループ１（S，下(xy)，
          白のグループ１（S，左(xy)，
            白のグループ１（S，右(xy),{xy}∪X））））；


    黒が除去される状態か？：
                state, grid, grid  →  boolean；
5-1':  黒が除去される状態か？(S，xy2,xy1)＝＝
    点の色（S，xy1）＝黒　&
    黒が包囲されたか？(白を置く（S，xy2）,xy1）；

    白を置く：state, grid  →  state;
5-2':点の色（白を置く（S，xy2），xy）＝＝
      if  xy2=xy  then  白
      else  点の色（S，xy）　；

    黒が包囲されたか？：state, grid  →  boolean；
5-3':黒が包囲されたか？（S，xy1）＝＝
    空∈石の種類（S，
      隣接点の集合（S,黒のグループ(S,xy2))
          －　黒のグループ(S,xy2)　）　&
    白∈石の種類（S，
      隣接点の集合（S,黒のグループ(S,xy2))
          －　黒のグループ(S,xy2)　）

    黒のグループ：state, grid  →  grid_set；
5-12':黒のグループ（S，xy）＝＝
      黒のグループ１（S，xy，φ）；

    黒のグループ１：
            state, grid, grid_set  →  grid_set；
5-13':黒のグループ１（S，xy，X）＝＝
      if  点の色（S，xy）≠黒　or　xy∈X　then　X
      else
      黒のグループ１（S，上(xy)，
        黒のグループ１（S，下(xy)，
          黒のグループ１（S，左(xy)，
            黒のグループ１（S，右(xy),{xy}∪X））））；

＃６．正しい手
    正しい黒の手か？：state,play  →  boolean；
    正しい白の手か？：state,play  →  boolean；
6-1:正しい黒の手か？（S，パスする）＝＝　真
6-2:正しい黒の手か？（S，打着（xy））＝＝
      盤面内か？(xy)　&
      点の色（S，xy）＝空　&
      not（黒の禁じ手か？（S，xy））；

6-1':正しい白の手か？（S，パスする）＝＝　真
6-2':正しい白の手か？（S，打着(xy)）＝＝
      盤面内か？(xy)　&
      点の色（S，xy）＝空　&
      not（白の禁じ手か？（S，xy））；

    黒の禁じ手か？：state, grid  →  boolean；
    白の禁じ手か？：state, grid  →  boolean；
6-3:黒の禁じ手か？（S，xy1）＝＝
      黒同形再現か？（S，xy1）　or
      黒自殺か？（S，xy1）；
6-3':白の禁じ手か？（S，xy2）＝＝
      白同形再現か？（S，xy2）　or
      白自殺か？（S，xy2）；
```

る点の白石（第３引数）が除去されるかを示す（5-1～5-13'）。これを調べる方法としては、ルールの定義３と同様に、

①第２引数の黒石を盤面においてみる（関数黒を置く，5-1、5-2）。

②着目する第３引数で指定された白石のグループを求める。このグループとは着目する白石から上下左右に隣接する白石を順次たどって到達できる白石の最大の集合をいう（5-12、5-13）。

③白のグループ中の各白石の隣接点の集合を求める（5-6～5-11）。

④　隣接点の集合－白のグループ　の各点に空はなく、かつ少なくとも一つは黒があれば除去される状態である（5-3）（白の最大の集合を求めているので、隣接点の集合－白のグループ　には常に白は含まれない）。

（４）ルール４，ルール５　禁じ手

ルール４及びルール５では、いわゆる禁じ手について述べている。ここではそれら禁じ手を含め、ある黒番において着手しようとする手が許されるかの述語正しい黒の手か？を設ける（6-1,6-2．白に対しても同様に6-1',6-2'）。ある抽象的碁盤Ｓにおける黒の着手ｐに対し、もし、正しい黒の手か？（Ｓ，ｐ）が偽になるならばｐは許されない着手である。関数黒番の第２引数に来る着手すべき手は、常にこの述語を真にするもののみであると仮定している。

公理6-1ではパスは常に許される手であることを述べて

表４－３　囲碁ルールの代数的記述（その３）

```
＃７．同形再現打着禁止ルール
    黒同形再現か？：state, grid  →  boolean；
    白同形再現か？：state, grid  →  boolean；
7-1:黒同形再現か？（S，xy）＝＝
      過去と同形か？（S，盤（黒番１（S，xy）））；
7-1':白同形再現か？（S，xy）＝＝
      過去と同形か？（S，盤（白番１（S，xy）））；

    黒番１：state, grid  →  state；
    白番１：state, grid  →  state；
7-2:点の色（黒番１（S，xy1），xy）＝＝
      if  xy1＝xy  then  黒
      else  if  白が除去される状態か？（S，xy1，xy)
            then  空
      else  点の色（S，xy）；
7-2':  点の色（白番１（S，xy2），xy）＝＝
      if  xy2＝xy  then  白
      else  if  黒が除去される状態か？（S，xy2，xy)
            then  空
      else  点の色（S，xy）；

    盤：state  →  board；
    点の色１：board, grid  →  color；
    初期盤面：          →  board；
7-3:点の色１（盤（S），xy）＝＝点の色（S，xy）；
7-4:点の色１（初期盤面，xy）＝＝
      if  盤面内か？(xy)  then  空
      else  外；

    過去と同形か？：state, board  →  boolean；
7-5:過去と同形か？（S，bd）＝＝
      if  盤（S）＝bd  then  真
      else  if  盤（S）＝初期盤面  then  偽
      else  過去と同形か？（前の状態（S），bd）；

    前の状態：state  →  state；
7-6:   前の状態（黒番（S，TE））＝＝S；
7-7:   前の状態（白番（S，TE））＝＝S；

＃８．自殺打着禁止ルール
    黒自殺か？：state, grid  →  boolean；
    白自殺か？：state, grid  →  boolean；
8-1:黒自殺か？（S，xy1）＝＝
      not（白を抜くか？（S，xy1））  &
      黒が包囲されたか？（黒を置く（S，xy1),xy1）；
8-1':白自殺か？（S，xy2）＝＝
      not（黒を抜くか？（S，xy2））  &
      白が包囲されたか？（白を置く（S，xy2),xy2);

    白を抜くか？：state, grid  →  boolean；
    黒を抜くか？：state, grid  →  boolean；
8-2:白を抜くか？（S，xy1）＝＝
      {点の色（S，上(xy1)）＝白  &
       白が包囲されたか？（黒を置く（S，xy1），
                                  上(xy1)）}
   or  {点の色（S，下(xy1)）＝白  &
       白が包囲されたか？（黒を置く（S，xy1），
                                  下(xy1)）}
   or  {点の色（S，右(xy1)）＝白  &
       白が包囲されたか？（黒を置く（S，xy1），
                                  左(xy1)）}
   or  {点の色（S，左(xy1)）＝白  &
       白が包囲されたか？（黒を置く（S，xy1），
                                  右(xy1)）}；
8-2':黒を抜くか？（S，xy2）＝＝
      {点の色（S，上(xy2)）＝黒  &
       黒が包囲されたか？（白を置く（S，xy2），
                                  上(xy2)）}
   or  {点の色（S，下(xy2)）＝黒  &
       黒が包囲されたか？（白を置く（S，xy2），
                                  下(xy2)）}
   or  {点の色（S，右(xy2)）＝黒  &
       黒が包囲されたか？（白を置く（S，xy2），
                                  左(xy2)）}
   or  {点の色（S，左(xy2)）＝黒  &
       黒が包囲されたか？（白を置く（S，xy2），
                                  右(xy2)）}；

＃９．仮終局
    仮終局か？：state  →  boolean；
9-1:仮終局か？（開始状態）＝＝偽；
9-2:仮終局か？（黒番（S，TE））＝＝
      if  仮終局か？（S）  or
       {パスか？(TE)  &  前の手（S）＝パスする}
       then  真
      else  偽；
9-2':仮終局か？（白番（S，TE））＝＝
      if  仮終局か？（S）  or
        {パスか？(TE)  &  前の手（S）＝パスする}
       then  真
      else  偽；

    前の手：state  →  play；
    無し：        →  play；
9-3:前の手（開始状態）＝＝無し；
9-4:前の手（黒番（S，TE））＝＝TE；
9-4':前の手（白番（S，TE））＝＝TE；

＃１０．  終局
    終局か？：state  →  boolean；
10-1:終局か？（開始状態）＝＝偽；
10-2:終局か？（黒番（S，TE））＝＝
      if  仮終局か？（S）  &  パスか？(TE)
          &  前の手（S）＝パスする  then  真
      else  偽；
10-2': 終局か？（白番（S，TE））＝＝
      if  仮終局か？（S）  &  パスか？(TE)
          &  前の手（S）＝パスする  then  真
      else  偽；

＃１１．ペナルティ
    ペナルティ白：state  →  integer；
    ペナルティ黒：state  →  integer；
11-1:ペナルティ白（開始状態）＝＝０；
11-2:ペナルティ白（白番（S，パスする））＝＝
      if not（仮終局か？（S））  then  0
      else if not（終局か？（白番（S，パスする)))
           then  アゲ石白（S）＋１
      else if 仮終局後先着（S）＝白
           then  アゲ石白（S）
      else アゲ石白（S）＋１；
11-3:ペナルティ白（白番（S，打着(xy2)））＝＝
        ペナルティ白（S）；
11-4:ペナルティ白（黒番（S，TE））＝＝
        ペナルティ白（S）；
11-1':ペナルティ黒（開始状態）＝＝０；
11-2':ペナルティ黒（黒番（S，パスする））＝＝
      if  not（仮終局か？（S））  then  0
      else  if  not（終局か？（黒番（S，パスする)))
           then  アゲ石黒（S）＋１
      else  if  仮終局後先着（S）＝黒
           then  アゲ石黒（S）
      else  アゲ石黒（S）＋１；
11-3':ペナルティ黒（黒番（S，打着(xy2)））＝＝
        ペナルティ黒（S）；
11-4':ペナルティ黒（白番（S，TE））＝＝
        ペナルティ黒（S）；

    仮終局後先着：state  →  color；
11-5:仮終局後先着（黒番（S，TE））＝＝
      if  仮終局か？（S）  then
          if  not（仮終局か？（前の状態（S））
              then  黒
              else  仮終局後先着（S）；
11-5':仮終局後先着（白番（S，TE））＝＝
      if  仮終局か？（S）  then
          if  not（仮終局か？（前の状態（S））
              then  白
              else  仮終局後先着（S）；
```

表４－４　囲碁ルールの代数的記述（その４）

```
＃１２．勝敗ルール
    黒の勝ちか？：state  →  boolean；
    白の勝ちか？：state  →  boolean；
12-1:黒の勝ちか？（Ｓ）＝＝終局か？（Ｓ）＆
     ｛（黒の得点（Ｓ）－コミ）＞（白の得点（Ｓ））｝
12-1':白の勝ちか？（Ｓ）＝＝終局か？（Ｓ）＆
     ｛（白の得点（Ｓ）＋コミ）＞（黒の得点（Ｓ））｝

    コミ    :       →  integer；
    黒の得点：state  →  integer；
    白の得点：state  →  integer；
12-2:黒の得点（Ｓ）＝＝
    黒地の数（Ｓ）－アゲ石黒（Ｓ）－ペナルティ黒（Ｓ）
12-2'白の得点（Ｓ）＝＝
    白地の数（Ｓ）－アゲ石白（Ｓ）－ペナルティ白（Ｓ）

    黒地の数：state  →  integer；
    白地の数：state  →  integer；
12-3: 黒地の数（Ｓ）＝＝｜黒地（Ｓ）｜；
12-3':白地の数（Ｓ）＝＝｜白地（Ｓ）｜；
    黒地：state  →  grid_set；
    白地：state  →  grid_set；
12-4: xy∈黒地（Ｓ）＝＝
       点の色（Ｓ，xy）＝空　＆　黒地か？（Ｓ，xy）；
12-4':xy∈白地（Ｓ）＝＝
       点の色（Ｓ，xy）＝空　＆　白地か？（Ｓ，xy）；

    黒地か？：state，grid  →  boolean；
    白地か？：state，grid  →  boolean；
12-5:黒地か？（Ｓ，xy）＝＝
      白∈石の種類（Ｓ，
          隣接点の集合（Ｓ，空のグループ(Ｓ,xy)）
      －  空のグループ(Ｓ,xy)  ）                ＆
      黒∈石の種類（Ｓ，
          隣接点の集合（Ｓ，空のグループ(Ｓ,xy)）
      －  空のグループ(Ｓ,xy))  ）                ；
12-5':白地か？（Ｓ，xy）＝＝
      黒∈石の種類（Ｓ，隣接点の集合（Ｓ，
                          空のグループ(Ｓ,xy)））  ＆
      白∈石の種類（Ｓ，隣接点の集合（Ｓ，
                          空グループ(Ｓ,xy)））；
    空のグループ：state，grid  →  grid_set；
    空のグループ１：
          state，grid，grid_set  →  grid_set；
12-6:空のグループ（Ｓ，xy）＝＝
      空のグループ１（Ｓ，xy，φ）；
12-7:空のグループ１（Ｓ，xy，Ｘ）＝＝
     if  点の色（Ｓ，xy）≠空  or  xy∈Ｘ  then  Ｘ
     else
     空のグループ１（Ｓ，上(xy),
       空のグループ１（Ｓ，下(xy),
         空のグループ１（Ｓ，左(xy),
           空のグループ１（Ｓ，右(xy),{xy}∪Ｘ））））；

    アゲ石白：state  →  integer；
    アゲ石黒：state  →  integer；
12-8:アゲ石黒（開始状態）＝＝０；
12-9:アゲ石黒（白番（Ｓ，打着（xy2））＝＝
     if  黒を抜くか？（Ｓ，xy2)  then
       ｜囲まれる黒石（Ｓ，xy2））｜＋アゲ石黒（Ｓ）
     else  アゲ石黒（Ｓ）；
12-10:アゲ石黒（白番（Ｓ，パスする））＝＝
       アゲ石黒（Ｓ）；
12-11:アゲ石黒（黒番（Ｓ，TE））＝＝アゲ石黒（Ｓ）；
12-8':アゲ石白（開始状態）＝＝０；
12-9':アゲ石白（黒番（Ｓ，打着（xy1））＝＝
     if  白を抜くか？（Ｓ，xy1)  then
       ｜囲まれる白石（Ｓ，xy1））｜＋アゲ石白（Ｓ）

     else  アゲ石白（Ｓ）；
12-10':アゲ石白（黒番（Ｓ，パスする））＝＝
       アゲ石白（Ｓ）；
12-11':アゲ石白（白番（Ｓ，TE））＝＝アゲ石白（Ｓ）；

    囲まれる黒石：state，grid  →  grid_set；
    囲まれる白石：state，grid  →  grid_set；
12-12:囲まれる黒石（Ｓ，xy2）＝＝
      [if  黒が除去される状態か？（Ｓ，xy2，上(xy2))
        then  黒のグループ（Ｓ，上(xy2))
       else  φ  ]
   ∪  [if  黒が除去される状態か？（Ｓ，xy2，下(xy2))
        then  黒のグループ（Ｓ，上(xy2))
       else  φ  ]
   ∪  [if  黒が除去される状態か？（Ｓ，xy2，左(xy2))
        then  黒のグループ（Ｓ，上(xy2))
       else  φ  ]
   ∪  [if  黒が除去される状態か？（Ｓ，xy2，右(xy2))
        then  黒のグループ（Ｓ，上(xy2))
       else  φ  ]；
12-12':囲まれる白石（Ｓ，xy1）＝＝
      [if  白が除去される状態か？（Ｓ，xy1，上(xy1))
        then  白のグループ（Ｓ，上(xy1))
       else  φ  ]
   ∪  [if  白が除去される状態か？（Ｓ，xy1，下(xy1))
        then  白のグループ（Ｓ，上(xy1))
       else  φ  ]
   ∪  [if  白が除去される状態か？（Ｓ，xy1，左(xy1))
        then  白のグループ（Ｓ，上(xy1))
       else  φ  ]
   ∪  [if  白が除去される状態か？（Ｓ，xy1，右(xy1))
        then  白のグループ（Ｓ，上(xy1))
       else  φ  ]；
```

いる。6-2ではある点xyへの打着は、もしそれがゲームが行なわれる盤面内で、かつその点が空であり、その打着が禁じ手でないならば許される手であることを表している。6-3は禁じ手が、同形再現打着禁止ルール及び自殺打着禁止ルールからなることを述べている。

公理7-1～7-7は同形打着禁止ルールを記述している。まず、注目する黒（または白）の打着xyを行なってみたとする（関数黒番１及び白番１がそれを行う。7-2,7-2'）。そして得られた盤面全体を取出し（関数盤、7-3,7-4）すでに現われた盤面と比較する（7-5）。過去の盤面は抽象的碁盤（state）に関数前の状態を順次ほどこすことにより得る（7-6,7-7）。

自殺打着禁止ルールは公理8-1～8-2'で記述される。すなわち相手の石を抜くような打着（関数白を抜くか？，黒を抜くか？で調べる）でなく、かつ自分自身が打つことによって相手の石に囲まれる（黒が包囲されたか？，白が包囲されたか？で調べる）場合、述語黒自殺か？、白自殺か？が真になる（8-1,8-1'）。ある打着により相手の石を抜くかどうかは、その打着の隣接点にある相手の石が囲まれたかどうかによって判定する（8-2,8-2'）。

（5）ルール6　仮終局、終局

ルール6・1及び6・3で述べる仮終局、終局に対応して、述語仮終局か？　及び、終局か？をそれぞれ設ける（9-1～10-2'）。対局者がお互い連続してパスしたかを知るために、相手の前の手を抽象的碁盤から得るための関数前の手を用いた（9-3～9-4'）。仮終局か？は、一度真になると、その値を保持しつづける。終局か？はいったん真になった場合、着手は停止するものとし、その後については考慮しない。

ルール6・2及び6・4で述べるペナルティに関しては公理11-1～11-5'で記述する。

（6）ルール7，8　勝敗

公理12-1～12-12は、終局時における勝敗を定義する。12-4は黒地の点の集合を与える関数黒地を定義しており、その集合に属する点xyは、その色が空でありかつその囲んでいる周囲の石に黒があり、白を含まないことを述べている。12-7～12-11'は黒白それぞれのアゲ石数を定義している。

5．あとがき

囲碁ルールの代数的記述言語による記述について述べた。この記述を作成するのに、この言語及び記述方法に不慣れな人間一人で約3週間を要した。現在この記述を関数型言語ＡＳＬ／Ｆプログラムに変換作業中である。この変換されたプログラムを実行すると、人間どうしの対局中の禁じ手の判定や、終局時の勝負の判定を行なういわゆるコンピュータ碁盤となる予定である。一方ルールのみならず、囲碁で用いられる種々の用語、戦略、定石等を同様に代数的に記述することも試みている。例えば表5-1は、xyへの黒打ちは白石を切るかどうかの述語を記述している。これらの記述は同様に効率的に実行可能な関数型プログラム等にほぼ機械的に変換することができるので、機械対人間の対局を行なう囲碁プログラムの一部が得られることのなる。

［謝辞］

ＡＳＬ／１の記述に関し助言を頂いた杉山裕二氏、東野輝夫氏、関浩之氏に感謝します。囲碁に関する種々の助言をいただいた山住巌氏に深謝します。また、本原稿作成に協力して頂いた林明宏氏に感謝します。

［参考文献］

(1)　嵩、谷口、杉山　”代数的言語の設計と処理系”、榎本編、ソフトウェア工学ハンドブック、オーム社，93-123 (1986-6).

(2)　嵩、谷口、杉山、関　”代数的言語ＡＳＬ／*、意味定義を中心に”、信学論(D)、J69-D7，pp.1066-1073(1986-7).

(3)”ＡＳＬ／１仕様書”、内部メモ.

(4)　田中、伊藤、松浦、谷口　”研究室内図書購入手続きの代数的仕様記述とその実現”、情報処理学会ソフトウェア工学研究会46-3，pp.17-24(1986-2).

(5)　八木、関、谷口、嵩　”関数型言語ＡＳＬ／Ｆコンパイラの代数的記述とその詳細化”、信学技報AL85-67，pp.81-92 (1986-1).

(6)　K. Torii, Y. Morisawa, Y. Sugiyama, T. Kasami "Functional Programming and Logical Programming for the Telegram Analysis Problem", Proc. of 7th International Conference on Software Engineering, pp.463-471 (1984-3).

(7)　K. Inoue, H. Seki, K. Taniguchi, T. Kasami "Compiling and Optimizing Methods for the Function-al Language ASL/F", Science of Computer Programming, 7, pp.297-312 (1986-11).

(8)　B. Wilcox "Reflection on Building Two Go Programs", ACM SIGART,94,pp.29-43 (1985-10).

(9)　池田敏雄　”囲碁ルールについて１～１１”、囲碁新潮、昭和43年9月号～昭和44年11月号　(1968-9～1969-11).

(10)　林裕　”囲碁百科辞典”、金園社、(1965-5).

表５－１　“切る”の代数的記述

```
        黒キリか？：state　－＞　boolean；

  黒キリか？（黒番（Ｓ，xy））＝＝
not　（黒トリか？（Ｓ，xy））　＆
not　（黒アテか？（黒番（Ｓ，xy））　＆
[｛点の色（Ｓ，左上(xy)）＝黒
　＆　点の色（Ｓ，左(xy)）＝白
　　＆　点の色（Ｓ，上(xy)）＝白｝　or
 ｛点の色（Ｓ，右上(xy)）＝黒
　＆　点の色（Ｓ，右(xy)）＝白
　　＆　点の色（Ｓ，上(xy)）＝白｝　or
 ｛点の色（Ｓ，左下(xy)）＝黒
　＆　点の色（Ｓ，左(xy)）＝白
　　＆　点の色（Ｓ，下(xy)）＝白｝　or
 ｛点の色（Ｓ，右下(xy)）＝黒
　＆　点の色（Ｓ，右(xy)）＝白
　　＆　点の色（Ｓ，下(xy)）＝白｝］；
    ・
    ・
    ・
    ・
```