論　文

# 囲碁ルールの形式化とその実行プログラムへの変換

正　員　井上　克郎[†]　　　正　員　鳥居　宏次[†]

# Formalization of GO Rule and Its Transformation to an Executable Program

Katsuro INOUE[†] *and* Koji TORII[†], *Members*

**あらまし**　ゲームのプログラムを作成する際，その基礎となるルールが厳密に規定されている必要がある．本研究では，代数的記述言語を用いて囲碁ルールを記述し，次にそれを関数型プログラムに変換した．まず，現在の盤面，打ち手，アゲ石等の要素から構成される状態を考え，ゲームの進行を，入力される打つ手によって起こる状態遷移の繰返しであると思う．そして，①初期状態，②状態遷移，③入力として許される正しい手，④最終状態，および⑤勝者，の五つを与える関数や定数を代数的記述言語 ASL/1 を用いて定義することによってルールの記述を行う．ASL/1 の仕様の記述スタイルとしては，効率的な関数型プログラムへ，機械的に変換可能な抽象的順序機械（ASM）を用いた．ここでは，ASL/1 で記述された囲碁のルールの仕様を，ASL/1 の部分言語である ASL/F のプログラムに変換した．得られたプログラムは，盤面の更新や禁じ手の判定等を記述されたルールに厳密に従って行う．これにいくつかの入出力関数を追加して，現在の盤面を表示したり，禁じ手や勝敗の判定を厳密に行う碁盤プログラムを作成した．

## 1. まえがき

ゲームを行う際，その基本となるルールが厳密に定まっている必要がある．ルールが厳密に決まっていないと，対戦者間で問題が生じたり，また勝つための作戦や手順を正しく考えることができない．人間相手にゲームの対局を行うゲームプログラムでゲームを行う場合も，対戦者の人間とゲームプログラム双方が，あいまいなく定義された同一のルールに基づいている必要がある．しかし，ゲームのルールを形式的に定義する方法については，知られていない．

本論文では，ある囲碁ルールの例を用いて，ゲームのルールを代数的記述言語で形式的に記述し，それを実行可能なプログラムに変換する方法について述べる．今までに，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ない．

変換されたプログラムは厳密に元の代数的仕様に基づいており，これをそのままゲームプログラムの中に組み込んで，厳密にルール検査を行うルーチンとして用いることができる．また，ゲームプログラムを実用的に十分強いレベルにするには，種々の戦略や手順を整理して組み込む必要がある．その際，ルールがプログラムの一部に整理された形で組み込まれていると，戦略や手順の良否の判定などに利用でき都合がよいと思われる．

記述の対象となる囲碁のルールとしては，日本国内の囲碁の対局ルールである日本棋院の囲碁規約[1]を用いることが考えられたが，これは過去の慣習をそのまま成文化したため，条文中にゲームの定義のみならず，マナーなど仕様化に不必要なものを含む．更に，例えば石の生き死に等の一部の定義があいまいである[2],[3]．従ってここでは，ルールを形式的に記述し，実行可能なプログラムへ変換を試みるという立場から，比較的簡潔かつ厳密に記述されている，池田が提案するルール[2],[3]を選んだ(以後，池田ルールまたは単にルールと呼ぶ)．

池田ルールは，8項の定義と8条のルールの記述で簡潔に構成されている．主な特徴としては，

①　コウだての規則をより一般化し，既に対局中に現れた形と同じ形になる打着を禁止する（同形再現禁止

† 大阪大学基礎工学部情報工学科，豊中市
Faculty of Engineering Science, Osaka University, Toyonaka-shi, 560 Japan

ルール).

② 石の死活の判定を不要にし，地を確定するために，仮終局を導入する．すなわち，(1)双方打つところがなくなって，パスを連続したとき仮終局とし(一般のルールの終局に相当)，その後も打着を続ける．(2)仮終局後双方連続してパスをしたとき，終局となる．(3)仮終局のパスは，そのつどペナルティとして自分の石を一つ相手に渡す．但し，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．

ルールの記述する方法として，ゲームの開始から終局までを1手打つごとに遷移する状態の系列と考え，それを五つの関数とその補助関数で定義することにした．ここではそれらの関数を代数的記述言語 ASL/1[4] を用いて，抽象的順序機械(Abstract Sequential Machine, ASM)[4],[5] と呼ばれる記述方式で記述する．この ASM 記述は，効率のよい実行ができる関数型プログラムへの変換方法が知られており[4],[5]，ここで記述した囲碁のルールもその方法に従って，関数型言語 ASL/F[4]のプログラムに変換した．得られたプログラムに，いくつかの入出力関数を追加することによって，人間同士の対戦の際に用いる碁盤プログラムとなった．これは，対戦中の現在の盤面を表示するとともに，禁じ手の判定や，勝敗の判定を厳密にルールに従って行う．このプログラムを実際に動かし，ルールに沿って正しくゲームを進行させせることができるかどうか調べることによって，もとの代数的なルールの記述が正しいかどうか知ることができる．このプログラムは，UNIX 上で動く ASL/F の最適化コンパイラ[6]によりコンパイルされ，実行される．

## 2. 代数的記述言語

ルールの記述に用いた ASL/1は，代数的仕様記述法 ASL/ *[4] における文法および公理の記述法を具体的に定めた代数的仕様記述言語である．図1に ASL/1の記述例を示す．(池田ルール記述に用いる補助的なデータ型の定義．詳細は**4.1**で述べる．) ASL/1のテキストは，次の三つの要素から構成される．

(1) プロジェクト文　　指定されたデータ型やそれに関する関数の仕様を取り込み，それらを基底データ型として仕様中で用いることができるようにする．include primitives によって，本論文で用いる基底データ型の整数 (integer) (基法関数としては，+，−，=，>等を用いる)，ブール (boolean) (&，or)，および集合 (set) (∪，−(差集合)，∈，∉，| |(要素数)) が取り込まれる．

```
include primitives;
  color    →  黒;  →  白;  →  空;          ①
  position →   <integer, integer>  ;
  integer  →    座標X (position) ;
  integer  →    座標Y (position) ;
0-1:座標X (<X,Y>) ==X;
0-2:座標Y (<X,Y>) ==Y;
  player   →    黒;  →  白;                 ②
  position →    パスする;                   ③
  move     →    <player , position>;
  position →    位置 (move) ;
  player   →    打ち手 (move) ;
0-3:位置 (<A, P>) ==P   ;
0-4:打ち手 (<A, P>) ==A ;
  board    →  盤(state) ;
  color    →  点の色 (state, position) ;
           →  点の色1 (board, position) ;
0-5:点の色1 (盤 (S) , P) ==点の色 (S, P) ;
  /*  点の色1は現在の盤面を与えれば点の色と同
     じ。引数の型が異なる  */
```

図1 ASL/1による記述例(データ型の定義)
Fig. 1 An example of ASL/1 specification.

(2) プロダクション文　　関数や定数の型宣言文．ある関数 $f$ の関数値の型が $t_0$ で，引数の型がそれぞれ $t_1$，$t_2$，…ならば，$t_0 \rightarrow f(t_1, t_2, \cdots)$; と書く．定数は引数をもたない関数である．ある定数 c の型が $t_0$ であるならば，$t_0 \rightarrow c$; と書く．これらのプロダクション文は，仕様中のいずれの場所においてもよい．また，略記法として，関数 $f_1$，$f_2$，…が同じ関数値の型 $t_0$ をもつ場合，$t_0 \rightarrow f_1(\cdots); \rightarrow f_2(\cdots); \cdots$; と書くことができる．

(3) 公理　　変数を含む二つの式の合同関係を示す．左辺には同じ変数が2度以上現れず，また，右辺に現れる変数は，必ず左辺に現れていなければならない．二つの式は，式1==式2；と書く．

代数的言語で仕様を書くとき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[4],[5](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，基底データ型や補助的に用いるデータ型以外に，いくつかの成分から構成される状態(記述中では state 型)と呼ばれる型が定義されている．記述中の関数は次の状態を出力する状態遷移関数，特定の成分を取り出す値関数 (Value Function)，そしてその他の関数に分類される．そして，遷移が起こったとき状態の各成分の値がどうなるかを

値関数 (遷移関数 ($S$, …)…)==式；

の形で公理で書く ($S$ は state 型の変数)．抽象的順序機械の形の記述は，関数型言語や手続き的言語の効率

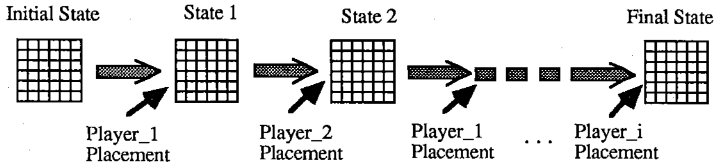


図2　ゲームの状態遷移
Fig. 2　State transitions in GO game.

的なプログラムに機械的に変換可能である．

ASL/Fは，ASL/＊に基づいた関数型プログラミング言語である．ASL/Fでは，通常のプログラミング言語によって用いることができるデータ型，例えば整数，実数，文字，文字列，配列，ファイルやそれらの並び(tuple)を用いて，プログラムできる．ASL/Fプログラムでは，公理の形が，$f(x_1,\ x_2,\ \cdots)==$式；に限られている．また，ASL/Fプログラムのための最適化コンパイラが作成されている[6]．

## 3.　ルールの定義方法

碁などの盤上で石，駒等を動かすゲームは，図2のように，初期状態から，最終状態への状態遷移の繰り返しとして考える．ここでいう状態とは，現在の盤面のみならず，ゲーム進行上必要な，アゲ石の数，次の手を打つ対戦者名等の値の集合である．状態遷移は，各対戦者が与える打ち手によって起こる．

ここでは，この状態遷移に関する関数も含めて以下の五つの関数(定数も含む)で，ゲームの開始から終了までの過程を表すものと考える．すなわち，これらの関数を定義することによりゲームのルールの定義を行う．なお以下の本文中では各関数名や定数名が他の用語と紛らわしいときは下線を引く場合がある．

① 初期状態を表す定数：初期状態

② 状態遷移関数：遷移

③ 打着した手が許される手か否かの関数：正しい手？

④ 最終状態か否かの関数：最終状態

⑤ 最終状態において勝者を与える関数：勝者

池田ルールの各条文のうち，石が除去される状態，ペナルティ石等の記述については，②に関する記述，交互着手，同形再現禁止，自殺打着禁止の記述は③に関する記述，仮終局や終局の記述は④，得点や条件設定の記述は⑤に関する記述と考えられよう．また，ルールでは陽に述べられていないものでも，③，④，⑤の定義のために必要な値で，打つ手によってその値が変化しうるものは(例えば仮終局を示すフラグ等)は②で記述する．①に関しては，陽にルールでは記述されていないが，盤面上のすべての点は空である，アゲ石やペナルティ石の数は0，等については定義する必要がある．また，②で用いる各状態の成分のうち，ある初期値をもつべきものも定義しておく．以下**4.**で各関数の定義について述べる．

## 4.　ルールの記述

ゲームを構成する五つの関数を公理の形で定義したものについて説明する．なお，定義中に用いる補助的な関数の定義の詳細は文献(3)を参照されたい．ここでは池田ルールに基づいて各関数を定義しているが，同様な方法で他の盤面を用いるゲームの定義もできよう．

### 4.1　データタイプ

図1に，五つの関数を簡潔に定義するために用いる，データ型，color，position，player，move，boardを示す．

（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)に置かれている石の色を示すデータ型で，黒，白，空のいずれかの値(定数)をもつ(図1中の①)．

（2） position　　盤面上の一つの点を指定する2整数の組で〈$X$ 座標，$Y$ 座標〉で構成される．関数座標 $X$，座標 $Y$ でそれぞれ $x$ 座標，$y$ 座標を取り出す(公理0-1，0-2)．

（3） player　　対戦者名のうちの1人を表す．ここでは，2人の対戦者をそれぞれ黒と白という値で表す(②)．

（4） move　　ある対戦者がどこに打つか(パスも含む，③)を示す．関数位置は，打つ場所を取り出し(0-3)，打ち手は対戦者を取り出す(0-4)．

（5） board　　一つの盤面全体を表す．関数盤によって状態から取り出され，関数点の色1によって点の色と同じ値が与えられる(0-5)．

### 4.2 初期状態

初期状態はstate型の定数である．状態の各成分のうち，ゲームの開始時の初期値を定義する必要のあるものは，各値関数を用いて次のような公理を書く(図3)．

値関数(初期状態)＝＝初期値；

ゲーム途中のある段階から，値が必要になるような成分については，このような公理を書かず，初期値の定義を行わない．ここでは，[1]初期状態として，盤面のすべての点は空(1-1)，[2]黒と白のアゲ石やペナルティ石は0(1-2～1-5)，[3]仮終局や終局の状態を表すフラグは偽(1-7～1-8)，[4]盤面が過去と同型かを調べるための盤面の集合は空(1-9)，が定義されている．

初期状態での盤面上に，ハンディのための置き石が必要な場合は，以下のようにその点(この例では〈4, 4〉と〈16, 16〉)を指定して石の色を書けばよい．

点の色(初期状態, $P$)＝＝

　if $P$＝〈4, 4〉or〈16, 16〉then 黒

　　　　　　　　　　　　　else 空；

### 4.3 状態遷移関数

状態遷移関数遷移は，状態(state型)と，打つ手(move型)を入力として，次の状態を出力する(図4)．公理2-1は，入力される手によって盤面上の各点がどのように変化するかを定義している．すなわち任意の点$P$の石の色(黒か白か空)は，手がパスであるなら変化しない．もしいま，黒の打つ番で位置($TE$)で与えられるある点に打つことによって，点$P$の黒石が囲まれるならば，点$P$の色は空になる．もし黒の番でも，黒石が点$P$にないか，あってても囲まれないならば点$P$の色は変わらない．また白の番についても同等のことを記述している．

```
        integer →  アゲ石白（state）；
                →  アゲ石黒（state）；
                →  ペナルティ白（state）；
                →  ペナルティ黒（state）；
        boolean →  仮終局か？（state）；
                →  終局か？（state）；
        board_set  →  過去の盤（state）；
1-1:点の色（初期状態，P）＝＝空；
1-2:アゲ石白（初期状態）＝＝0；
1-3:アゲ石黒（初期状態）＝＝0；
1-4:ペナルティ白（初期状態）＝＝0；
1-5:ペナルティ黒（初期状態）＝＝0；
1-6:仮終局か？（初期状態）＝＝False；
1-7:仮終局か１？（初期状態）＝＝False；
1-8:終局か？（初期状態）＝＝False；
1-9:過去の盤（初期状態）＝＝φ；
```

図3　初期状態の定義
Fig. 3　Definition of the initial state.

公理2-2は，正しい手の判定の一つとして，黒または白が連続して打たないよう，一つ前の打ち手を状態の成分として記憶することを定義している．この成分の初期値は定義する必要がないので，前の打ち手(初期

```
  boolean →  前の手はパスか？（state）；
          →  前の手はパスか１？（state）；
  player  →  前の打ち手（state）；
2-1:点の色（遷移（S,TE），P）  ＝＝
  if  パス？（TE）  then  点の色（S，P）
  else  if  黒番？（TE）  then
    if  位置（TE）＝P  then  黒
    else  if
     白が除去される状態か？（S，位置（TE），P）
                         then  空
                         else  点の色（S，P）
  else              ／＊  白番  ＊／
    if  位置（TE）＝P  then  白
    else  if
       黒が除去される状態か？（S，位置（TE），P）
                         then  空
                         else  点の色（S，P）    ；
2-2:前打ち手（遷移（S，TE））＝＝打ち手（TE）；
2-3:過去の盤（遷移（S，TE））＝＝
       過去の盤（S）  ∪  （盤（遷移（S，TE）））；
 ／＊  同形判定のために過去の盤面を覚える  ＊／
2-4:前の手はパスか？（遷移（S，TE））＝＝
                              パス？（TE）    ；
2-5:仮終局か？（遷移（S，TE））＝＝
    仮終局か？（S）
or   （パス？（TE）＆前の手はパスか？（S））；
2-6:前の手はパスか１？（遷移（S，TE））＝＝
                  仮終局（S）＆パス？（TE）    ；
     ／＊  仮終局後のパスか？  ＊／
2-7:終局か？（遷移（S，TE））＝＝
       パス？（TE）＆前の手はパスか１？（S）    ；
2-8:アゲ石白（遷移（S，TE））＝＝
if  黒番？（TE）＆白を抜くか？（S，位置（TE）
 then
  ｜囲まれる白石（S，位置（TE））｜＋アゲ石白（S）
  else  アゲ石白（S）    ；
  ／＊  アゲ石として取られた白石の数  ＊／
2-9:アゲ石黒（遷移（S，TE））＝＝  ．．．
  ／＊  2-8  と白黒が異なる同様な定義  ＊／
2-10:ペナルティ白（遷移（S，TE））＝＝
       if  not（仮終局か？（S））  then  0
  else  if  白番（TE）＆パスか？（TE）  then
         if  前の手はパスか１？（S）／＊終局＊／
           ＆  仮終局後先着（S）＝白
         then  ペナルティ白（S）
         else  ペナルティ白（S）＋1
  else    ペナルティ白（S）    ；
2-11:ペナルティ黒（遷移（S，TE））＝＝  ．．．
    ／＊  2-10と白黒が異なる同様な定義  ＊／
2-12:仮終局後先着（遷移（S，TE））＝＝
       if  仮終局か１？（S）  then  打ち手（TE）
                      else  仮終局後先着（S）；
    ／＊  仮終局直後の打つ者の名前  ＊／
2-13:仮終局か１？（遷移（S，TE））＝＝
       not（仮終局か？（S））
     ＆  パス？（TE）  ＆  前の手はパスか？（S）；
    ／＊  仮終局直後かどうか  ＊／
```

図4　状態遷移の定義
Fig. 4　Definition of state transition.

状態）を左辺にもつような公理はない．

公理 2-3 では，同形の盤面を調べるための過去の盤面の集合に，現在の盤面を加えることを定義している．2-4～2-6 は仮終局および終局状態を示すフラグの値の定義で，2-8～2-13 は，黒，白各々のアゲ石，ペナルティ石の数の定義である．

### 4.4 正しい手，最終状態，および勝者

図 5 に正しい手，最終状態，および勝者の定義を示す．ここでは，正しい手とは，①前の打ち手と違う打ち手の行う打着で（同じ打ち手が連続して着手しない），かつ②その手がパスかまたは実際の盤面内の空点への禁じ手でない置き石と定義される（3-1）．禁じ手は，(a) 打つことによってゲーム開始から今までのいずれかの盤面と同形の盤面になる手か（同形再現），または，(b) 打つことによって打ち石自身が直ちに囲まれる状態になるような手（自殺打着）をいう（3-2）．それぞれの関数の詳細な定義はここでは省略する．

```
  /*  3．正しい手？        */
    boolean  → 正しい手？（state, move）；
3-1:正しい手？（S, TE）==
     （前の打ち手（S）≠打ち手（TE））&
     （パス？（TE）   or
        （盤面内か？（位置（TE））&
         点の色（S, 位置（TE））＝空  &
         not（禁じ手か？（S, TE））））   ；
3-2:禁じ手か？（S, TE）==
   同型再現か？（S, TE）  or  自殺か？（S, TE）；

  /*  4．最終状態  */
    boolean  → 最終状態（state）；
4-1:最終状態（S）==終局か？（S）；

  /*  5．勝者  */
    winner  →  player；
            →  引き分け；
            →  勝者（state）；
5-1:勝者（S）==
   if  （黒の得点（S）－コミ）＝白の得点（S）
     then  引き分け
     else  if
  （黒の得点（S）－コミ）＞白の得点（S）  then 黒
     else              白；

   integer →  コミ；
           →  黒の得点（state）；
           →  白の得点（state）；
           →  アゲ石白（state）；
           →  アゲ石黒（state）；
5-2:黒の得点（S）==
アゲ石白（S）＋ペナルティ白（S）＋黒地の数（S）；
5-3:白の得点（S）==
アゲ石黒（S）＋ペナルティ黒（S）＋白地の数（S）；
```

図 5　正しい手，最終状態，勝者の定義
Fig. 5　Definition of the legal movements, the final state, and the winner.

最終状態は，状態遷移関数と共に定義された値関数終局か？をそのまま用いることができる（4-1）．関数勝者は，黒，白または引き分けのいずれかの定数を返す（5-1）．定数コミはいわゆるハンディで先に打つ打ち手から定められた数を得た得点から引く．ここでは先に打つ打ち手を黒か白に定義していない．従って黒が先ならばコミは正の値，そうでないなら負の値をとるものとする．黒の得点とはアゲ石として得た白石の数，ペナルティとして渡された白石の数，そして囲んだ黒地の数の総和である（5-2）．

得られた公理全体は，データ型や補助関数などの詳細な定義を含め全体で 70 個，約 400 行になった．

## 5. 関数型プログラムへの変換

前の章で述べられた ASL/1 仕様はいくつかのデータ型の公理（color, move, player, board）および集合型の値を返すいくつかの補助関数（例えば石の種類，黒地，白地）の定義を除き抽象的順序機械の仕様を満たす．これらの定義されているデータ型については，関数型言語組み込みの整数やその配列，組（tuple）を用いて実現できる．また集合値を返す補助関数については，各々の要素について公理に書かれた条件を調べることによって関数型の定義に変換した．これによりすべての記述は抽象的順序機械の仕様を満たすように変更できた．

### 5.1 関数型プログラムへの変換方法の概要

いま，抽象的順序機械の記述として

$f_1$(初期状態)$==c_1$；

…

$f_n$(初期状態)$==c_1$；

$f_1$(遷移$(S,\ MV)$)$==e_1$；

…

$f_n$(遷移$(S,\ MV)$)$==e_n$；

が与えられたとする．ここで $f_1$～$f_n$ は値関数，遷移は状態遷移関数，$c_1$～$c_n$ は定数，$e_1$～$e_n$ は，関数 $f_1$～$f_n$，変数 $S$ や $MV$，その他定数や補助関数から構成される式である．このほか，関数正しい手か？，最終状態，勝者が定義されているとする．

変換方法は，一般の抽象的順序機械の記述から関数型の記述への方法[(4),(5)]に加えて，プログラムの停止条件や，そのときの出力，打つ手の入力方法等について，前章で定義した関数を利用するよう考慮した．以下，関数型プログラムに変換する手順を示す．

（1） 遷移関数に対応して関数型プログラムに関数名遷移′を設ける．抽象的順序機械における状態遷移が，

この遷移′の呼出しに対応する．

（2） 各値関数 $f_1$～$f_n$ に対応して変数 $x_1$～$x_n$ を設け，遷移′の引数とする．すなわち，状態の各成分に対応した引数を遷移′はもつ．

（3） 遷移′の右辺は，自分自身を再帰的に呼び出すように定義する．その際各引数は $e_1$～$e_n$ で定義されるように各状態成分を更新して渡す．すなわち各 $e_i$ 中に含まれる各部分式 $f_j(S)$ を変数 $x_j$ に置き換えた式 $e_i'=e_i[x_j/f_j(S)]$，$1 \leqq j \leqq n$ を，呼び出す遷移′の引数として書く．この再帰呼出しの停止条件としては関数最終状態′を用いる（同様に関数最終状態の定義中に現れる値関数を変数で置き換えたもの．以下勝者′，正しい手′も同様）．また出力として勝者を出力するように定義する．

（4） この遷移′を各初期値 $c_1$～$c_n$ で呼び出す関数 start を定義する．この start の入力は黒と白の，ゲーム開始から終了までの各打ち手の系列とする．この系列から打ち手を一つずつ取り出すために関数 head および残りの系列を得るために tail を用いて，遷移′に各々渡す．

（5） 各入力が正しい手か否かの判定を正しい手か？′を用いて遷移′の各呼出時に行う．もし正しい手でなければ関数例外処理を呼び出すものとする（例外処理の例としては，例えば再入力等が考えられよう）．得られた関数は以下のようになる．

start($IN$)＝＝
　遷移′($c_1$, $c_2$, …, $c_n$, head($IN$), tail($IN$))；
遷移′($x_1$, $x_2$, …, $x_n$, $MV$, $IN$)＝＝
　if 最終状態′ then 勝者′
　else if 正しい手′ then
遷移′($e_1'$, $e_2'$, …, $e_n'$,
　　　head(IN), tail(IN))
　　else 例外処理；

### 5.2 碁盤プログラム

抽象的順序機械記述で用いられた値関数点の色および点の色1は，引数として状態以外に点の座標をとる．そして，各公理では，この座標値として変数を用いて任意の値を取り得るように記述されている．この変数を用いることによって盤面上の各点それぞれの値関数を用意する必要がなかった．関数型プログラムでは，各々の点に対応した引数を遷移′に用意するのではなく，これらをまとめて現在の盤面を覚える配列一つを設けそれを遷移′の引数とした．その各要素は点の色の定義右辺に従って更新する．抽象的順序機械記述で陽に初期値が与えられていない，値取出し関数に対応する遷移′の引数については，適当な初期値を与えておく．

得られた各関数に，入出力関数を付加することによって，碁盤プログラムが得られた．これは人間2人が交互に手を入力しゲームを行うもので，①初期盤面等のゲーム開始時の設定，②盤面等の各状態成分の更新，表示，③入力された手が正しい手か否かの判定，④ゲームの終了の判定，⑤終了時に勝者の決定，が行われる．このプログラムを実行させて，実際にゲームの進行を行うことによっても，もとの代数的なルールの記述が正しく行われているかどうかを知ることができた．

得られた関数型プログラム全体は78の定義関数をもち，約720行であった．そのうち8定義関数75行は入出力に関するものである．このプログラムはUNIX上で動くASL/Fコンパイラを用いて実行される．

## 6. むすび

囲碁のルールのASL/1を用いた代数的記述には，約3週間，またASL/Fを用いた関数的プログラムへの変換は約1カ月を要した．

ここでは，比較的厳密に自然語の記述として与えられているルールを形式的に記述し，更にそれを実行可能な関数型プログラムに変換した．同様な方法で，盤を用いる他のゲームのルールを形式的に記述し，実行可能なプログラムを得ることができる．また得られたプログラムは，人間と対戦するゲームプログラムの一部となろう．

ゲームプログラムには勝つための種々の戦略や作戦を組み込む必要がある．定石のようにある決まった打ち方しか要求されない部分は，同様な手法で形式化でき，それを実行するプログラムを得ることができよう．また，容易に形式化できないような場合でも，形式化を試みることによって，その形式化が困難な部分とその他の部分が分離でき，作戦や戦略の整理に役立つであろう．

**謝辞** ASL/1の記述に関し助言を頂いた大阪大学基礎工学部の杉山裕二助教授，東野輝夫博士，関浩之博士に感謝します．関数型プログラムへの変換に協力して頂いた林明宏氏（現㈱丸紅）に感謝します．囲碁に関する種々の助言を頂いた日本ユニシス㈱の山住厳氏に深謝します．

## 文　献

(1) 林　裕："囲碁百科辞典"，金園社(昭 40-05).
(2) 池田敏雄："囲碁ルールについて1～11"，囲碁新潮，43年

9月号～44年11月号（昭43-09～44-11）.
(3) 井上克郎，鳥居宏次：“囲碁のルールの代数的記述について”，情報処理ソフトウェア工学研究会，**SW52-8**, pp. 57-64（昭62-02）.
(4) 嵩　忠雄，谷口健一，杉山裕二：“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，pp. 93-123，オーム社（昭61-06）.
(5) K. Torii, Y. Morisawa, Y. Sugiyama and T. Kasami : “Functional programming and logical programming for the telegram analysis problem”, Proc. of 7th International Conference on Software Engineering, pp. 463-471 (March 1984).
(6) 関　浩之，八木　輝，井上克郎，杉山裕二，後藤信一：“関数形言語 ASL/F コンパイラの UNIX での実現”，昭60前期情処全大，IQ-3, pp. 373-374（昭60-03）.
(7) 井上克郎，大西　諭，鳥居宏次：“囲碁ルールの代数的記述とその関数型プログラムへの変換”，日本ソフトウェア科学会第4回大会論文集，C-2-5, pp. 199-202（昭62-11）.

（昭和63年3月15日受付）

**井上　克郎**

昭54阪大・基礎工・情報卒．昭59同大大学院博士課程了．同年同大・基礎工・情報・助手．昭59～61ハワイ大助教授．関数型言語の処理系等の研究に従事．工博．情報処理学会，ACM，IEEE 各会員．

**鳥居　宏次**

昭37阪大・工・通信卒．昭42同大大学院博士課程了．工博．同年電気試験所（現電子技術総合研究所）入所．昭50ソフトウェア部言語処理研究室室長．昭59阪大・基礎工・情報教授．ソフトウェア工学の研究に従事．