知能プログラミング演習 II 課題 5

グループ 8
29114060 後藤 拓也

2019 年 12 月 10 日

■提出物　rep5

■グループ　グループ 8

■メンバー

学生番号	氏名	貢献度比率
29114003	青山周平	20
29114060	後藤拓也	20
29114116	増田大輝	20
29114142	湯浅範子	25
29119016	小中祐希	15

1　課題の説明

必須課題 5-1　目標集合を変えてみたときに, 動作が正しくない場合があったかどうか, 実行例を示して考察せよ. また, もしあったならその箇所を修正し, どのように修正したか記せ.

必須課題 5-2　教科書のプログラムでは, オペレータ間の競合解消戦略としてランダムなオペレータ選択を採用している. これを, 効果的な競合解消戦略に改良すべく考察し, 実装せよ. 改良の結果, 性能がどの程度向上したかを定量的に（つまり数字で）示すこと.

必須課題 5-3　上記のプランニングのプログラムでは, ブロックの属性（たとえば色や形など）を考えていないので, 色や形などの属性を扱えるようにせよ. ルールとして表現すること. 例えば色と形の両方を扱えるようにする場合,A が青い三角形,B が黄色の四角形,C が緑の台形であったとする. その時, 色と形を使ってもゴールを指定

できるようにする（”green on blue” や”blue on box”のように）

必須課題 5-4　上記 5-2, 5-3 で改良したプランニングシステムの GUI を実装せよ．ブロック操作の過程をグラフィカルに可視化し, 初期状態や目標状態を GUI 上で変更できることが望ましい．

発展課題 5-5　ブロックワールド内における物理的制約条件をルールとして表現せよ．例えば, 三角錐（pyramid）の上には他のブロックを乗せられない等, その世界における物理的な制約を実現せよ．

発展課題 5-6　ユーザが自然言語（日本語や英語など）の命令文によってブロックを操作したり, 初期状態／目標状態を変更したりできるようにせよ．なお, 命令文の動詞や語尾を 1 つの表現に決め打ちするのではなく, 多様な表現を許容できることが望ましい．

発展課題 5-7　3 次元空間 (実世界) の物理的な挙動を考慮したブロックワールドにおけるプランニングを実現せよ．なお, 物理エンジン等を利用する場合,Java 以外の言語のフレームワークを使って実現しても構わない．

発展課題 5-8　教科書 3.3 節のプランニング手法を応用できそうなブロック操作以外のタスクをグループで話し合い, 新たなプランニング課題を自由に設定せよ．さらに, もし可能であれば, その自己設定課題を解くプランニングシステムを実装せよ．

2　発展課題 5-6

ユーザが自然言語（日本語や英語など）の命令文によってブロックを操作したり, 初期状態／目標状態を変更したりできるようにせよ．なお, 命令文の動詞や語尾を 1 つの表現に決め打ちするのではなく, 多様な表現を許容できることが望ましい．

2.1　手法

ユーザの命令によるブロック操作を行う際, 何も考えずにただ漠然とブロックを動かしていては, プランニングの意味をなさない．例えば「ブロックを A, B, C の順に上から積んでいく」という目標が設定されている場合, オペレータは「C の上に B を置く」のちに「B の上に A を置く」とするのが理想であり, プランニングではその処理を行えなければならない．ここでユーザが「C の上に A を置く」などでたらめなことをしてしまっては,

プランニングの意味がない. そのため, ここにおける”ユーザによるブロック操作”の定義を, ”プランニングを行っているユーザによるブロック操作”とする.
具体的には, プログラムが”おすすめ”の操作をユーザに知らせる. ユーザはそのおすすめを参考に処理を選択し, 実行する. もちろんユーザはプランニングを行っているという仮定のもとでの話なので, 条件に合わないオペレータ選択 (つまり, 現在の目標に関わる内容がオペレータの Add-list に存在しない) 場合は, そのユーザの選択はプランニングにそぐわないとして実行されず, 代わりにおすすめの処理が行われるとする.
プログラムが勧める”おすすめ”のオペレータとは, 課題 5-2 で行った競合解消戦略によるオペレータ選択の内容である.

ここで問題となるのは, オペレータを 4 つの中から選択できたとしても, 中身までは選択できない. 上記の例をもう一度考えると, 副目標「B on C」に対して, 「Place B on C」という内容はできても, 「Place A on C」という処理は, 行えない.

ブロック操作における競合解消戦略として, 教科書のプログラムではランダム関数を用いたオペレータの選択をしているが, その選択を適切に変える. 以下の 2 つの操作を試みている.

1. 推論の最中にユーザーの入力により, オペレータをその都度選択していく
2. ゴール状態と現在の状態を比べてより最適なオペレータを選択する

この推論システムは大きく, plannnig メソッドと plannigAGoal メソッドを再帰的に呼び合うことで成り立っている. 以下の図 1 を参考にしてほしい.
plannningAGoal メソッドで, plannnig メソッドから 1 つ取り出された目標状態を見て, 現状態に含まれていなければ各ルールの Add-list から Unify できるオペレータを, 現状態に含まれていればそのまま次の目標へといった流れである. この planningAGoal メソッドでルールを選ぶ方法をランダムから変えれば良いことが分かる.

ランダムにオペレータを選択する際に, 教科書のプログラムは選択したオペレーションを一度, リストから取り除き, その後リストの最後に格納しているが, それだと適切なオペレータを選択できない.
そのため, 図 2 のように, operators リストの順序を適切に扱うことで, 自由にオペレータを選択, 取り出すことにする.

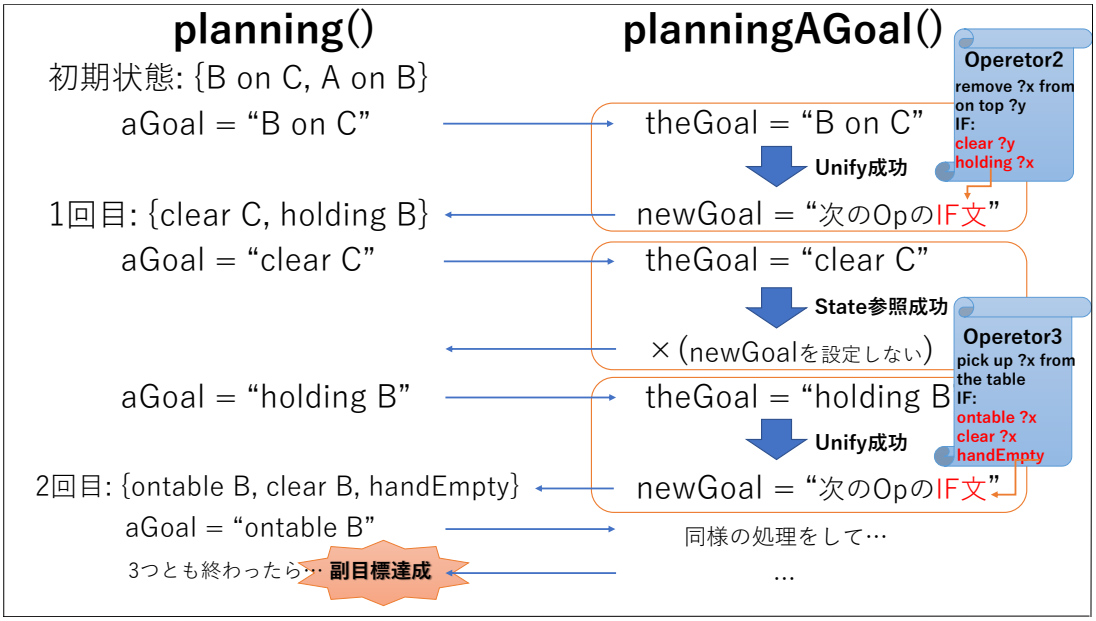

図 1　plannnig における相互メソッドの再帰構造

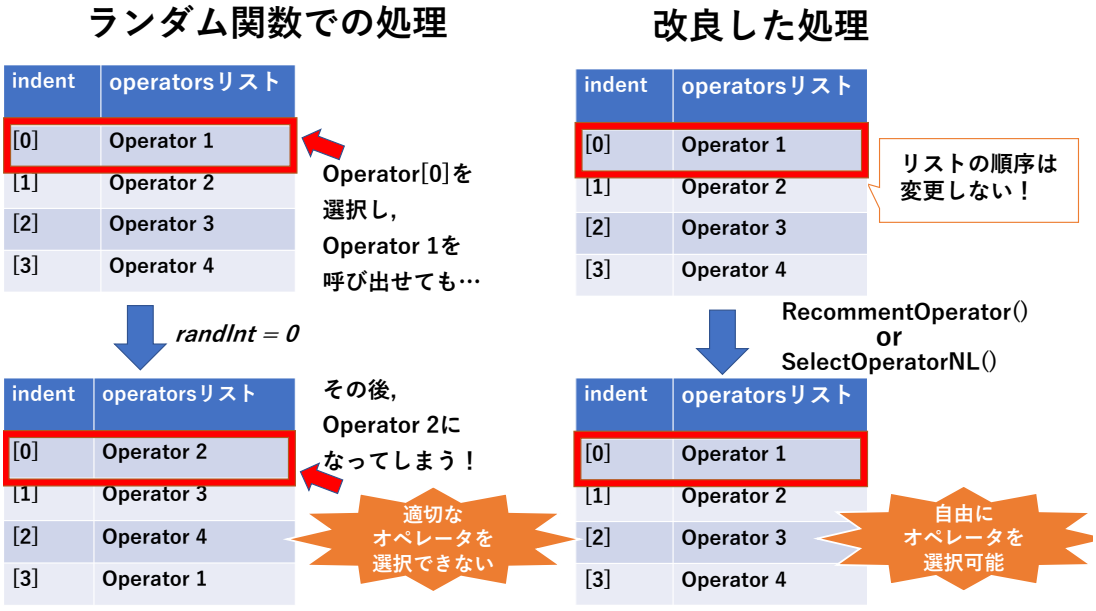

図 2　operetars リストの動き

2.2 実装

planningAGoal メソッドにおけるオペレータの選択においては, それぞれ新しくメソッドを作成し, そこで適切なオペレータの番号を返すことになっている. 推論の最中にオペレータを選択する SelectOperetorNL メソッドと,, 最適なオペレータ選択をする ReccomendOperetor メソッドを作成した.

ソースコード 1　オペレータ選択

```
1         //1.ランダム用
2         int randInt = Math.abs(rand.nextInt()) % operators.size();
3         Operator op = (Operator)operators.get(randInt);
4         operators.remove(randInt);
5         operators.add(op);
6         //cPoint = randInt;
7
8         //2.発展課題 5-6用
9         //int numOp = SelectOperatorNL();
10
11        //3.その他開発用
12        int numOp = RecommentOperator(theGoal);
13
14        /* 2.3のどちらかを使うときは,このコメントアウトを外す！
15        Operator op = (Operator)operators.get(numOp);
16        System.out.println("オペレータ内容は = " + op.name);
17        System.out.println("Thank you!");
18        cPoint = numOp;
```

ソースコード 2　推論中におけるオペレータ選択

```
1   /*
2         * 自然言語の命令によってオペレータの選択
3         * return オペレータの番号
4         */
5         private int SelectOperatorNL() {
6                 int opNumber = 0;
7                 String opString = null;
8                 Scanner scanner = new Scanner(System.in);
```

```
9                System.out.println("行う操作を入力してください");
10               opString = scanner.nextLine();
11
12               if(opString.contains("Place")) {
13                       opNumber = 0;
14               }
15               else if(opString.contains("remove")) {
16                       opNumber = 1;
17               }
18               else if(opString.contains("pick")) {
19                       opNumber = 2;
20               }
21               else if(opString.contains("put")) {
22                       opNumber = 3;
23               }
24               return opNumber;
25       }
```

自然言語といっても, Place や remove などを選ぶだけなので, 本質的な内容はできていない. 本来であれば, オペレータの中身（A や B など）も選択したかったが, Plannnig と組みあわせて行うことは難しかった.

ソースコード 3　最適なオペレータ選択

```
1    /*
2        * 最適な操作をできるようなオペレータの選択
3        * 仮引数 : theGoal の内容
4        * return: オペレータの番号
5        */
6        private int RecommentOperator(String theGoal) {
7                int opNumber = 0;
8                if(theGoal.contains("on")) {
9                        opNumber = 0;
10               }
11               else if(theGoal.contains("holding")) {
12                       opNumber = 2;
13               }
14               return opNumber;
15       }
```

現状は上に積むことしか考えていないので, 「A on B」のような目標では, 必ずオペレータの 0 番目「Place ?x on ?y」を選択し, 「holding A」の目標では, 必ず 2 番目のオペレータ「pick up ?x from the table」を選択するようになっている.

2.3 実行例

最終的なプランニングの結果を以下に示す.
以下はランダムなオペレータの選択をしている.

```
***** This is a plan! *****
pick up B from the table
Place B on A
remove B from on top A
Place B on A
remove B from on top A
Place B on A
remove B from on top A
Place B on C
pick up A from the table
Place A on B
```

次にプランニングによる最適解を”おすすめ”として表示させながらユーザーによるオペレータ選択の実行例を載せる.

```
goalList =
[B on C, A on B]
initGoalList() =
[B on C, A on B]
*** GOALS ***[B on C, A on B]
**B on C
現在の目標
B on C
現在の状態
[clear A, clear B, clear C, ontable A, ontable B, ontable C, ontable
    pyramid, handEmpty]
おすすめは = Place ?x on ?y
行う操作を入力してください
Place B on C
```

```
14 オペレータ内容は = Place ?x on ?y
15 unify 成功
16 オペレータの具体化:Place B on C
17 *** GOALS ***[clear C, holding B]
18 **clear C
19 [clear A, clear B, clear C, ontable A, ontable B, ontable C, ontable
       pyramid, handEmpty]
20 *** GOALS ***[holding B]
21 **holding B
22 現在の目標
23 holding B
24 現在の状態
25 [clear A, clear B, clear C, ontable A, ontable B, ontable C, ontable
       pyramid, handEmpty]
26 おすすめは = pick up ?x from the table
27 行う操作を入力してください
28 pick up B from the table
29 オペレータ内容は = pick up ?x from the table
30 unify 成功
31 オペレータの具体化:pick up B from the table
32 *** GOALS ***[ontable B, clear B, handEmpty]
33 **ontable B
34 [clear A, clear B, clear C, ontable A, ontable B, ontable C, ontable
       pyramid, handEmpty]
35 *** GOALS ***[clear B, handEmpty]
36 **clear B
37 [clear A, clear B, clear C, ontable A, ontable B, ontable C, ontable
       pyramid, handEmpty]
38 *** GOALS ***[handEmpty]
39 **handEmpty
40 [clear A, ontable A, ontable C, ontable pyramid, B on C, clear B,
       handEmpty]
41 *** GOALS ***[A on B]
42 **A on B
43 現在の目標
44 A on B
45 現在の状態
46 [clear A, ontable A, ontable C, ontable pyramid, B on C, clear B,
```

```
    handEmpty]
おすすめは = Place ?x on ?y
行う操作を入力してください
Place A on B
オペレータ内容は = Place ?x on ?y
unify 成功
オペレータの具体化:Place A on B
*** GOALS ***[clear B, holding A]
**clear B
[clear A, ontable A, ontable C, ontable pyramid, B on C, clear B,
    handEmpty]
*** GOALS ***[holding A]
**holding A
現在の目標
holding A
現在の状態
[clear A, ontable A, ontable C, ontable pyramid, B on C, clear B,
    handEmpty]
おすすめは = pick up ?x from the table
行う操作を入力してください
pick up A from the table
オペレータ内容は = pick up ?x from the table
unify 成功
オペレータの具体化:pick up A from the table
*** GOALS ***[ontable A, clear A, handEmpty]
**ontable A
[clear A, ontable A, ontable C, ontable pyramid, B on C, clear B,
    handEmpty]
*** GOALS ***[clear A, handEmpty]
**clear A
[clear A, ontable A, ontable C, ontable pyramid, B on C, clear B,
    handEmpty]
*** GOALS ***[handEmpty]
**handEmpty
***** This is a plan! *****
pick up B from the table
Place B on C
pick up A from the table
```

```
80 Place A on B
```

実際に, 「上に積む」ということだけに関しての最適なオペレータ選択を”おすすめ”として表示し, 現状態と現在の目標を見比べることで, ユーザがオペレータを選択できるようになっている.

2.4 考察

競合解消戦略を上手く取り入れれば, いろいろな初期化や目標状態でも対応できるが, その設定ができなかった. また, ユーザのオペレータ選択に関しても, 一文を見ているのではなく, あくまで「place」や「remove」といった一文字のみを見ているので, おそまつなプログラムであると言わざるを得ない. これからの課題でこの改善策を作っていきたい.

3 必須課題 5-2

> 教科書のプログラムでは, オペレータ間の競合解消戦略としてランダムなオペレータ選択を採用している. これを, 効果的な競合解消戦略に改良すべく考察し, 実装せよ.

3.1 手法

発展課題 5-6 を実装する際に利用した”ユーザーがオペレータを選択する方法”をもとに, デバックをしていき, 初期プログラムでは発生してしまう「Place A on A」や, 必須課題 5-3 での「Triangle on Ball」などの禁止制約を Plannig クラス, ないしは Unifier クラスを改善していった. Planner クラスの planning メソッドと plannnigAGoal メソッドの相互再帰構造においては, 図 1 をもとに簡単に説明をしたが, ここでは, その 1 つ 1 つの変数の内容まで踏み込み, Unifier クラスの Mtaching メソッドまで掘り下げていく必要があった.

まずは, 下の図を参考に, 問題点とその改良手法を示す. 上の図は, 始めの問題としては, [A を丸, B を台形, C を四角] という状態で, 属性の禁止制約として [丸の上に台形や四角を置いてはいけない] が存在する中で, 「Place B on A」が成立してしまう状態を示している. planning メソッドの 4 つ目の内容を見てみると, y2=A としてバインディングされると, y2 は y1 ともバインディングされているため, バックトラックの結果, 「Place B on

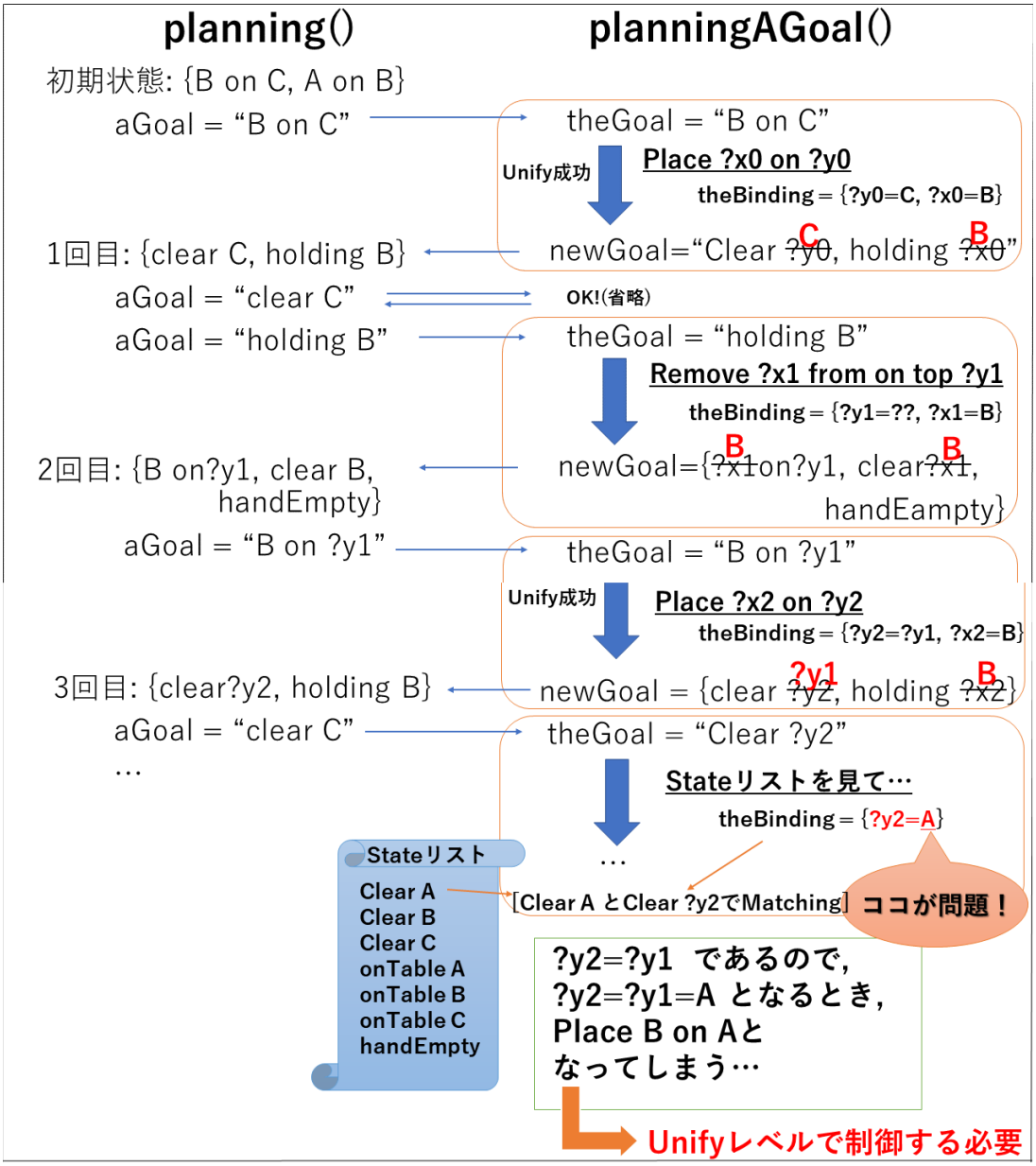

図 3　operetars リストの動き

A」が成立してしまうのである. もっと具体的には, 実際に y2 がバインディングされるのは, 「Clear ?y2」と State リストを上から順にマッチングしているので, State リストの一番上に存在する「Clear A」とマッチングしてしまい, ?y2=A となるのである. ここで, 禁止制約「B on A」からマッチングが不成立 (false を返す) ようにすればよい. またこれをベースに「Place A on A」のようなバグも改良していく.

禁止制約もとい, 実際に生成されていく「X on Y」の関係は, 全て HashMap を使い, X を Key とし, Y を Value として扱うことで, 物の上下関係を表現する.

3.2 実装

まずは, 属性をベースにした禁止制約を HashMap に格納していくプログラムを下に示す. これは, 課題 5-3 でかかわった属性クラス Attributions で定義されている.

ソースコード 4　禁止制約を HashMap に管理

```
1         //格納するHashMap
2         HashMap<String,List<String>> keyValueProhibit = new HashMap<
              String, List<String>>();
3
4         // 禁止制約追加メソッド
5         private void addProhibitRules() {
6                 System.out.println("###### Add prohibitRule ######");
7                 prohibitRules.add("ball on ball");
8                 prohibitRules.add("trapezoid on ball");
9                 prohibitRules.add("trapezoid on trapezoid");
10                prohibitRules.add("box on ball");
11                prohibitRules.add("box on box");
12                prohibitRules.add("pyramid on ball");
13
14                prohibitRules.add("ball on pyramid");
15                prohibitRules.add("box on pyramid");
16                prohibitRules.add("trapezoid on pyramid");
17                prohibitRules.add("pyramid on pyramid");
18                for(String prohibitRule: prohibitRules) {
19                        System.out.println("****** ProhibitRule:"+
                              prohibitRule+" ******");
20                }
21        }
22
23        //禁止制約をHashMap の Key と Value に格納
24        public void keyValueProhibit(ArrayList<String> list){
```

```
25              StringTokenizer st;
26              String tokenBuffer[];
27              ArrayList<String> buffer;
28              ArrayList<String> xValueList = new ArrayList<String>();
29              ArrayList<String> yValueList = new ArrayList<String>();
30              ArrayList<String> zValueList = new ArrayList<String>();
31              for(int i = 0; i < list.size(); i ++) {
32                      //リストの 1要素をトークンに分解して
33                      st = new StringTokenizer(list.get(i));
34                      //tokenBuffer に格納
35                      int length = st.countTokens();
36                      tokenBuffer = new String[length];
37                      for (int j = 0; j < length; j++) {
38                              tokenBuffer[j] = st.nextToken();
39                              System.out.println("tokenBuffer["+j+"]
                                    = " + tokenBuffer[j]);
40                      }
41
42                      //HashMap の Key と Value を格納していく
43                      if(tokenBuffer[0].equals("A")) {
44                              xValueList.add(tokenBuffer[2]);
45                      }
46                      else if(tokenBuffer[0].equals("B")) {
47                              yValueList.add(tokenBuffer[2]);
48                      }
49                      else if(tokenBuffer[0].equals("C")) {
50                              zValueList.add(tokenBuffer[2]);
51                      }
52              }
53
54              keyValueProhibit.put("A", xValueList);
55              keyValueProhibit.put("B", yValueList);
56              keyValueProhibit.put("C", zValueList);
57
58              System.out.println("keyValueProhibit = " +
                    keyValueProhibit);
59      }
```

addProhibitRules で定義された禁止制約のリストを, ここでは省略しているが, 属性表現「pyramid on pyramid」から通常状態「B on A」に変換してくれる editStatementList メソッドをかますことで, 禁止制約の表現をすべて”A, B, C”のものとして管理できる.(詳しくは, 課題 5-3 を参照してもらいたい). そして変換後のリストを受け取って, keyValueProhibit メソッドで「X on Y」を 3 つのトークンに分解し, 1 つ目のトークンを Key に, 2 つ目のトークンは”on”なので無視し, 3 つ目のトークンを Value に設定している.
禁止制約には 1 つの Key に対して複数の Value を持つものが存在する (「B on A, B on B」のように...) ので, HashMap の Value はリストにしなければならなかった.
ここの禁止制約で, 「Place B on A」(ただし, B は台形で, A はボールであり, ボールの上には何も置いてはいけないという属性禁止条件が存在する) や, 「Place A on A」(根本的にあり得ない状態) などを定義することができるので, 課題 5-3 の内容も課題 5-2 の内容もまとめて処理することができる.

次に, 実際の推論中に生成される「X on Y」関係と, 制約条件との照らし合わせを下に示す.
Planner クラスでオペレータ選択などが行われ, 実際に変数が具体化されるのは, Unifier クラスなので, その 2 クラスでの様子が見られる.

ソースコード 5　禁止制約を HashMap に管理

```
1  //Planner クラス
2    HashMap<String, String> p_productKeyOnValue; //現在の [X on Y]関係
3
4    //Attributes クラスから Planner クラスへ
5    setHashMap(attributions.a_productKeyOnValue);
6
7    //ゲッター
8    public HashMap<String, String> getHashMap(){
9          return p_productKeyOnValue;
10   }
11
12   //セッター
13   public void setHashMap(HashMap<String, String> pHashMap){
14         this.p_productKeyOnValue = pHashMap;
15   }
```

```
16
17   //planningAGoal メソッド内
18   //現在のKeyOnValue リストを Unifier クラスからもらってくるため
19         Unifier unification = new Unifier();
20         if (unification.unify(theGoal, aState, theBinding, attributions
                .keyValueProhibit, p_productKeyOnValue)) {
21               System.out.println("theBinding" + theBinding);
22
23               //Unifier からもらった KeyOnValue リストを,
                     Planner クラスに保存
24               for(String str : unification.getHashMap().keySet()) {
25                       System.out.println("Key = " + str + " Value =
                             " + unification.getHashMap().get(str));
26                       p_productKeyOnValue.put(str, unification.
                             getHashMap().get(str));
27               }
28               return 0;
29         }
30     }
31
32 //Unifier クラス
33 HashMap<String, List<String>> prohibit; //禁止制約
34 HashMap<String, String> productKeyOnValue; //現在の [X on Y]関係
35
36   //unify メソッド内
37     //Attributions クラスの Key 制約を持ってくる
38     HashMap<String, List<String>> keyProhibit = prohibit;
39         if(productKeyOnValue != null) {
40            System.out.println("unify 内:keyProhibit" + keyProhibit);
41         }
42         //これから調べる「Key on Value」の値を格納
43         String[] keyOnValue = new String[2];
44
45         for (int i = 0; i < length; i++) {
46           if (!tokenMatching(buffer1[i], buffer2[i])) {
47                 return false;
48           }
49           if(buffer1[1].equals("on")){
```

```
50            //「B on C」と「?x on ?y」を考えて,
51                 if(i == 0){
52                   keyOnValue[0] = buffer2[i];
53                   System.out.println("keyOnValue[0] ="+keyOnValue[0]);
54                 }
55                 else if(i == 2){
56                   keyOnValue[1] = buffer2[i];
57                   System.out.println("keyOnValue[1] ="+keyOnValue[1]);
58                 }
59              }
60         }
61         //保存
62         if(buffer1[1].equals("on")){
63           productKeyOnValue.put(keyOnValue[0], keyOnValue[1]);
64           System.out.println("productKeyOnValue="+productKeyOnValue);
65         }
66
67    //varMatching メソッド内
68         System.out.println("varMatching で..." + vartoken +"と"+ token
               );
69         System.out.println("productKeyOnValue で..." +
               productKeyOnValue);
70         for(String str1 : productKeyOnValue.keySet()){
71          //ここは,Value に変数「?y2」が入る
72          if(var(productKeyOnValue.get(str1))) {
73           //そのvartoken と token の組み合わせが,Init_KeyValue と同じだったら
74           for(String str2 : prohibit.keySet()){
75             for(int num = 0; num < prohibit.get(str2).size(); num++)
                   {
76                 System.out.println("prohibit.get(str2).get("+num+") =
                       " + prohibit.get(str2).get(num));
77                 if(str2.equals(str1) & prohibit.get(str2).get(num).
                       equals(token)){
78                     return false;
79                 }
80               }
81              }
82           }
```

```
83          }
```

Planner クラスと Unifier クラスで共通のリスト (produvtKeyOnValue) を共有しあい, 現在の [X on Y] の関係を保存共有する. Planner クラスで Unifier クラスを呼び出し, その内容をトークンごとで分解し,[X on Y] の分解に基づき, Key と Value を格納していく. 実際に, varMatching メソッドでは, 2 つ目の引数の token が, 次のバインディング候補であるため, それが禁止制約リストの Value 値と同じであれば false を返すようにしている.

3.3 出力結果

```
1  unify 内:keyProhibit{A=[A], B=[A, B], C=[A, C]}
2  varMatching で...?y3 と B
3  productKeyOnValue で...{B=?y3}
4  str2 = A
5  prohibit.get(str2).get(0) = A
6  str2 = B
7  prohibit.get(str2).get(0) = A
8  prohibit.get(str2).get(1) = B
9
10 unify 内:keyProhibit{A=[A], B=[A, B], C=[A, C]}
11 varMatching で...?y3 と C
12 productKeyOnValue で...{B=?y3}
13 str2 = A
14 prohibit.get(str2).get(0) = A
15 str2 = B
16 prohibit.get(str2).get(0) = A
17 prohibit.get(str2).get(1) = B
18 str2 = C
19 prohibit.get(str2).get(0) = A
20 prohibit.get(str2).get(1) = C
21 Init_Key(str) = A
22 Init_Key(str) = B
23 Init_Key(str) = C
24 theBinding{?y0=C, ?x0=B, ?y2=C, ?x2=B, ?y3=C, ?x3=B}
```

上は出力結果の一部を示している. Attributes クラスでの禁止制約が Planner クラスそして, Unifier クラスにうまく用いられ, keyProhibit として出力されていることが分かる.

はじめは ?y3 と B でマッチングを行うところであり, それが「X on Y」関係を持つように, HashMap の productKeyOnValue に保存されている. そして,「B on ?y3」に当てはまるものを禁止制約と照らし合わせている.「B on A」や「B on B」が実行されようとしているので, 禁止制約が発動していることがわかる. 次に「B on ?y3」に当てはまる候補として, C が用いられ,「B on C」は大丈夫であるので, ?y3=C と具体化されている.

3.4 考察

Planner クラスと Unifier クラスで共通のリストがあるおかげで, 離れたバインディ状態 (?y3=?y2 があり, その後?y3=C となる状態) でもしっかりと具体化することができてた. 上手く禁止制約条件を Unify の Matching メソッドまで行えたが,「A を取って B の上に置く」と「A を B から取り除く」などのループ処理が行われているため, プランニングが無限に陥る可能性が取り除けていない. 制約条件ではじいたあとの処理をさらに考える必要があった.

3.5 感想

出力結果とプログラムを理解していくのがとても大変であった. 初期プログラムの出力だけでは, なにが起こっているのか全く分からない. 分岐箇所やメソッドに結果などを細かく出力していくことで, プログラムを解読できた.
また, Planner クラスは Planning メソッドと PlanningAGoal メソッドの再帰構造によるバックトラックだったので, プログラムの改良がとても難しかった.「Place A on B」を一つとっても, いろいろな処理をしたのちに, 最終的にこの結果が出てくるのであって, その内部構造の理解が必須だった. まだ, 改良点や, 不具合が多いので, これからの課題でさらに深めていきたい.

参考文献

[1] Java による知能プログラミング入門 –著：新谷 虎松