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 9 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	No
29114060	後藤拓也	No
29114116	増田大輝	No
29114142	湯浅範子	No
29119016	小中祐希	No

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... よね... う〜ん... ユーザの選択をどこまで許すかという点である. 上記の例をもう一度考えると, 副目標「B on C」に対して, 「A on C」という処理は, 行えな... いかな... んんｎ（ ,,‘･ω･´ ）ﾝﾝﾝ？
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.

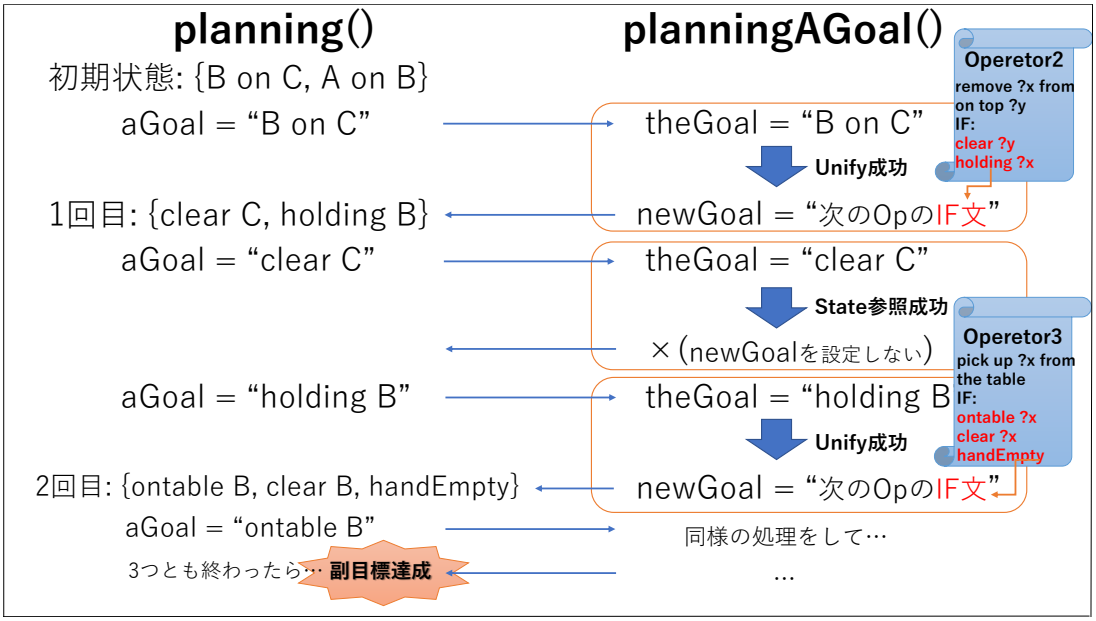

図 1　plannnig における相互メソッドの再帰構造

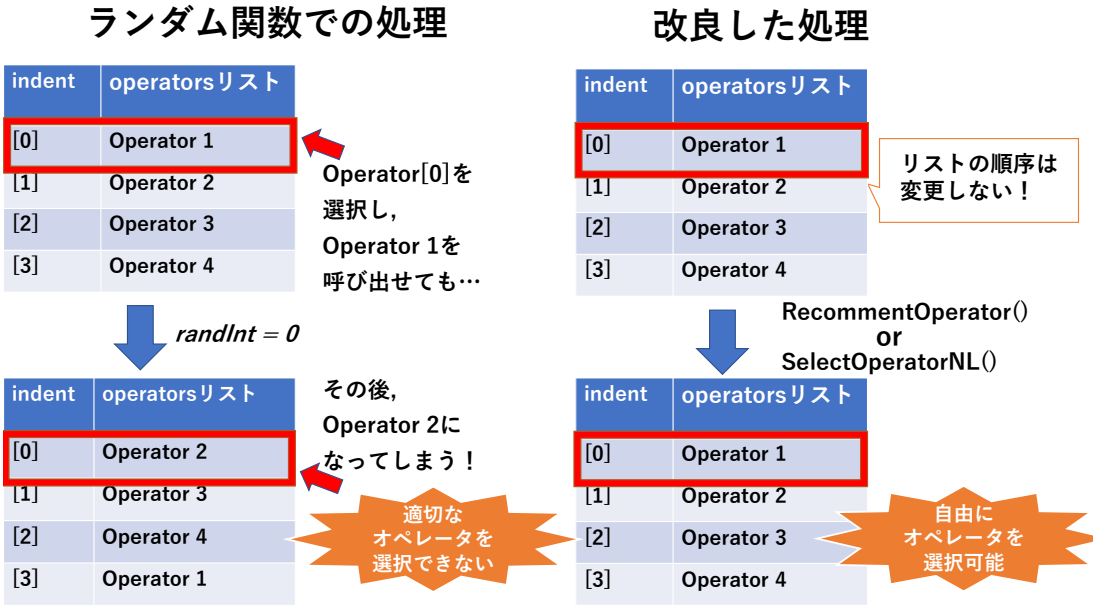

図 2　operetars リストの動き

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してね！
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                Scanner scanner = new Scanner(System.in);
8                System.out.println("数値を入力してください。");
```

```
9                 opNumber = scanner.nextInt();
10                return opNumber;
11        }
```

現状はただオペレータの数値を入力することしかできていない. この仮引数に String 型の自然言語をもち, その内容からオペレータを選択できるようにしていこう.

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.

```
1 ***** This is a plan! *****
2 pick up B from the table
3 Place B on A
4 remove B from on top A
5 Place B on A
```

```
6 remove B from on top A
7 Place B on A
8 remove B from on top A
9 Place B on C
10 pick up A from the table
11 Place A on B
```

次にオペレータ選択をチューニングした最適なプランニングを示す.

```
1 ***** This is a plan! *****
2 pick up B from the table
3 Place B on A
4 remove B from on top A
5 Place B on A
6 remove B from on top A
7 Place B on A
8 remove B from on top A
9 Place B on C
10 pick up A from the table
11 Place A on B
```

2.4 考察

2.5 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.

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