

知能プログラミング演習 II 課題 6

グループ 8

29114060 後藤 拓也

2020 年 01 月 04 日

■提出物 rep6

■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	null
29114060	後藤拓也	null
29114116	増田大輝	null
29114142	湯浅範子	null
29119016	小中祐希	null

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 必須課題 5-2 の改良

2.1 前回の内容

前回の課題 5-2 では、「Place A on A(存在しない制約)」や「Place B on A(属性を考量した場合の禁止制約)」を改良させるために、属性クラスである `Attributions` クラスで定義した禁止制約の `XonY` 関係を `HashMap` に格納し、その条件と現状態の `X'onY'` 関係を照らし合わせるという方法を用いた。オペレータ選択にはランダム選択を用いることで、どんなに適当な状態になっても禁止制約が発動されることを保証するようにした。

実行結果は以下のように失敗していた。

ソースコード 1 失敗実行例その 1

```
1 *** GOALS ***[holding B]
2 **holding B
3 そのオペレータは実行できません
```

```

4  おすすめのオペレータを使います
5  **clear ?y3
6  [clear A, clear B, clear C, ontable A, ontable B, ontable C, ontable
    pyramid, handEmpty]
7  *** GOALS ***[holding B]
8  **holding B
9  そのオペレータは実行できません
10 おすすめのオペレータを使います
11 **clear ?y3
12 [clear A, clear B, clear C, ontable A, ontable B, ontable C, ontable
    pyramid, handEmpty]
13 *** GOALS ***[holding B]
14 **holding B
15 そのオペレータは実行できません
16 おすすめのオペレータを使います
17 **clear ?y3
18 ... (以下無限に続く)

```

問題点としては、離れた制約がうまく機能していなかったことである。「B on ?y1」という目標に対して、オペレータ「Place ?x2 on ?y2」によって、「?x2 = B」と「?y2 = ?y1」となるのはよいが、その後、「Clear A」と「Clear ?y2」のマッチングを行う際に、「?y2 = ?y1 = A」となるが、この処理において、禁止制約 (X on Y 関係) との関係がうまく取れていなかったのである。以下の図 1 を参照してほしい。

初期設定から与えられる禁止制約:prohibit と、現在の状態を表す:productKeyOnValue を照らし合わせて、積み方に制約をかけている。productKeyOnValue には、「X on Y 関係」の目標が生じたときに、現在問題にしている「X on Y 関係」の何が X で何が Y かを保存する。これは、上記で述べたような離れた制約 (B on ?y1 の後に, Clear ?y2 など) にも対応できるようにするためである。

これを実装するにあたり、replaceBuffer のタイミングも変更した。以前のプログラムの内容では、「?x1 = A」と決まった後で、「?x2 = A」と制約を加える形をとっていた。それでは、?x2 = ?x1 の対応が取れない。(実装で内容を詳しく説明)

また、木構造を用いてこれらの再帰構造の内容を表現すると図 2 のようになる。図 2 は、目標「B on C, A on B」における「B on A」を単体目標としたときの再帰構造の様子である。1 階の「B on C」に対して、オペレータ「Place ?x on ?y」の add 部がマッチング


```

1 boolean varMatching(String vartoken, String token) {
2     System.out.println("vars="+vars);
3     System.out.println("vartoken="+vartoken);
4     System.out.println("token="+token);
5     if(vars.containsKey(vartoken)) {
6         if(token.equals(vars.get(vartoken))) {
7             return true;
8         }else{
9             return false;
10        }
11    }else{
12        /* ココじゃなくて...
13        replaceBuffer(vartoken, token);
14        if (vars.containsValue(vartoken)) {
15            replaceBindings(vartoken, token);
16        }
17        */
18        if(productKeyOnValue != null) {
19            //product_KeyValue の数
20            for(String str1 : productKeyOnValue.keySet()){
21                System.out.println("vars="+vars);
22                if(var(productKeyOnValue.get(str1))) {
23                    for(int num=0; num<prohibit.get(str1).size(); num++) {
24                        if(vars.containsKey(productKeyOnValue.get(str1))) {
25                            System.out.println(productKeyOnValue.get(str1));
26                            if(vars.get(productKeyOnValue.get(str1)).equals(
                                vartoken) & prohibit.get(str1).get(num).
                                equals(token)){
27                                System.out.println("禁止制約発動しました.");
28                                return false;
29                            }
30                        }
31                    }
32                }
33            }
34        }
35        //ココで行います！
36        replaceBuffer(vartoken, token);

```

上記のプログラムでは自分でも何をやっているのかわからなくなるので以下の図 2、図 3 を参考にしながらプログラムを見ていただきたい。

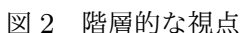


図 2 の各階のノードが決定された時点で、図 3 の productKeyOnValue にはそのノードの値が代入され、その後、選択されたオペレータの add 部 (X on Y に絞って言及中) と Matching をする。productKeyOnValue も prohibit も、HashMap なので、「X on Y」が「Key on Value」の関係で呼び出すことができる。1,2 階間でのマッチング処理は、varMatching メソッドでマッチングした?x0, ?y0 をそのまま prohibit の禁止制約と照らし合わせればよいが、3~5 階間でのマッチング処理は、図 2 のノードはあくまで、「B on ?y1」なので、?y2 との関係は「B on ?y1」と「?x2 on ?y2」のマッチングが終わった後に確定される。その関係はハッシュマップ:vars に格納されるので、それを用いて、ノードの「B on ?y1」を「B on ?y2」とする。そして、4~5 階で?y2=C とマッチングすると、

その情報を現状態のノード:productKeyOnValue に代入することで、「B on A」という関係を生成させる。それと禁止制約を照合することで、多階層の X on Y 関係でも対応できるようにした。プログラムの for 文では、productKeyOnValue の Key 値 (str1) と Value 値 (.get(str1)) を見て、その Key(str1) に対応する禁止制約の prohibit の Value 値 (.get(str2)) をもってきたのち、if の条件文で、HashMap:vars の ?y1=?y2 関係と禁止制約:prohibit の value 値を 1 つずつ照らし合わせる処理をしている。

そして以上の vars の関係 [?y1=?y2] の関係を見た後で、?y2=C とし、?y1=C とするところから、replaceBuffer メソッドの呼び出しは、上記の内容を行った後で実行するようにした。先に replaceBuffer メソッド呼び出してしまうと、Key:?y1 の Value 値は?y2 となってしまう、具体化されないまま終わってしまうからである。

2.3 実行例

今回の実行においては、上記の説明を実際に表示させるために、オペレータの選択を自分で選択し、逐一状況を確認していく。

ソースコード 3 3～5 階層のマッチング

```
1 Place ?x2 on ?y2
2 addList = [?x2 on ?y2, clear ?x2, handEmpty]
3 string1=B on ?y1
4 string2=?x2 on ?y2
5 productKeyOnValue 保存 = {B=?y1}
6 vars={?y0=C, ?x0=B, ?x1=B}
7 vartoken=?x2
8 token=B
9 productKeyOnValue.get()=?y1
10 vars={?y0=C, ?x0=B, ?x1=B}
11 vars={?y0=C, ?x0=B, ?x1=B, ?x2=B}
12 vartoken=?y1
13 token=?y2
14 productKeyOnValue.get()=?y1
15 vars={?y0=C, ?x0=B, ?x1=B, ?x2=B}
16 その 2
17 Key = B Value = ?y1
18 unify 成功
```

```

19 オペレータの具体化:Place B on ?y2
20 *** GOALS ***[clear ?y2, holding B]
21 **clear ?y2
22 string1=clear ?y2
23 string2=clear A
24 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
25 vartoken=?y2
26 token=A
27 productKeyOnValue.get()=?y1
28 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
29 禁止制約発動しました.
30 string1=clear ?y2
31 string2=clear B
32 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
33 vartoken=?y2
34 token=B
35 productKeyOnValue.get()=?y1
36 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
37 禁止制約発動しました.
38 string1=clear ?y2
39 string2=clear C
40 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
41 vartoken=?y2
42 token=C
43 productKeyOnValue.get()=?y1
44 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
45 theBinding{?y0=C, ?x0=B, ?y1=C, ?x1=B, ?y2=C, ?x2=B}

```

「?x2 on ?y2」関係から「clear C」に至るまでの処理を図 2,3 を参考にしながら見ていただけると、よくわかると思う。「B on A」という禁止制約が発動され、A, B ではマッチングが行われず、C のみで成功している。実際に、最後の Binding 情報では、?y1=C,...,?y2=C と正しく行われていることが分かる。

2.4 考察

図 1 の木構造からも分かるように、問題に上がっていた繰り返しの処理 (Place A on B, remove A from on top B の繰り返し) をなくすには、「pick up B from the table」を選

択するしかないということが分かる。

ただ、以下の実行結果を見てもらいたい。最終結果のみを記す。

ソースコード 4 A on C が生じてしまう

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

確かに、「A on C」という禁止制約は存在しないが、既に「B on C」が存在している状況で、「A on C」とすることは... 物理的に不可能である。そのため、実行中にも、禁止制約を随時追加していかなければならないことが分かる。ただ、「B on C」が確定したのちに「A on C」はダメという禁止制約、「remove B from on top C」を実行された後には、「A on C」はダメではないので禁止制約から取り除くといった処理を行うのは、また... 大変であり、planning の実装に向けては、かなりゲームフィールドの制限を強くするしかないことが分かった。”知識システム”の授業で、新谷先生が「条件を限定した仮想空間を用いないとプログラムに書ききらないよ」と言っていた内容が良く分かった。

3 必須課題 5-2 のさらなる改良

3.1 序論

上記では、禁止制約を「X on Y 関係」の HashMap を用いることで解決するアプローチを試みた。その際に、図 1 で紹介したが、「A をとって B の上に置く」、「A を B から取り除く」、「A をとって B の上に置く」、「A を B から取り除く」といった無駄な処理をなくす方法として、オペレータの選択に制限をかける必要があると述べた。今回はそのプログラムを実装してみる。

3.2 手法

図 1 でもわかるように, 2 階で「holdingB」という目標が設定されていて, その後, 4 階でも同じ「holdingB」という目標が出てくる. 4 階で 2 階と同じオペレータ (ここでは, remove X from on top Y) を選択したらどんどんと木は深くなる. そのため, planningAGoal メソッドで設定された 1 つの目標 (木構造におけるノード) と, そのノードに対して選択されたオペレータを保存しておき, 繰り返しを防ぐという方法をとる.

3.3 実装

Planner クラスに, 1 つの目標 (木構造におけるノード) とその時に選択されたオペレータの対応関係を保存する HashMap:goalOp を作成. Key には目標, Value にはオペレータのインデント [0~3] を保存する. (オペレータの選択とインデントの関係に関しては, 課題 5 のレポートを参照してもらいたい.)

ソースコード 5 目標とオペレータの対応関係の保存

```
1 //Planner クラス
2  HashMap<String, Integer> goalOp; //Key:単目標/Value:オペレータの番号
3 //コンストラクタ
4  goalOp = new HashMap<String, Integer>();
5
6 //planningAGoal メソッド内
7  //1.state リストとの照らし合わせ処理(省略)
8  //2.ノードをKeyとして保存(Value)は未定値
9  if(!goalOp.containsKey(theGoal)) {
10     goalOp.put(theGoal, -1);
11 }
12 //3.operator の選択(ランダム)
13 int randInt = Math.abs(rand.nextInt())\%operators.size();
14 Operator op = (Operator)operators.get(randInt);
15 cPoint = randInt;
16 //4-1.選択したオペレータの実行(一部省略)
17 Operator anOperator = rename((Operator) operators.get(cPoint));
18 for (int j = 0; j < addList.size(); j++) {
19     Unifier unification = new Unifier();
20     if(unification.unify(theGoal, (String) addList.get(j), theBinding,
```

```

        attributions.keyValueProhibit, p_productKeyOnValue)) {
21    //5.一度は選択されていたら,break
22    if(goalOp.get(theGoal) == cPoint) {
23        System.out.println("break します");
24        break;
25    }
26    //6.break しなかったら, 保存する
27    goalOp.put(theGoal, cPoint);
28    System.out.println("goalOp"+goalOp);
29    ...
30    //4-2.選択されたオペレータが実行されない場合,
31    for (int i = 0; i < operators.size(); i++) {
32        if(i != cPoint) {
33            anOperator = rename((Operator) operators.get(i));
34            addList = (ArrayList<String>) anOperator.getAddList();
35            for (int j = 0; j < addList.size(); j++) {
36                Unifier unification = new Unifier();
37                if(unification.unify(theGoal, (String) addList.get(j),
                    theBinding, attributions.keyValueProhibit,
                    p_productKeyOnValue)) {
38                    //5.既にその目標に対してオペレータが選択されていたら,
39                    if(goalOp.get(theGoal) == i) {
40                        System.out.println("break します");
41                        break;
42                    }
43                    //6.保存する
44                    goalOp.put(theGoal, i);
45                    System.out.println("goalOp"+goalOp);
46                    ...

```

選択したオペレータが実際に実行されるか, それとも別のオペレータが実行されるかに関しては, 課題5のレポートを参考にしてもらいたい.

planningAGoal メソッドが実行されたら, まず, HashMap:goalOp にその目標が Key として設定されていない場合は, Value を-1 としておく. それ以前にすでにその目標が Key として保存されていたら, -1 の初期化は行わない. その後, 実際に, オペレータの add 部と unify をしていき, 成功したら, そのオペレータのインデントを goalOp の Value に加える (上書き保存する). すでにその目標に対して, 以前実行されたオペレータがあり, 保

存されていたら (つまり-1 ではなかったら), 別のオペレータを探すために for 文を break し, 下の”選択されたオペレータ以外のオペレータの実行”へ移動する. そこで, 一度も選択されていないオペレータだったら, そのオペレータを実行する.

3.4 実行結果

ソースコード 6 goalOp によって制約される

```
1 [B on C, A on B]
2 *** GOALS ***[B on C, A on B]
3 **B on C
4 Place ?x0 on ?y0
5 goalOp{B on C=0}
6 *** GOALS ***[clear C, holding B]
7 **clear C
8 theBinding{?y0=C, ?x0=B}
9 *** GOALS ***[holding B]
10 **holding B
11 remove ?x1 from on top ?y1
12 goalOp{holding B=1, B on C=0}
13 オペレータの具体化:remove B from on top ?y1
14 *** GOALS ***[B on ?y1, clear B, handEmpty]
15 **B on ?y1
16 Place ?x2 on ?y2
17 goalOp{holding B=1, B on C=0, B on ?y1=0}
18 オペレータの具体化:Place B on ?y2
19 *** GOALS ***[clear ?y2, holding B]
20 **clear ?y2
21 theBinding{?y0=C, ?x0=B, ?y1=C, ?x1=B, ?y2=C, ?x2=B}
22 *** GOALS ***[holding B]
23 **holding B
24 Place ?x3 on ?y3
25 そのオペレータは実行できません
26 remove ?x4 from on top ?y4
27 break します
28 pick up ?x5 from the table
29 オペレータの具体化:pick up B from the table
30 *** GOALS ***[ontable B, clear B, handEmpty]
```

```

31 **ontable B
32 theBinding{?y0=C, ?x0=B, ?y1=C, ?x1=B, ?y2=C, ?x2=B, ?x4=B, ?x5=B}
33 *** GOALS ***[clear B, handEmpty]
34 **clear B
35 theBinding{?y0=C, ?x0=B, ?y1=C, ?x1=B, ?y2=C, ?x2=B, ?x4=B, ?x5=B}
36 *** GOALS ***[handEmpty]
37 **handEmpty
38 theBinding{?y0=C, ?x0=B, ?y1=C, ?x1=B, ?y2=C, ?x2=B, ?x4=B, ?x5=B}
39 Success !
40 Success !
41 Success !
42 ...

```

「holdingB」が2回目の目標として設定された際に、「Place ?x3 on ?y3」のオペレータを選択しているが、add 文には「holdingX」が存在しないので、次のオペレータを探す。すると、「remove ?x4 from on top ?y4」が選択されるが、goalOp にすでに holding B=1 と保存されているので、インデント 1 の remove オペレータは実行されない。その後、「pick up ?x5 from the table」が実行され、成功していることが分かる。

3.5 考察

このプログラムにより、オペレータの選択をランダムにしても、無駄な処理 (A を B の上に置いて、A を B から取り除いて、また A を B の上において... など) はなくなった。完璧なプログラムかといえば、結局、「Place A on C」という前に述べた”物理的に不可能な置き方”が残ってしまったので、残念である。また、”机に置いてある 3 つの積み木 (A, B, C) を順に上に積んでいく”という処理しかできなくなり、汎用性がなくなってしまった。

本当は積み木が 4 つになって、初期状態がどうであれ、最終的な目標状態がどうであれ、適切なオペレータ選択ができればよかったが、なかなか難しい...

4 感想

前回の内容が行き当たりばったりのプログラムであったので、今回の修繕にとっても時間を要した。具体的なエラーに対して対処しても次のエラーが生じて、全てのエラーを直すのに本当に時間がかかってしまった。汎用性の高いプログラムを作るには、もっとさまざまなパターンを頭の中でイメージしないといけないので、やはり難しい。

再帰構造のプログラムの処理の流れを把握するには、やはり木構造を用いるのが一番わかりやすいということが分かった。木構造を用いることができれば、2年生の前期に大園先生から学んだ”知能処理学”の内容や、加藤先生から学んだ”知識表現と推論”がとても生きてくることが分かり、今までの総復習をしながら今回の課題に取り組むことができてよかった。

先生に最後にお願いしたいこととしては、ぜひ次の世代の子にも、今回自分が取り組んだ Planning の木構造表現方法を勧めてほしい。欲を言えば、これを一つの課題にしてもらえると、バックトラックを用いたプランニングの全体概要が面白いようにわかると思う。この 3.3 章のプランニングの実行過程に関しては、”知能プログラミング入門”の教科書に詳しくのっていないなかったため、プランニングの内部構造をしっかりと理解できているのが、グループの中でも僕だけで、今回の課題もだいぶしんどかった。木構造で表現することで、もう少し、グループ内で、プランニングの処理の流れを共有し、役割分担しやすくなると思う。さらに言えば、今回自分は時間がなかったのでできなかったが、プログラム実行しながら、この木構造を表現する GUI があると、もっと直観的にわかりやすくなるとも思った。これは来年度の発展課題にってもらえると面白いかもしれない。

最後に、この知能系演習を通して、たくさんの経験を積むことができた。ここまで 6 つの課題を最後まで一緒に取り組んでくれたグループのメンバーには本当に感謝する。また、今まで教えてくださった知能系の先生方にも感謝申し上げます。

”知識”とは”データの関連”であり、”推論”とは、”知識を利用して結論を導くこと”である。この 3 年間で人工知能の本質を経験することができた。ありがとうございました。

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
- [2] 知識システムの実装基礎 –著：新谷 虎松
- [3] 人工知能の基礎知識 –著：太原 育夫
- [4] LATEX2 ϵ 美文書作成入門 –著：奥村 晴彦