知能プログラミング演習 II 課題 5

グループ 8
29114116 増田大輝

2019 年 12 月 2 日

■提出物　rep5, group08

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

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 5-3

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）

2.1　手法

ブロックの属性を扱えるようにするために, まずはルールによってブロックと属性の関係を表現し, それらの関係を保持する必要がある．その上で, 属性によって表現されたゴールや, 初期状態をブロックによる表現に変換し, 後ろ向き推論を行う方針とした．

ここで, ブロックの属性によってゴールを表現する場合, そのまま後ろ向き推論を行うと, 同一ブロックの異なる属性による表現がワーキングスペースに含まれる場合などへの対応が非常に複雑となる. したがって, 入力されたルールと初期状態はあらかじめ属性表現をブロックに変換することで, もとの後ろ向き推論が正常に機能するような実装を行うものとする.

2.2 実装

まず, 属性を扱うクラス Attributions を作成した. Attributions クラスの変数として HashMap¡String,String¿型の attributions,List¡String¿型の rules がある. それぞれ, 属性とブロックの関係 (属性が主キー) の集合, ルールの集合に対応する. この Attributions クラスには以下の 3 つのコンストラクタが存在し, 全て上記変数への操作を行うものであるが, それぞれに次のような用途がある.

Attributions() テスト用のデフォルトコンストラクタ. デフォルトのルールが用意されている. ルールは自然言語で記述.

Attributions(List <String>rules) 自然言語で記述されたルールをもとに属性とブロックの関係を構築する.

Attributions(HashMap <String, String>attributions) あらかじめ, 属性とブロックの関係が構築された状態のものを受け取る.

ただし, 自然言語で記述されているものは,"ブロック is 属性"の形で表現されているものを扱うものとする. 例えば,"A is blue"と記述されたものを受理する.

次に, 自然言語から属性とブロックの関係を構築するメソッド addAttribution(String attributionState) についての説明を行う. このメソッドは上記コンストラクタ Attributions() と Attributions(List <String>rules) から呼び出される. これらのコンストラクタからは rules の各要素を引数として渡すものとする. addAttribution メソッドは仮引数 attributionState でこれらを受け取り,"ブロック is 属性"を空白で分割したリストの第二要素が"is"である場合に,attributions に関係を加える. この時, 属性を主キーとし, ブロックを紐づけられるオブジェクトとする.
すなわち,"A is blue"とした場合には, 第二要素は"is"であるので"blue"を主キーとして,"A"と紐づけられることとなる.

以上により, 属性を持つブロックを扱うための下準備が整った.

続いて,attributions に格納された属性とブロックの関係をゴールリストや初期状態のリストに適応するためのメソッド editStatementList(ArrayList¡String¿ statementList) についての説明である. このメソッドでは, 仮引数として渡された statementList の各要素について,attributions 内に格納されている属性との照合作業が行われ, 属性がある場合には紐づけられたブロック名に置き換えられる作業が行われる. 以下にメソッドのソースコードを示す.

ソースコード 1　editStatementList メソッドの実装

```
1    ArrayList<String> editStatementList(ArrayList<String> statementList
         ) {
2                System.out.println("++++++ EditStatement ++++++");
3                ArrayList<String> newStatementList = new ArrayList();
4                for (String statement: statementList) {
5                        List<String> tokens = Arrays.asList(statement.
                             split(" "));
6                        String newStatement = "";
7                        for(int tokenNum = 0; tokenNum < tokens.size
                             (); tokenNum++) {
8                            String token = tokens.get(tokenNum);
9                            if(attributions.containsKey(token)) {
10                                   token = attributions.get(token
                                         );
11                           }
12                           newStatement += token;
13                           if(tokenNum < tokens.size()-1) {
14                                   newStatement += " ";
15                           }
16                   }
17                   newStatementList.add(newStatement);
18                   System.out.println(statement+" =====> "+
                         newStatement);
19               }
20               return newStatementList;
21       }
```

特に,9 11 行目で HashMap の機能を利用して属性からブロック名を導き出す処理を行っている.

以上によって, 自然言語で記述されたルール集合から導出された属性とブロックの関係や与えられた属性とブロックの関係を元にして, ゴールリストや初期状態のリストに含まれる属性表現をブロック名に統一する処理が完成した.

具体的な利用法は, 使用したい任意のコンストラクタによって Attributions クラスを初期化し, 以下のような形で, ゴールリストや初期状態のリストに適用すればよい.

ソースコード 2　editStatementList の利用

```
1    goalList = attributions.editStatementList(initAttributeGoalList());
2        initialState = attributions.editStatementList(
             initAttributeInitialState());
```

2.3　実行例

以下に,CUI 上での実行結果を示す. ただし, コンストラクタはデフォルトコンストラクタを使用するものとする.

ソースコード 3　ゴールと初期状態に属性を指定した場合の実行結果

```
1    ~/Programming2/Work5
2    ●java Planner【 re-fix/attribute-rule 】
3    ========== goal:green on ball ==========
4    ========== goal:blue on pyramid ==========
5    ++++++ EditStatement ++++++
6    green on ball =====> B on C
7    blue on pyramid =====> A on B
8    ---------- initInitialState:clear blue ----------
9    ---------- initInitialState:clear green ----------
10   ---------- initInitialState:clear red ----------
11   ---------- initInitialState:ontable box ----------
12   ---------- initInitialState:ontable pyramid ----------
13   ---------- initInitialState:ontable ball ----------
14   ---------- initInitialState:handEmpty ----------
15   ++++++ EditStatement ++++++
16   clear blue =====> clear A
17   clear green =====> clear B
```

```
18    clear red =====> clear C
19    ontable box =====> ontable A
20    ontable pyramid =====> ontable B
21    ontable ball =====> ontable C
22    handEmpty =====> handEmpty
23    *** GOALS ***[B on C, A on B]
24    **B on C
25    Place B on C
26    *** GOALS ***[clear C, holding B]
27    **clear C
28    [clear A, clear B, clear C, ontable A, ontable B, ontable C,
          handEmpty]
29    *** GOALS ***[holding B]
30    **holding B
31    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,
          handEmpty]
35    *** GOALS ***[clear B, handEmpty]
36    **clear B
37    [clear A, clear B, clear C, ontable A, ontable B, ontable C,
          handEmpty]
38    *** GOALS ***[handEmpty]
39    **handEmpty
40    pick up B from the table
41    Place B on C
42    [clear A, ontable A, ontable C, B on C, clear B, handEmpty]
43    *** GOALS ***[A on B]
44    **A on B
45    Place A on B
46    *** GOALS ***[clear B, holding A]
47    **clear B
48    [clear A, ontable A, ontable C, B on C, clear B, handEmpty]
49    *** GOALS ***[holding A]
50    **holding A
51    pick up A from the table
52    *** GOALS ***[ontable A, clear A, handEmpty]
```

```
53      **ontable A
54      [clear A, ontable A, ontable C, B on C, clear B, handEmpty]
55      *** GOALS ***[clear A, handEmpty]
56      **clear A
57      [clear A, ontable A, ontable C, B on C, clear B, handEmpty]
58      *** GOALS ***[handEmpty]
59      **handEmpty
60      pick up A from the table
61      Place A on B
62      ***** This is a plan! *****
63      pick up B from the table
64      Place B on C
65      pick up A from the table
66      Place A on B
```

まず,3・4 行目で属性によってゴールが表現されていること示している．その次に,5 7 行目でこれらのゴールを属性からブロック名による表現に変換している．続いて,8 14 行目で属性によって式状態が表現されていることを示している．先ほどと同様に,15 22 行目では, これらの表現を属性によるものからブロック名のみの形に変更している．したがって, 以降の実行結果はブロック名のみを用いた場合と同様となる．

2.4 考察

属性を扱うに当たって, 初めは属性のまま推論を行い, プランニングを実行できるようにしようと考えた．しかし, 同じブロックの属性であっても, 異なる属性同士において, 字面だけではマッチングが成功しないことに気が付いた．例えば”A is blue”と”A is box”がルールとして与えられていたとする．この時, ゴールが”blue on green”で, 状態として”box on green”が保持されているものとする．本来,blue と box は同じブロック A の属性であるが, 字面の上では, 異なる定数として認識される．

これらを同一のブロック A を指し示すものとして認識するには, 属性からブロックそのものへと変換する機能が必要である．したがって, 変換後のゴール”A on B”と状態”A on B”であればマッチングは成功する．

ここで, もう一度属性のまま推論を行う意義を考えることとした．上記の方法では, マッチングにおいて最終的に属性からブロック名への変換処理を行っている．ところが, 初めから同様に属性をブロック名に変換する手法にすることでもマッチングは可能である．例え

ば, あらかじめゴールを”blue on green”から”A on B”に変換し, 初期状態を全てブロック名による表現にすることで, プランニング中の状態を”A on B”とする. 結果として, 即座に”A on B”と”A on B”のマッチングが行われるようにする.
加えて, 後者の方がゴールリストと初期状態のリストから属性表現をブロック名に変換するのみであるので, 処理も非常にシンプルとなる. 後ろ向き推論の特徴と処理の簡潔さから, 後者の手法を採用することにした.
複数ある実現方法の中から, 既存のプログラムとの相性や論理的な汎用性を考えてもっとも良いと思える方法を選択できたと考えている.

3　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.

3.1　手法

ブロックワールド内における物理的制約条件（以下、禁止制約と表現する）をルールとして表現し, プランニングに反映するために, 課題 5-3 で作成した属性に関する処理を扱うクラス Attributions 内に禁止制約のルールを保持させる方法を考案した. 例えば,”box on pyramid”（三角錐の上に正四面体がある）のようなルールが物理法則に反する禁止制約に当たる.
これらの禁止制約のルール集合を, 課題 5-3 で結びつけた「属性とブロックの関係」をもとに, 属性を用いた表現からブロックによる表現に置き換える.
以上により, ゴールリストやオペレーター適用後の状態とブロックによって表現される禁止制約を比較することで, 物理法則に反する状態を削除することができる. また, ゴールリストに物理法則に反するものが含まれる場合には, ゴール不成立とする.

3.2　実装

まず, 属性を扱う Attributions クラス内に ArrayList¡String¿型の禁止制約のルール集合 prohibitRules を保持させる. この prohibitRules はコンストラクタから呼び出される

メソッド addProhibitRules() により禁止制約のルールが追加される.
加えて, 属性によって表現された禁止制約のルールをブロックによる表現に置き換えたルール集合 prohibitBlockStates も定義する. prohibitRules から prohibitBlockStates に変換を行うに当たっては, 課題 5-3 で作成したメソッド editStatementList を用いる.
次に, 状態と禁止制約を比較するメソッド checkProhibitBlockState(String state) を実装した. このメソッドを以下に示す.

ソースコード 4　checkProhibitBlockState メソッドの実装

```
1    private Boolean checkProhibitBlockState(String state) {
2                for(String prohibitBlockState: prohibitBlockStates) {
3                        if(prohibitBlockState.equals(state)) {
4                                System.out.println("【Warning!:状態"+
                                     state+"は禁止制約です！！】");
5                                return false;
6                        }
7                }
8                return true;
9        }
```

2 7 行目に渡って, 全ての禁止制約のルール集合の要素と仮引数で受け取った状態を比較する処理を行なっている. ここで, 一つでも禁止制約と一致すると,false が返却される. 反対に, 全ての禁止制約と一致しなかった場合には,true が返却される.
続いて, この checkProhibitBlockState メソッドを利用する checkStates(ArrayList¡String¿ states) メソッドの説明を行う. このメソッドについても以下に示す.

ソースコード 5　checkStates メソッドの実装

```
1    ArrayList<String> checkStates(ArrayList<String> states) {
2                ArrayList<String> checkedStates = new ArrayList<String
                     >();
3                for(String state: states) {
4                        if(checkProhibitBlockState(state)) {
5                                checkedStates.add(state);
6                        }
7                }
8                return checkedStates;
9        }
```

3 7 行目に渡って, 仮引数で受け取った状態集合の各要素を checkProhibitBlockState の引数として渡すことで, 禁止制約のチェックを行なっている. 結果として true が帰ってきた場合にはチェック済みの状態集合 checkedStates に格納し,false の場合には格納されない.
全ての要素のチェックが終わると,checkedStates を返却する.
checkedStates メソッドを, 変化直後の状態集合に対して使用することによって, 物理法則に反する状態を削除することができる. 以上の実装により, 禁止制約のブロックワールドへの反映を実現した.

3.3 実行例

以下に,CUI 上での実行結果を示す. ただし, コンストラクタはデフォルトコンストラクタを使用するものとする.

ソースコード 6　禁止制約によりゴールが成立しなくなった場合

```
1  ~/Programming2/Work5
2  ●java Planner【 feature/prohibit-rules 】
3  ###### Add prohibitRule ######
4  ****** ProhibitRule:box on pyramid ******
5  ****** ProhibitRule:box on ball ******
6  ****** ProhibitRule:ball on pyramid ******
7  ****** ProhibitRule:pyramid on ball ******
8  ****** ProhibitRule:trapezoid on pyramid ******
9  ****** ProhibitRule:trapezoid on ball ******
10 ++++++ EditStatement ++++++
11 box on pyramid =====> A on B
12 box on ball =====> A on C
13 ball on pyramid =====> C on B
14 pyramid on ball =====> B on C
15 trapezoid on pyramid =====> trapezoid on B
16 trapezoid on ball =====> trapezoid on C
17 ========== goal:green on ball ==========
18 ========== goal:blue on pyramid ==========
19 ++++++ EditStatement ++++++
20 green on ball =====> B on C
21 blue on pyramid =====> A on B
22 【Warning!:状態B on C は禁止制約です！！】
```

```
23      【Warning!:状態A on B は禁止制約です！！】
24      ---------- initInitialState:clear blue ----------
25      ---------- initInitialState:clear green ----------
26      ---------- initInitialState:clear red ----------
27      ---------- initInitialState:ontable box ----------
28      ---------- initInitialState:ontable pyramid ----------
29      ---------- initInitialState:ontable ball ----------
30      ---------- initInitialState:handEmpty ----------
31      ++++++ EditStatement ++++++
32      clear blue =====> clear A
33      clear green =====> clear B
34      clear red =====> clear C
35      ontable box =====> ontable A
36      ontable pyramid =====> ontable B
37      ontable ball =====> ontable C
38      handEmpty =====> handEmpty
39      禁止制約によってゴールが成立しなくなりました
```

はじめに,3 9 行目で禁止制約のルールが prohibitRules に追加されている. 次に,10 16 行目で editStatementList によって属性による表現からブロックによる表現に変換されている. 17,18 行目で属性表現によって指定されたゴールが 19 21 行目でブロックによる表現に変換されている.

そして,22,23 行目で禁止制約がゴールに使用されていることが検出されている. 詳しく見てみると,22 行目の検出は 14 行目と 20 行目の変換結果が一致することによるものであり, 23 行目の検出は 11 行目と 21 行目の変換結果が一致することによるものである. これらによって, ゴールリストに物理法則に反するものが含まれていることが判明したため,39 行目においてメッセージが表示され, プランニングが終了している.

3.4 考察

課題 5-3 の実装をもとに禁止制約をブロックワールドに反映する実装を行なった. 特に,editStatementList のように再利用することのできるメソッドを作成したことは大きなプラスとなった. また, 課題 5-3 で取り決めた, 属性をブロック名に置き換えてからプランニングを行う方針は, 課題 5-5 を進めるに当たって大きな助けにもなった. 具体的には,”A is red”,”A is pyramid”,”B is blue”,”B is ball”のようなルールのもと, ”pyramid on ball”のような禁止制約があるとする. このとき,”red on blue”が状態に含まれるとき,

属性表現のままプランニングを行うと見逃してしまうことになるが, 先に状態と禁止制約の属性表現をブロック名に変換して,”A on B”とすることによって, 物理法則に反する状態であると判定することができる.
したがって, 課題 5-3 で取り決めた方針によって, 課題 5-5 のプログラムの質を向上させることができたと考えている.
以上より, より良い実装を目指すに当たって, あらかじめ頭の中でプログラムの構造や処理の手順を考えておくことは非常に効果的であると感じた.

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