知能プログラミング演習II 課題4

グループ8
29114003 青山周平

2019年11月11日

提出物　rep4

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	20
29119016	小中祐希	20

1　課題の説明

必須課題 4-1 まず，教科書3.2.1の「前向き推論」のプログラムと教科書3.2.2の「後向き推論」のプログラムとの動作確認をし，前向き推論と後ろ向き推論の違いを説明せよ．また，実行例を示してルールが選択される過程を説明せよ．説明の際には，LibreOfficeのDraw（コマンド soffice –draw）などのドロー系ツールを使ってp.106 図3.11やp.118 図3.12のような図として示すことが望ましい．

必須課題 4-2 CarShop.data，AnimalWorld.data 等のデータファイルを実際的な応用事例（自分達の興味分野で良い）に書き換えて，前向き推論，および後ろ向き推論に基づく質問応答システムを作成せよ．どのような応用事例を扱うかは，メンバーで話し合って決めること．

なお，ユーザの質問は英語や日本語のような自然言語が望ましいが，難しければ変数を含むパターン等でも可とする．

必須課題 4-3 上記 4-2 で実装した質問応答システムの GUI を作成せよ．質問に答える際の推論過程を可視化できることが望ましい．

発展課題 4-4 上記 4-3 で実装した GUI を発展させ，質問応答だけでなく，ルールの編集（追加，削除，変更）などについても GUI で行えるようにせよ．

2 必須課題 4-3

> 上記 4-2 で実装した質問応答システムの GUI を作成せよ．質問に答える際の推論過程を可視化できることが望ましい．

私の担当箇所は，必須課題 4-3 における GUI の，Swing による実装である．

2.1 手法

まず，前向き推論のための GUI を実装するにあたり，以下のような方針を立てた．

1. 推論のルール・推論結果を受け取る．
2. 推論過程を可視化するためのパネルを作る．

1．に関して，MVP アーキテクチャを導入し，Presenter を他の班員に作ってもらい，それを介してデータを受け取ることで，Model である RuleBaseSystem.java との独立性の高い GUI を設計できるような仕様とした．

2．に関して，表示パーツの部品化や配置を工夫することで，ユーザにとって統一的な表示がなされるように心がけた．

次に，後向き推論のための GUI を実装するにあたり，以下のような方針を立てた．

1. 前向き推論のクラスを部品化して活用する．

1. に関して，前向き推論で制作したFwdChainTableクラスをChainTableクラスで抽象化することで，プログラム全体の多層化を高めた．また，これにより前向き推論と後向き推論の互換性を高めることによって，プログラム全体がまとまりを持つような仕様とした．

2.2 実装

双方の推論に共通するプログラムChainGUI.java，StepPanel.javaには以下のクラスが含まれる．

ChainGUI メニューパネルといった，FwdChainGUI，BwdChainGUIの共通部分を管理するための親クラス．

ChainTable FwdChainTable，BwdChainTableを管理するための抽象クラス．

StepPanel 導出の各ステップを表示するためのクラス．

EdgePanel 導出のステップ間の繋がりを管理するためのクラス．

前向き推論のGUIに関するプログラムBwdChainGUI.javaには以下のクラスが含まれる．

FwdChainGUI メソッドmainを実装した，フレームを実装するためのクラス．

FwdChainTable 前向き推論のアサーション・ルール・過程などを管理・表示するパネルを実装するためのクラス．

後向き推論のGUIに関するプログラムBwdChainGUI.javaには以下のクラスが含まれる．

BwdChainGUI メソッドmainを実装した，フレームを実装するためのクラス．

BwdChainTable 後向き推論のアサーション・ルール・過程などを管理・表示するパネルを実装するためのクラス．

2.2.1 推論のルール・推論結果を受け取る

推論を行う上で，RuleBaseSystem.java とデータのやりとりをして前提条件の引き渡しと結果の受け取りをできるようにする必要があった．そこで，他の班員に GUI と RuleBaseSystem.java との仲介を行うプログラム Presenter.java を制作してもらい，FwdChainGUI では，Presenter 型のフィールド pres を用いて以下のようにデータの受け渡しをコンストラクタに組み込んだ．

ソースコード 1: FwdChainTable クラスのコンストラクタ

```
1    FwdChainTable(String fileName) {
2        super(fileName);
3        view = new View();
4        pres = new Presenter(view);
5        pres.start(new ArrayList<String>(), fileName);
6        ...
7    }
```

データの受取についても同様に行った．例として質疑応答の結果を受け取るプログラムをソースコード 2 に示す．

ソースコード 2: FwdChainTable クラスのメソッド schStep

```
1    void schStep(ArrayList<String> astList, ArrayList<
        String> schAst) {
2        ...
3        pres.restart(astList);
4        ArrayList<StepResult> stepList = pres.stepResult
            (); // 全ルートの受け取り
5        paintPnls(stepList);
6
7        if (schAst.get(0).equals("")) {
8            System.out.println("WARNING: 検索文の格納に失
                敗");
9        } else {
10           ArrayList<ArrayList<SearchStep>> resList =
                pres.searchAssertion(schAst);    // 質疑応
                答結果の受け取り
11           answerPnls(resList.get(0));
12       }
13   }
```

これらのデータの受取の実装は，Presenter の制作と同時並行して行ったため，Presenter に渡す変数の個数や型，Presenter から受け取る形などに実装を窮したが，密に班員とコニュニケーションを取ることにより，細かい仕様をすり合わせて実装までですることができた．

2.2.2 推論過程を可視化するためのパネルを作る．

推論における各ステップについて StepPanel インスタンスを作り，StepPanel 間の繋がりを表示するために EdgePanel を作ることで，推論の可視化を行った．

StepPanel のコンストラクタは，以下に示すように 2 種類のコンストラクタを持つ．

ソースコード 3: StepPanel クラスのコンストラクタ 2 種

```
1     StepPanel(Assertion ast) {
2         setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS
              ));
3         setBackground(Color.ORANGE);
4         setBorder(new BevelBorder(BevelBorder.RAISED));
5         setSize(200, 30);
6         setLocation(10, 10);
7
8         alabel = new JLabel(ast.getName());
9         add(alabel);
10    }
11
12    StepPanel(StepResult step) {
13        setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS
              ));
14        setBackground(Color.ORANGE);
15        setBorder(new BevelBorder(BevelBorder.RAISED));
16        setSize(200, 150);
17        setLocation(10, 10);
18
19        Rule apply = step.getApply();
20        if (apply != null) {
21            JPanel rpanel = new RulePanel(apply);
22            add(rpanel);
23        }
24
```

```
25          JLabel alabel = new JLabel(" => " + step.
                getSuccess().getName());
26          add(alabel);
27      }
```

これらはインスタンス生成時に渡される引数の型によって挙動が異なることを意味する．すなわち，1 つ目のインスタンスでワーキングメモリ用の StepPanel と，ルールとそのルールから得られた結果である推論過程を，同じ StepPanel クラスで同一的に管理できるような仕様としたことが分かる．

これにより，EdgePanel の管理も容易になる．StepPanel の位置関係のみから，それらの繋がりを表示するための StepPanel のコンストラクタと描写に関するメソッドを以下に示す．

ソースコード 4: StepPanel クラスのコンストラクタと paintComponent，paintArrows メソッド

```
1       EdgePanel(StepPanel topPnl, StepPanel btmPnl) {
2           this.topPnl = topPnl;
3           this.btmPnl = btmPnl;
4           grace = 10;
5           pass = false;
6           topPoint = new Point(topPnl.getX() + (topPnl.
                getWidth() / 2), topPnl.getY() + topPnl.
                getHeight());
7           btmPoint = new Point(btmPnl.getX() + (btmPnl.
                getWidth() / 2), btmPnl.getY());
8
9           int lpX = getLeft();
10          int lpY = getTop();
11          int lpWidth = getRight() - lpX;
12          int lpHeight = getBtm() - lpY;
13          setBounds(lpX, lpY, lpWidth, lpHeight);
14
15          setOpaque(false); // パネルの透過
16      }
17
18      @Override
19      public void paintComponent(Graphics g) {
20          super.paintComponent(g);
21          Graphics2D g2d = (Graphics2D) g;
22          paintArrows(g2d);
```

```
23      }
24
25      void paintArrows(Graphics2D g) {
26          int fromX = topPoint.x;
27          int fromY = topPoint.y;
28          int toX = btmPoint.x;
29          int toY = btmPoint.y;
30
31          // 相対座標の考慮
32          int relX = getLeft();
33          int relY = getTop();
34
35          if (pass) {
36              g.setColor(Color.RED);
37          } else {
38              g.setColor(Color.BLUE);
39          }
40          BasicStroke stroke = new BasicStroke(2.0f);
41          g.setStroke(stroke);
42          g.drawLine(fromX - relX, fromY - relY, toX - relX
                , toY - relY);
43      }
```

描写に関して，Graphics2D クラスを活用することで，より太く見やすい線の描写が実現されている．

次に，各 StepPanel をどのような位置関係で配置するかを FwdChainGUI 側で設定する必要があった．そこで，そのためのメソッド paintPnls,tracePar を以下のように作った．

ソースコード 5: FwdChainGUI クラスのメソッド paintPnls，tracePar

```
1       void paintPnls(ArrayList<StepResult> srList) {
2           stepMap.clear();
3           edgeMap.clear();
4           locMap.clear();
5
6           for (StepResult sr : srList) {
7               tracePar(sr);
8           }
9       }
10
11      void tracePar(StepResult sr) {
12          if (sr.getAddSR() != null) {
```

```
13              for (StepResult from : sr.getAddSR()) {
14                  tracePar(from); // 再帰
15              }
16              StepPanel sp = stepMap.get(sr);
17              if (sp == null) {
18                  sp = new StepPanel(sr);
19                  stepMap.put(sr, sp);
20                  add(sp);
21
22                  Integer locY = stepMap.get(sr.getAddSR().
                        get(0)).getY() + next;
23                  Integer preX = locMap.get(locY);
24                  Integer locX = 10;
25                  if (preX != null) {
26                      locX = preX + next;
27                  }
28                  locMap.put(locY, locX);
29                  sp.setLocation(locX, locY);
30              }
31
32              for (StepResult from : sr.getAddSR()) {
33                  StepPanel fromPnl = stepMap.get(from);
34                  EdgePanel ep = new EdgePanel(fromPnl, sp);
35                  edgeMap.put(sr, ep);
36                  add(ep);
37              }
38          } else {
39              if (!stepMap.containsKey(sr)) {
40                  StepPanel sp = new StepPanel(sr.getSuccess
                        ()); // ここで作ったり
41                  stepMap.put(sr, sp);
42                  add(sp);
43
44                  Integer locY = 10;
45                  Integer preX = locMap.get(locY);
46                  Integer locX = 10;
47                  if (preX != null) {
48                      locX = preX + next;
49                  }
50                  locMap.put(locY, locX);
51                  sp.setLocation(locX, locY);
52              }
```

```
53            }
54        }
```

tracePar メソッドから分かるように，この処理は再帰的に行われている．すなわち，導出の各ステップを保持する StepResult 型の変数がフィールドに持つ，自身のアサーションがどこから生成されたかのリンクを用いて，初期値で渡されるワーキングメモリまで辿られていることが分かる．すなわち，導出の順番と可視化される配置が同じになるように処理がなされている．また，パネル同士が重ならないように，locMap というマップで座標の管理がなされている．

また，stepMap や edgeMap では，同じ推論過程に関する StepPanel や EdgePanel が重複して生成されないかの管理が行われている．

2.2.3　前向き推論のクラスを部品化して後向き推論に活用する．

後述する発展課題 4-4 でも述べるが，ここでは特に StepPanel.java の資源活用について述べる．元々は後向きでも前向きでも同一の StepPanel.java で動作するように設計していたのだが，Presenter 側の抽象化が難しかったので，前向き・後向きそれぞれで StepPanel.java を持つこととした．

しかし，その設計の名残の恩恵もあり，Presenter の型と用いるメソッドの引数や名前を変えるだけで，他はほとんど手を加えずに後向き推論の実装もできた．

2.3 実行例

FwdChainGUI を実行したところ，下図のような画面が得られる．

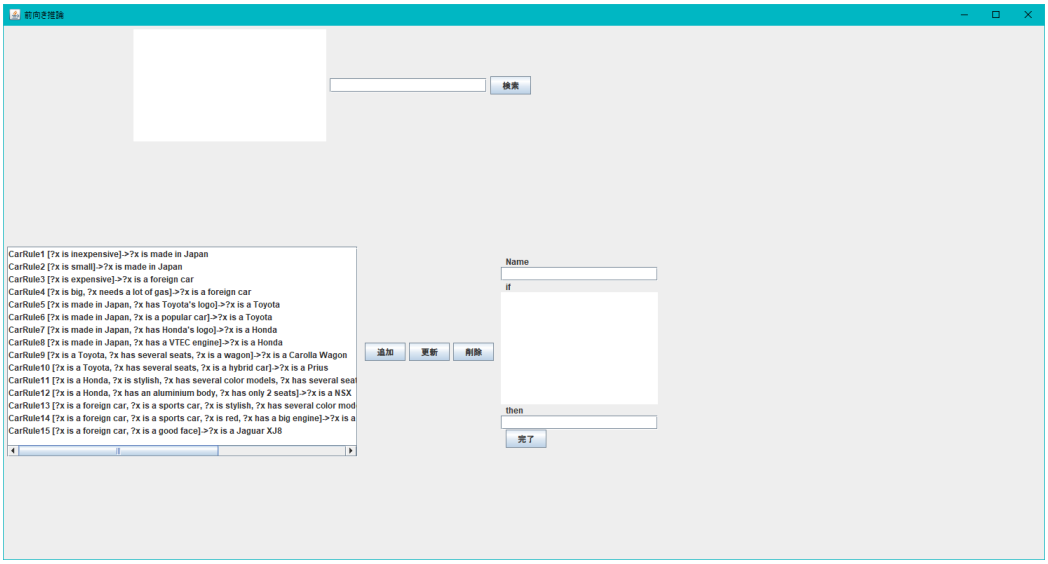

図 1: FwdChainGUI 初期状態

FwdChainGUI を実行したところ，下図のような画面が得られる．

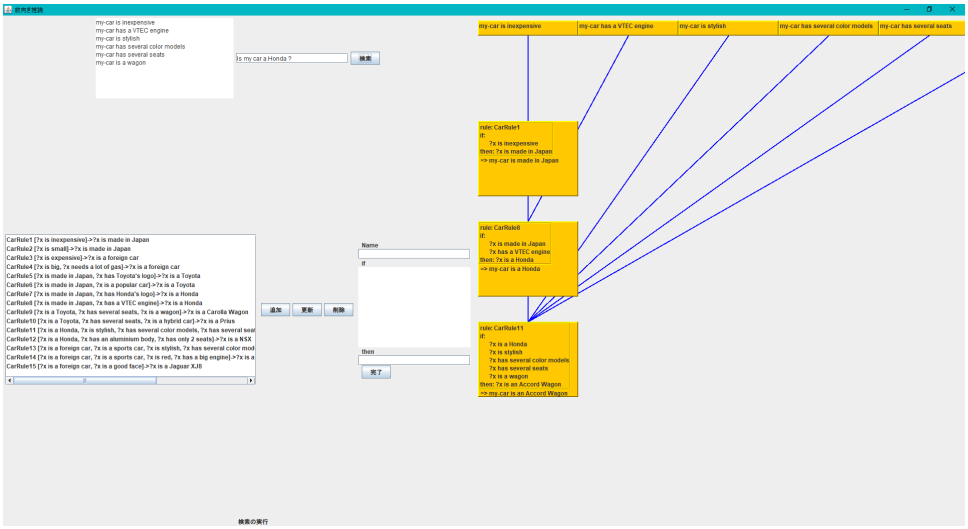

図 2: 前向き推論過程を可視化

BwdChainGUI を実行したところ，下図のような画面が得られる．

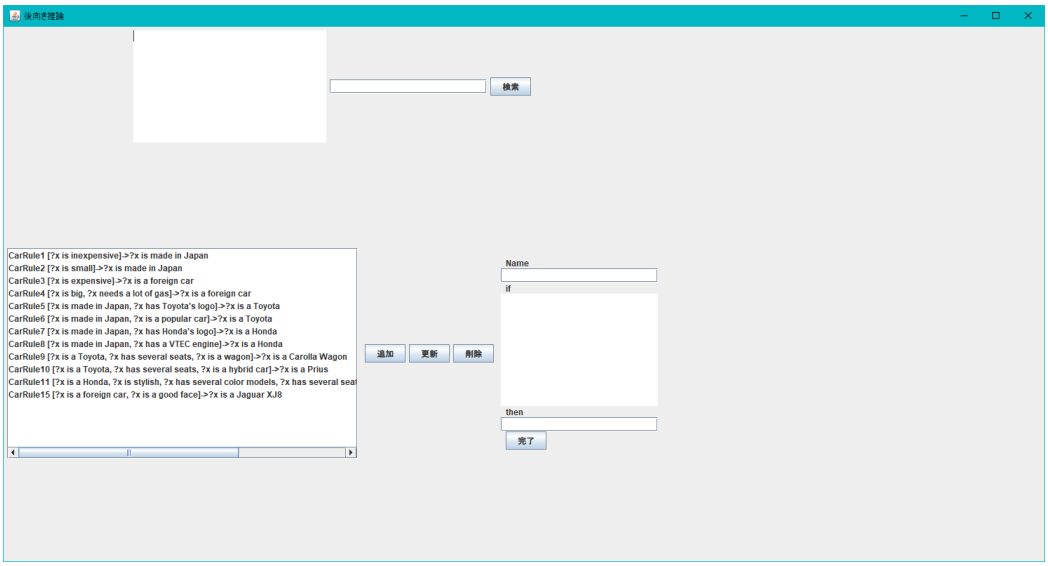

図 3: BwdChainGUI 初期状態

BwdChainGUI を実行したところ，下図のような画面が得られる．

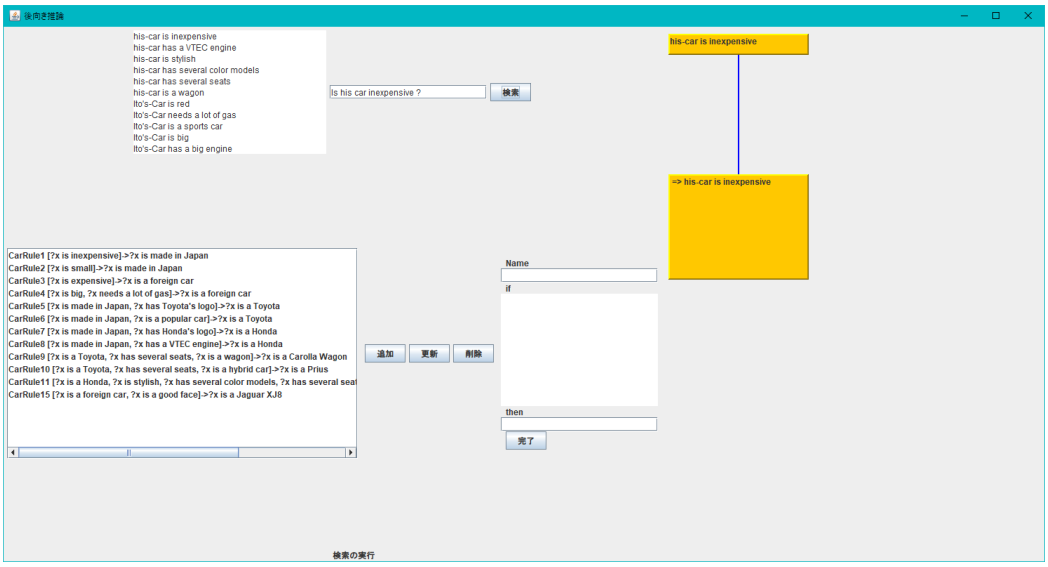

図 4: 後向き推論過程を可視化

2.4　考察

GUI から見た動作としては，前向き推論も後向き推論も似たようなものだと考えられたため，今回の課題ではクラスの抽象化をテーマにして取り組んでみた．初めは ChainGUI や ChainTable といった抽象クラスにおける前向き・後向きで共通となるであろうメソッドの抽出も出来ていたが，いざ Presenter と組み合わせて動作させようとなったときに，Presenter 側は抽象化のことを考えていなかったため，後で Presenter の抽象化も試みてはみたが，メソッド名や引数・戻り値の微妙な違いの修正が大変だったため，妥協することとなった．

この経験から，ある程度出来上がってきたものを後で抽象化することは難しく，上手に抽象化するには設計段階でどの部分を抽象化するかを考えてから実装に移る必要があると分かった．以後の課題では，それを考慮して抽象化を活用してゆきたいと思う．

今回の課題は今までと取り組み方が異なり，一つ機能を完成させたら次の機能に取り掛かるというアプローチではなく，全体的に未完成なまま作り上げてくというようなアプローチになってしまった．Presenter とのデータの受け渡しが必要だったからやむを得ずということもあったが，今回のアプローチは進捗管理やテスト動作のしづらさから見てもあまり良くなかったと考えられる．

１つの未完成のまま次の機能に取り掛かるにしても，最低限動作するラインまでは次の機能に移ってはいけないと考えられる．そのために，Presenter 担当者にも一つの機能の動作の担保を優先して実装してもらうような連携を取り合う必要があったと反省している．

3　発展課題 4-4

> 上記 4-3 で実装した GUI を発展させ，質問応答だけでなく，ルールの編集（追加，削除，変更）などについても GUI で行えるようにせよ．

私の担当箇所は，発展課題 4-4 におけるルールの編集機能の，Swing による実装である．

3.1　手法

ルールの編集機能を実装するにあたり，以下のような方針を立てた．

1. ルールの一覧表示と編集のためのメニューパネルを作る．

2. 追加・更新のためのパネルを作る．

1．に関して，JList クラスを用いてルールの一覧表示を行うことで，ユーザが更新・削除したいルールの選択を視覚的に行えるような仕様とした．

2．に関して，追加・更新のための専用のパネルを用意することで，直感的な編集操作を行えるような仕様とした．

3.2　実装

実装に関連するプログラムは，必須課題 4-3 と同様である．

3.2.1　ルールの一覧表示と編集のためのメニューパネルを作る

ルールの表示や編集については，前向き推論においても後向き推論においても共通の処理であるため，FwdChainGUI,BwdChainGUI の抽象クラスである ChainGUI を活用して，実装を行った．

まず，ChainGUI に内部クラスとして MenuPanel を実装した．MenuPanel では，JTextFIeld や JTextArea から入力を受け取り，反映するための各ボタンを実装した．また，JList を用いたルール一覧の表示や，後述する RuleEditor の管理等を行っている．

MenuPanel におけるボタンが押されたときの動作を表すメソッド actionPerformed をソースコード 6 に示す．

ソースコード 6: RuleEditor クラスのメソッド actionPerformed

```
1         @Override
2         public void actionPerformed(ActionEvent e) {
3             String cmd = e.getActionCommand();
4
5             if (cmd.equals("検索")) {
6                 ArrayList<String> astList = new ArrayList
                      <>(Arrays.asList(wmTA.getText().split
                      ("\n")));
```

```
7                   // ArrayList<String> schAst = new
                        ArrayList<>(Arrays.asList(schTA.getText
                        ().split("\n"))); // 複数の質問文のと
                        き?
8                   ArrayList<String> schAst = new ArrayList
                        <>();
9                   schAst.add(schTF.getText());
10
11                  ctable.schStep(astList, schAst);
12              } else {
13                  if (cmd.equals("追加")) {
14                      re.addRule();
15                  }else if (cmd.equals("更新")) {
16                      if (!rulePnl.isSelectionEmpty()) {
17                          int index = rulePnl.
                                getSelectedIndex();
18                          Rule val = (Rule) rulePnl.
                                getSelectedValue();
19                          re.udRule(val);
20                      } else {
21                          System.out.println("更新失敗(未選択
                                のため) ");
22                      }
23                  } else if (cmd.equals("削除")) {
24                      if (!rulePnl.isSelectionEmpty()) {
25                          int index = rulePnl.
                                getSelectedIndex();
26                          Rule val = (Rule) rulePnl.
                                getSelectedValue();
27                          ctable.rmRule(val);
28                      } else {
29                          System.out.println("削除失敗(未選択
                                のため) ");
30                      }
31                  }
32              }
33              status.setText(cmd + "の実行");
34          }
```

status ラベルの setText が最後に呼び出されることによって，RuleEditor のメソッド呼び出しより後に，この actionPerformed メソッドが終了するため，RuleEditor におけるルール一覧への反映を瞬時に行えるようになっ

ている．

3.2.2 追加・更新のためのパネルを作る

MenuPanel の内部クラスとして RuleEditor を実装した．

Rule は name 部，if 部と then 部のそれぞれについて編集を可能とする必要があり，RuleEditor では Rule の追加と編集におけるそれらの包括的な処理が行われるような仕様とした．各フィールドが入力され，完了ボタンが押されたときに，RuleBaseSystem に反映を依頼するメソッド actionPerformed は以下のようになる．

ソースコード 7: RuleEditor クラスのメソッド actionPerformed

```
1            @Override
2            public void actionPerformed(ActionEvent e) {
3                String nameText = nameTF.getText();
4                ArrayList<String> ifList = new ArrayList
                     <>(Arrays.asList(ifTA.getText().split
                     ("\n")));
5                String thenText = thenTF.getText();
6
7                if(rule == null) {
8                    ctable.addRule(nameText, ifList,
                         thenText);
9                } else {
10                   ctable.udRule(new Rule(nameText,
                         ifList, thenText));
11               }
12               udRuleMod();
13           }
```

ctable は ChainTable 型のメソッドであり，前向き・後向きそれぞれの ChainTable から Presenter を呼び出して反映が行われることが分かる．

また，udRuleMod メソッドは以下のようになる．

ソースコード 8: MenuPanel クラスのメソッド udRuleMod

```
1        void udRuleMod() {
2            ruleList = ctable.getRules();
3            ruleMod.clear();
4            for (Rule r : ruleList) {
5                ruleMod.addElement(r);
```

```
6                 }
7             }
```

これにより，GUI 上でのルール一覧の表示の更新が行われている．

3.3　実行例

FwdChainGUI を実行したところ，以下のような画面が得られる．なお，BwdChainGUI でも同様の動作をする．

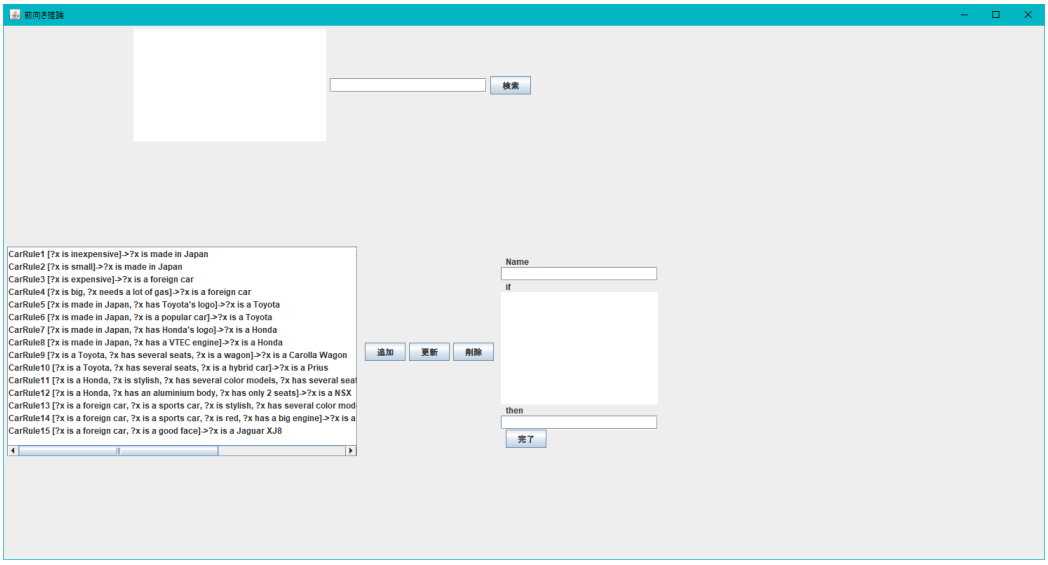

図 5: 初期状態

Rule5 を選択し，削除ボタンを押したところ以下のような画面が得られる．

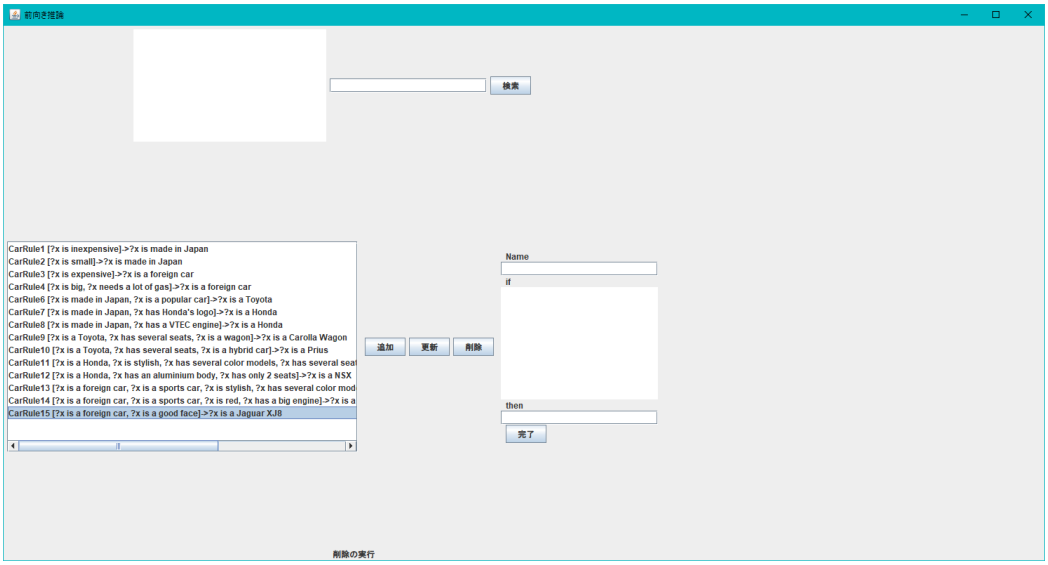

図 6: 削除の実行

追加ボタンを押し，テキストを入力後，完了ボタンを押したところ以下のような画面が得られる．

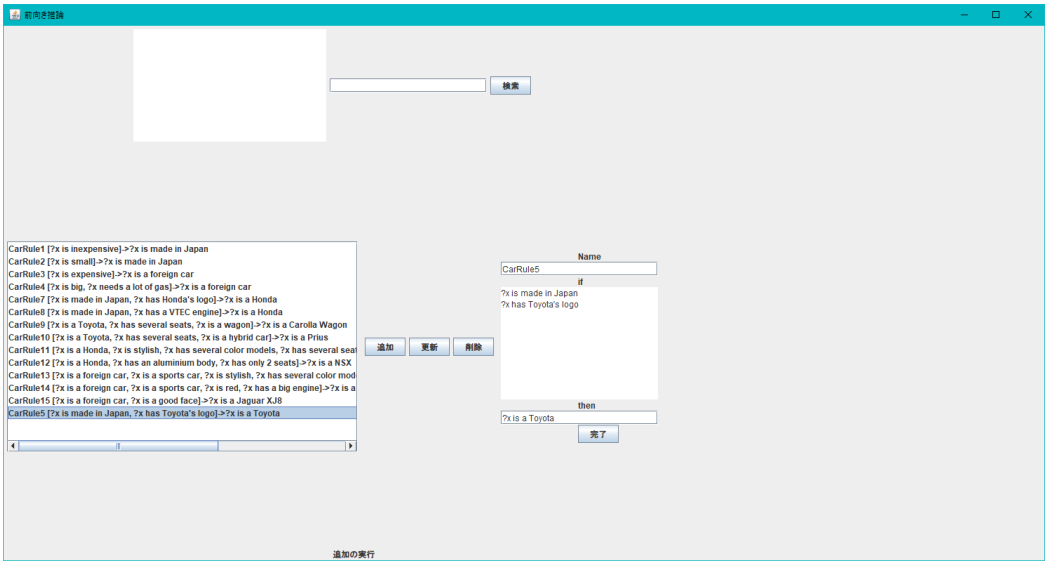

図 7: 追加の実行

Rule3 を選択して更新ボタンを押し，if のテキストに”?x is big” を加え，完了ボタンを押したところ以下のような画面が得られる．

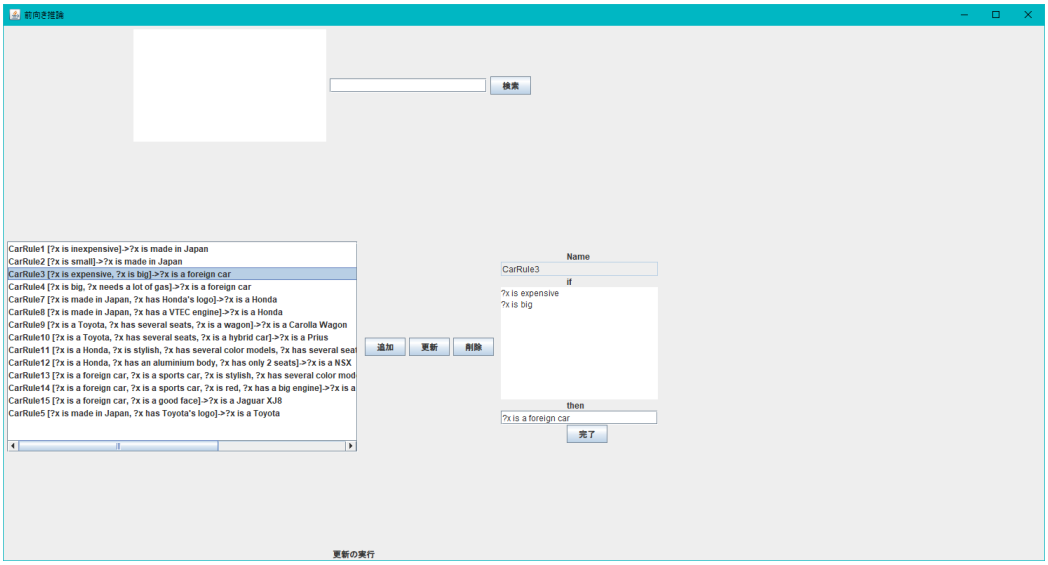

図 8: 更新の実行

3.4 考察

前回までの課題で JList や JButton を組み合わせた追加・削除は経験しているため，実装はあまり大変ではなかった．

しかし，レイアウトの設定が中々上手にできず，意図とは異なった挙動も所々見られた．例えば，MemuPanel で getContentPane().add メソッドを利用すると，MenuPanel のインスタンスを生成した元のフレームにおいてそれが反映されてしまうことや，同じく MenuPanel において，BoxLayout を用いて setLayout を行うと，エラーが吐かれてしまうことなどである．

これらの問題はいずれも，Layout と Container の違いを正しく理解できていなかったためだと考えられる．おそらくペインを用いた配置については，JPanel ではなく JFrame からしか使えないためだと考えられる．やはり Swing においても基礎から段階的に学んでいかないと，こういった場面でものの違いが分からずに躓くことが多くなると感じた．

また，イベントモデルについてもあまり理解せずに使っている節がある．例えばルールの更新をしたあとの udRuleMod メソッド呼び出しが，ボタンを連打しないと実行されないように見えたりする，という問題が挙げられる．これは実装でも述べたように，status ラベルの setText メソッドを udRuleMod 後に実行するような仕様とすることで解決したが，これは Swing 特有の問題ではなく，まさにマルチスレッド処理における問題であったことにそこで初めて気づいた．イベントモデルのようなメソッドをただ便利だからと使うだけでなく，裏でどんな処理がなされているかを考えることが，問題解決の一つの手段となることを気づいたと同時に，やはりイベントモデルについても基礎から学んでおかないと，解決できてもその場限りのものとなり，自分の中で応用できないものとなりうることに気づいた．

4 感想

今回の課題では前回の反省も活かして，班員との仕様の打ち合わせは早い段階から行っていたが，前回よりも実装は大変なものとなった．これは RuleBaseSystem の構造や挙動が前回の課題と比較しても複雑であるため，そこに外部からデータを取り込んだり，内部からデータを取り出すのは大変であったからだと考えられるが，その他にも班員との意識のすれ違いがあったことが大きな要因として挙げられると考えられる．

例えば，GUI の実装側からすると，推論過程を線で繋げて表示するために，内部の挙動的にどのアサーションがどこからきて，どこに繋がってゆくかを知りたいのに，Presenter からは断片的なリストが渡されたりすることや，Presenter から渡される型が独自のものであり，更にはそれも RuleBaseSystem 内で実装されているため，Presenter から受け取ったデータのフィールドやメソッドを調べるためにわざわざ RuleBaseSystem.java まで調べないといけなかったりすることが挙げられる．

これらは GUI 側としての要求定義が甘かったことも考えられるが，班員の Presenter 担当者が GUI 未経験であることが要因の一つとして挙げられる．今回の課題で，GUI 担当者が本当に欲しいものは GUI を作った経験がある人しか分からないということを痛感したため，次回の課題では GUI の制作を他の班員に任せ，私が Presenter を担当することで，GUI 実装の円滑なサポートと，Presenter 担当側から見た，苦悩や GUI 担当者から欲しい情報等を学びたいと思った．

また，今回 GUI を実装していて RuleBaseSystem.java をしばし調べ，内部の挙動を知る必要があったが，GUI 製作者からすると本当に必要なのはデータの受け渡しだけであるため，次回の課題で実装する Presenter では，実装するメソッドやクラスの説明も充分にできるよう意識したい．

参考文献

TATSUO IKURA：『Swing を使ってみよう - Java GUI プログラミング』 https://www.javadrive.jp/tutorial/ (2019/11/18 アクセス)