知能プログラミング演習 II 課題 4

グループ 07
29114007 池口 弘尚
29114031 大原 拓人
29114048 北原 太一
29114086 飛世 裕貴
29114095 野竹 浩二朗

2019 年 11 月 26 日

提出物　このレポート グループプログラム”group07.zip”

グループ　グループ 07

グループ　グループ 07

メンバー

学生番号	氏名	担当箇所
29114007	池口弘尚	4-3,4-4
29114031	大原拓人	4-3
29114048	北原太一	4-1
29114086	飛世裕貴	4-2
29114095	野竹浩二朗	4-2,4-4

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 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．

2.1 実行例 29114048-北原太一

2.1.1 前向き推論

まず、デフォルトのまま前向き推論の RuleBaseSystem.java を実行した時における推論過程を図 1 に示す。なお、一番上の点線で囲まれたアサーションは元々ワーキングメモリ上にあることを表す。

次に、AnimalWorld.data を利用するために RuleBaseSystem.java の RuleBase クラスのコンストラクタの一部をソースコード 1 のように変えた。

ソースコード 1: RuleBase2.java(一部抜粋)

```
1 fileName = "AnimalWorld.data";
2 wm = new WorkingMemory();
3 wm.addAssertion("my-friend has hair");
4 wm.addAssertion("my-friend eats meat");
5 wm.addAssertion("my-friend has tawny color");
6 wm.addAssertion("my-friend has black");
```

このプログラムを実行した時における推論過程を図 2 に示す。

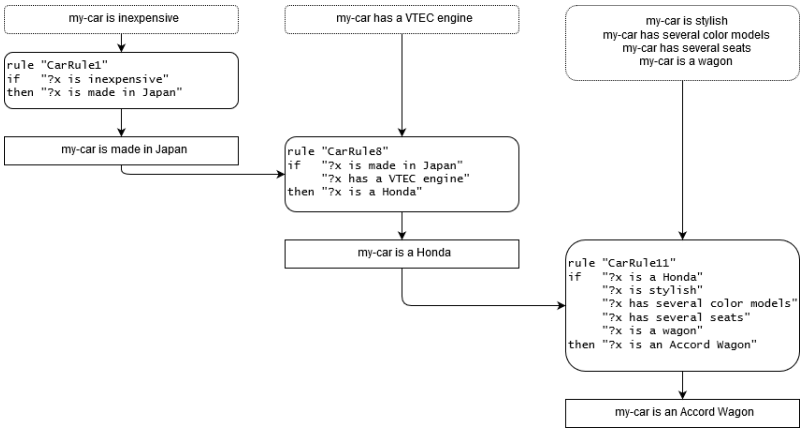

図 1: 前向き推論:CarShop.data

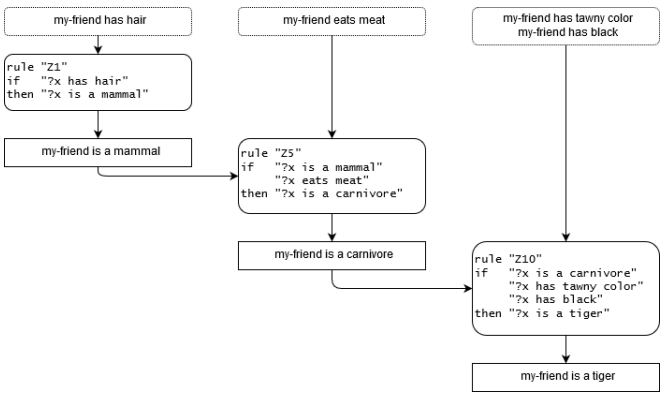

図 2: 前向き推論:AnimalWorld.data

2.1.2 後向き推論

まず、デフォルトのまま後向き推論の RuleBaseSystem.java をソースコード 2 に示すコマンドで実行した。

ソースコード 2: 後ろ向き推論 1

```
1 >java RuleBaseSystem "?x is an Accord Wagon"
```

この実行における推論過程を図 3 に示す。なお、矢印の隣の括弧つき数字はルールの適用順を、(WM) はワーキングメモリ上のアサーションを、矢印上の×印はワーキングメモリ上のアサーションと一致しなかったことを表す。

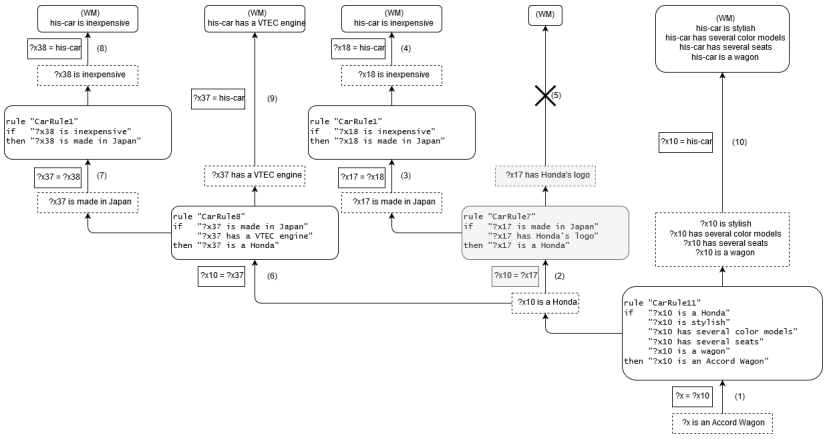

図 3: 後向き推論:CarShop

次に、デフォルトの RuleBaseSystem.java のコメントアウトを付け替え、CarShop.data, CarShopWm.data の代わりに AnimalWorld.data, AnimalWorldWm.data を使用するようにしたうえでソースコード 3 に示すコマンドを実行した。

ソースコード 3: 後向き推論 2

```
1 >java RuleBaseSystem "?x is giraffe"
```

この実行における推論過程を CarShop の場合と同じように図 4 に示す。

2.2 考察 29114048-北原太一

前向き推論では、すでにワーキングメモリ上にあるアサーションからルールを適用して新しいアサーションを導く。すなわち、すでにある知識から新しい知識を導く手法といえる。また、後向き推論では、与えられたパターンにルールを適用し、新しいパターンを作り、最後にワーキングメモリ上のパ

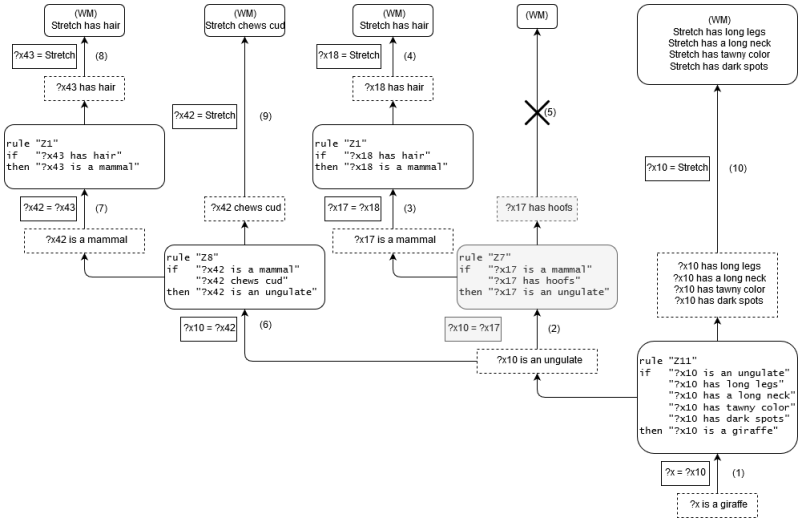

図 4: 後向き推論:AnimalWorld

ターンとマッチングさせる。すなわち、仮定にどの知識が合致する、あるいは当てはまるかを導く手法といえる。

この両者の大きな違いとして、ワーキングメモリの使い方を挙げることができる。すなわち、前向き推論においてはワーキングメモリが推論のスタート地点であり、後向き推論においてはワーキングメモリが推論のゴール地点である。

3 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．

3.1 手法 29114031-大原拓人

私は自然言語（英語）によって質問を解釈する質問応答システムを担当した。時間内に完成できなかったため、考察のみ記す。

3.2　考察 29114031 大原拓人

自然言語で質問を解釈するシステムを作成するにあたって、解決すべき課題は以下のようになった。

1. 疑問詞を検出し変数に置き換える
2. 疑問詞の種類によって変数を配置する場所が異なる
3. 文脈によって変数を配置する場所が異なる
4. クエリと異なり、答えとなる部分は 1 語ではない
5. 3 人称になったときの動詞の活用、名詞の複数形を検出する

以上に挙げたものの解決策を挙げていくと、

1. OpenNLP のライブラリを用いて、文節 chunk に分けることで「whose car」のようなかたまりを検出できるので、それを変数に置き換えればよい
2. 疑問詞の種類自体は少ないので、switch 文で場合分けをすればよい
3. これといった解決策が思いつかなかった。パターン照合のシステムと連携して複数のパターンを探索する形であれば実現の可能性がある。
4. こちらも、パターン照合のシステムに変数が 1 つ、2 つ、3 つ... のそれぞれのパターンを引数として渡すことで実現の可能性がある。
5. OpenNLP のライブラリに、用いられている単語が 3 人称の動詞や複数形であっても、原型が何であるかを返すメソッドがあるのでパターン照合のシステム上に組み込むことで、比較が可能になる。

しかし、以上に挙げた解決策を実装すると、文脈検出の探索と、複数単語の回答のための探索でプログラムの命令数が爆発的に増えると予想される。つまり、ルールや照合すべきパターンが増えると実用的な時間内に結果を返すことができない恐れがある。この爆発的な増加を抑える方法を模索していたが、時間内にその方法を見つけ出すことができなかったので実装には至らなかった。

3.3　手法 29114086-飛世裕貴

本課題ではスポーツに関するデータファイルを作成し、そのデータファイルを用いて前向き推論、後ろ向き推論に基づく質問応答システムを作成する. 以下に今回用いるデータファイル（data.txt、dataWm.txt）の一部を示す.

```
rule  "Rule1"
if  "We play ?x indoors"
then  "?x is an indoor sport"

rule  "Rule2"
if  "We play ?x outdoors"
then  "?x is an outdoor sport"

rule  "Rule3"
if  "We use a ball in ?x"
then  "?x is a ball sport"

rule  "Rule4"
if  "?x is a ball sport"
"We use a racket in ?x"
then  "?x is a racket sport"

rule  "Rule5"
if  "?x is an indoor sport"
"?x is a ball sport"
"We play ?x in 5 people"
then  "?x is basketball"
⋮
（後略）
```

図 1 ：ルールベースに関するデータファイル（data.txt）の一部

```
my-club-activity is an indoor sport
We use a ball in my-club-activity
my-club-activity is a sport over the net
We play my-club-activity in 6 people
His-competition is swimming
We runs 800 meter to 5000 meter on his-competition
```

図 2 ：ワーキングメモリに関するデータファイル（dataWm.txt）

このようなデータファイルを用いて、質問応答システムを作成していく. 尚、質問応答システムの作成に関してはパターンマッチングによる質問応答機能を作成したあと、自然言語に対応させていくという流れで進行していく. 私は後ろ向き推論におけるパターンマッチングによる質問応答機能の作成を担当した.

3.4 実装 29114086-飛世裕貴

パターンマッチングにおいてルールベース、ワーキングメモリの読み込みを loadRules、loadWm メソッドで行い、後ろ向き推論による質問応答は既存の RuleBase クラスの backwardChain メソッドを用いて行った. ここでワーキングメモリを読み込む際、図 2 に示したように数字をトークンとして読み込むと正しく読み込まれず、質問応答が正しく行われない. そのため、loadWm メソッドを以下のように実装した.

ソースコード 4: loadWm メソッド

```
1  public ArrayList<String> loadWm(String theFileName){
2    ArrayList<String> wm = new ArrayList<String>();
3    String line;
4    try{
5      int token;
6      f = new FileReader(theFileName);
7      st = new StreamTokenizer(f);
8      st.eolIsSignificant(true);
9      st.wordChars('\'','\'');
10     while((token = st.nextToken())!= StreamTokenizer.TT_EOF
           ){
11       line = new String();
12       while( token != StreamTokenizer.TT_EOL){
13         //以下､変更点
14         if(st.sval == null)
15           line = line + (int)st.nval + " ";
16         else
17           line = line + st.sval + " ";
18         token = st.nextToken();
19       }
20       wm.add(line.trim());
21     }
22   } catch(Exception e){
23     LogArea.println(e);
24   }
25   return wm;
26 }
```

変更を加えたのは上記プログラム中の 14〜17 行目で、読み込んだトークンが文字列でなければ数値として扱うようにしている. このようにする事でワーキングメモリ中の「We play my-club-activity in 6 people」のようにトークンとして数字を含む場合においても質問応答が可能となった.

3.5 実行例 29114086-飛世裕貴

後ろ向き推論に関するプログラムを実行したとき、その結果を標準出力に出力したものを以下に示す.

```
Enter data-filename:data.txt
Enter workingmemory-filename:dataWm.txt
add rule? delete rule? add...1/delete...2/No thanks...3 :3
Enter Search pattern:?x is volleyball
Hypothesis:[?x is volleyball]
Success RULE
Rule:Rule6 [?x5 is an indoor sport, ?x5 is a ball sport,
?x5 is a sport over the net,We play ?x5 in 6 people]
->?x5 is volleyball <=> ?x is volleyball

⋮
(中略)
⋮

Success WM
We play my-club-activity in 6 people <=> We play ?x5 in 6 people
Yes
{?x=my-club-activity, ?x5=my-club-activity, ?x8=my-club-activity}
binding: {?x=my-club-activity, ?x5=my-club-activity, ?x8=my-club-activity}
tmp: ?x, result:  my-club-activity
Query: ?x is volleyball
Answer:my-club-activity is volleyball
```

図 3 ：後ろ向き推論による質問検索の実行例

このようにトークンとして数字を含んでいたとしても正しく検索ができていることを確認できた.

3.6　考察 29114086-飛世裕貴

今回の課題ではトークンとして数字が含まれている場合においても検索できるように改良を加えたが、「5000 meter」を「5000m」のように略記した際には単一トークン中に数字と文字が含まれることでトークン読み込みが正しく行われず、検索も正常に行われない事がわかった. このような問題は自然言語における形態素解析によって解決できるのではないかと考えられる. トークンという考え方においては「5000m」のような文字列は一つのトークンとして認識してしまうために正常に検索が行えなかったが、この文字列を形態素解析器に読み込ませると「5000」と「m」という二つの形態素として認識する事ができる. そのため、一つのトークンを文字列部と数字部に分離して考える事ができる. 一例として、実際に文字と数字を含む文字列に対して形態素解析器 MeCab を用いて形態素解析を行なった結果を以下に示しておく.

```
5000m
5000 名詞, 数,*,*,*,*,*
m 名詞, 固有名詞, 組織,*,*,*,*
EOS

G7 サミット
G 名詞, 一般,*,*,*,*,*
7 名詞, 数,*,*,*,*,*
サミット 名詞, 一般,*,*,*,*, サミット, サミット, サミット
EOS
```

図４：形態素解析による文字と数字の分割の実行例

このようにトークンとしてではなくさらに細かい分割である形態素を用いることで、より一般的な文字列に対して正常な検索が可能な質問検索システムを実装する事が可能であると考える.

3.7　手法 29114095-野竹浩二朗

この課題ではスポーツに関するデータファイルを作成し, そのデータを用いて前向き推論、後ろ向き推論に基づいて処理をしていく。私は自然言語ではなく、パターンによる処理ができるプログラムを作成した。

3.8　実装 29114095-野竹浩二朗

データファイルの中身は以下のようになっている。

```
rule "Rule1"
if "We play ?x in the water"
then "?x is swimming"

rule "Rule2"
if "?x is swimming"
   "?x can be any way of swimming"
then "?x is freestyle"

rule "Rule3"
if "?x is swimming"
   "?x strokes hands in front of chest"
   "?x move one kick backward"
then "?x is breaststroke"
```

パターン認識には、以前の課題で作成した Unify クラスを用いて、今回の RuleBaseSystem.java のメインメソッドに以下のコードを追加した。

ソースコード 5: 追加したコード

```
1    while(true){
2      System.out.print("Enter Search Pattern:");
3      String query = stdIn.nextLine();
4      if(query.equals("exit")){
5        break;
6      }
7      for(String st:rb.wm.assertions){
8        (new Unifier()).unify(st,query);
9      }
10   }
```

「Enter Search Pattern:」の後に「We play ?x indoors」のように入力することで、入力されたパターンとマッチする文を出力する。

また、データファイル名、ワーキングメモリを書き込んだファイル名を読み込めるよう、RuleBase クラスの RuleBase メソッドに以下を追加した。

ソースコード 6: RuleBase メソッド

```
1    System.out.print("Enter data-filename:"); // ファイル名の
         入力
2    fileName = scan.nextLine();
3    wm = new WorkingMemory();
4    fm = new FileManager();
5
6    System.out.print("Enter workingmemory-filename:");
```

```
7     dataFilename = scan.nextLine();
8     ArrayList<String> wms = fm.loadWm(dataFilename); //ワーキ
          ングメモリの取り込み
9     for(String str : wms){
10      wm.addAssertion(str);
11    }
```

ルールやワーキングメモリに入れる文を変更する場合は、データファイルを変更する。

3.9 実行例 29114095-野竹浩二朗

DataBaseSystem を実行した結果を以下に示す。ルールを追加する際、Rule1 から Rule34 までが羅列されてしまい、実行結果が長くなってしまうため、一部省略している。(「...」の部分)

```
1  Enter data-filename:data.txt
2  Enter workingmemory-filename:dataWm.txt
3  ADD:We play tennis outdoors
4  ADD:We use a ball in tennis
5  ADD:We use racket in tennis
6  Rule1 [We play ?x indoors]->?x is an indoor sport
7  Rule2 [We play ?x outdoors]->?x is an outdoor sport
8  ...(中略)
9  Rule33 [?x is throwing competitions, We throw a hammmer in ?x
       ]->?x is hammer throw
10 Rule34 [?x is throwing competitions, We throw a spear]->?x is
        javelin
11 apply rule:Rule1
12 apply rule:Rule2
13 Success: tennis is an outdoor sport
14 ADD:tennis is an outdoor sport
15 apply rule:Rule3
16 Success: tennis is a ball sport
17 ADD:tennis is a ball sport
18 apply rule:Rule4
19 apply rule:Rule5
20 ...(中略)
21 apply rule:Rule33
22 apply rule:Rule34
23 Working Memory[We play tennis outdoors, We use a ball in
       tennis, We use racket in tennis, tennis is an outdoor
       sport, tennis is a ball sport]
24 No rule produces a new assertion
25
```

```
26 Enter Search Pattern:We play ?x indoors
27 Enter Search Pattern:We play ?x ?y
28 {?x=tennis, ?y=outdoors}
29 Enter Search Pattern:exit
```

ルールとワーキングメモリを読み込んだ後、探索するパターンを入力した結果を見ると、dataWm.txt の中身のデータとマッチする要素を取り出せていることが分かる。

3.10 考察 29114095-野竹浩二朗

私が作成したプログラムでは、入力されたパターンによる検索しかできない。入力された文を単語ごとに分け、キーとなる単語、今回では play や indoor などを識別することができれば自然言語処理に近い形ができると考えられる。このプログラムでは、データファイルに「800m」といった数字と文字が混ざった単語の処理がうまくできず、ユーザーが使える文法が制限されてしまっている。数字が入っていても単に文字列として読み込むことができれば改善すると考えられる。

4 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．

4.1 手法 29114007-池口弘尚

課題 4-2 で作成した探索をカプセル化し、GUI 側からそれを呼び出すことによって実装した。

4.2 実装 29114007-池口弘尚

基本的な構造は今までと同じなので省略する。今回で新たに使用したのは、JTextField でエンターを押したときの判定の取り方である。

ソースコード 7: JTextField の ActionListener

```
1 public ForwardSearchPanel(String name, kadai4.ForwardChain.
      RuleBaseSystem rbs) {
2         （略）
```

```
3         ActionListener searchActionListener = new
              ActionListener() {
4                 @Override
5                 public void actionPerformed(ActionEvent e) {
6                         rbs.patternSearch(text1.getText());
7                         text1.setText("");
8                 }
9         };
10        text1.addActionListener(searchActionListener);
11        button.addActionListener(searchActionListener);
12 }
```

上のコードのように今までボタンに付けていた ActionListener を JTextField にも加えることによって、入力のわずらわしさを減らすことができた。

また、今回はそれぞれのパネルを JTabbedPane を使用してタブで分けた。そのために今まで使ってきた FrameBase を継承して PageFrameBase を以下のように作成した。

ソースコード 8: PageFrameBase

```
1  public class PageFrameBase extends FrameBase {
2         JTabbedPane tabbedpane;
3
4         public PageFrameBase(String title, int x, int y, int
              width, int height, PagePanelBase[] pages) {
5                 super(title, x, y, width, height);
6                 MakePages(pages);
7         }
8
9         public PageFrameBase(String title, int width, int
              height, PagePanelBase[] pages) {
10                super(title, width, height);
11                MakePages(pages);
12        }
13        //ページを作成
14        private void MakePages(PagePanelBase[] pages) {
15                JPanel p = new JPanel();
16
17                tabbedpane = new JTabbedPane();
18                for (PagePanelBase Panel : pages) {
19                        tabbedpane.add(Panel, Panel.getName
                              ());
20                }
21                //tabbedpane.setSize(1600, 100);
22
23                LogArea logArea = new LogArea(30, 140);
24                SpringLayout layout = new SpringLayout();
```

```
25                  p.setLayout(layout);
26
27                  layout.putConstraint(SpringLayout.NORTH,
                        tabbedpane, 5, SpringLayout.NORTH, p);
28                  layout.putConstraint(SpringLayout.SOUTH,
                        tabbedpane, 305, SpringLayout.NORTH, p);
29                  layout.putConstraint(SpringLayout.EAST,
                        tabbedpane, -30, SpringLayout.EAST, p);
30                  layout.putConstraint(SpringLayout.WEST,
                        tabbedpane, 30, SpringLayout.WEST, p);
31
32                  layout.putConstraint(SpringLayout.NORTH,
                        logArea, 10, SpringLayout.SOUTH,
                        tabbedpane);
33                  layout.putConstraint(SpringLayout.
                        HORIZONTAL_CENTER, logArea, 0,
                        SpringLayout.HORIZONTAL_CENTER, p);
34
35                  p.add(tabbedpane);
36                  p.add(logArea);
37                  add(p, BorderLayout.CENTER);
38          }
39
40  }
```

PagePanelBase は JPanel に名前を持たせただけのクラスなので説明は省く。

課題 2 の時にはそれぞれのログは別で表示されていたが、今回は PageFrameBase でログエリアを作成することによって共通で使えるようにしてある。また、今回の課題では過程を表示しなければならないため、どこからでもログを表示できる必要があった。そのため、LogArea は以下のように実装した。

ソースコード 9: Log

```
1  public class LogArea extends JScrollPane {
2          static LogArea instance;
3          JTextArea area;
4          JViewport viewport;
5
6          public LogArea(int rows, int columns) {
7                  super();
8                  area = new JTextArea(rows, columns);
9                  EtchedBorder border = new EtchedBorder(
                       EtchedBorder.RAISED);
10                 area.setEditable(false);
11                 area.setBorder(border);
12                 setViewportView(area);
13                 viewport = getViewport();
```

```
14                  instance = this;
15          }
16          // l 行ログを表示
17          public static void println(String txt) {
18          //インスタンスが無ければSystem.out
19                  if (instance != null)
20                          instance.area.append(txt + "\n");
21                  else
22                          System.out.println(txt);
23          }
24          //続けてログを表示
25          public static void print(String txt) {
26          //インスタンスが無ければSystem.out
27                  if (instance != null)
28                          instance.area.append(txt);
29                  else
30                          System.out.print(txt);
31          }
32  }
```

これは、GUI が作成されていなければ System.out が使われるため、作業段階から利用することができる。

4.3 実行例 29114007-池口弘尚

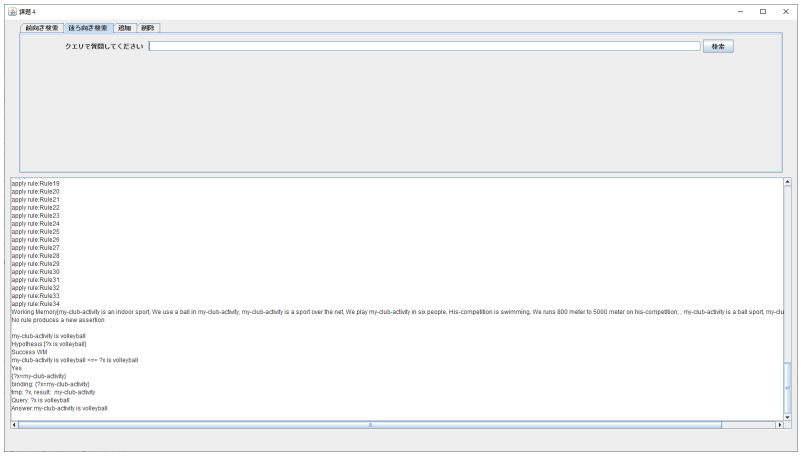

図 5: 検索例

4.4 考察 29114007-池口弘尚

外側のメソッドを呼ぶだけなので問題は生じなかった。

5 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.

5.1 手法 29114007-池口弘尚

今までと同様な手法によって GUI に追加する。変更に関しては、削除してから新たに追加することとした。

5.2 実装 29114007-池口弘尚

GUI 部分に関しては今までと変わらないため説明は省く。RuleBase 部分を編集してルールの追加などを行えるようにした。

ソースコード 10: 編集

```
1  //ルールを 1 つ追加
2  public void addRules(String name, ArrayList<String>
       antecedents, String consequent) {
3          rules.add(new Rule(name, antecedents, consequent));
4  }
5  //ルール名で削除
6  public void deleteRules(String name) {
7          for (int i = 0; i < rules.size(); i++) {
8                  if (rules.get(i).getName().equals(name)) {
9                          LogArea.println("Delete:" + rules.get
                               (i));
10                         rules.remove(i);
11                         i--;
12                 }
13         }
14 }
15 //アサーションを削除
16 public void deleteAssertion(String assertion) {
17         wm.assertions.remove(assertion);
18 }
19 //ルールをまとめて追加
```

```
20 public void addRules(List<Rule> rules) {
21         for (Rule rule : rules) {
22                 if (!this.rules.contains(rule)) {
23                         LogArea.println("Add:" + rule);
24                         this.rules.add(rule);
25                 }
26         }
27 }
28 //アサーションをまとめて追加
29 public void addWorkingMemory(List<String> wmlist) {
30         for (String string : wmlist) {
31                 wm.addAssertion(string);
32         }
33 }
34 //アサーションを 1 つ追加
35 public void addWorkingMemory(String wmline) {
36         wm.addAssertion(wmline);
37 }
```

5.3　実行例 29114007-池口弘尚

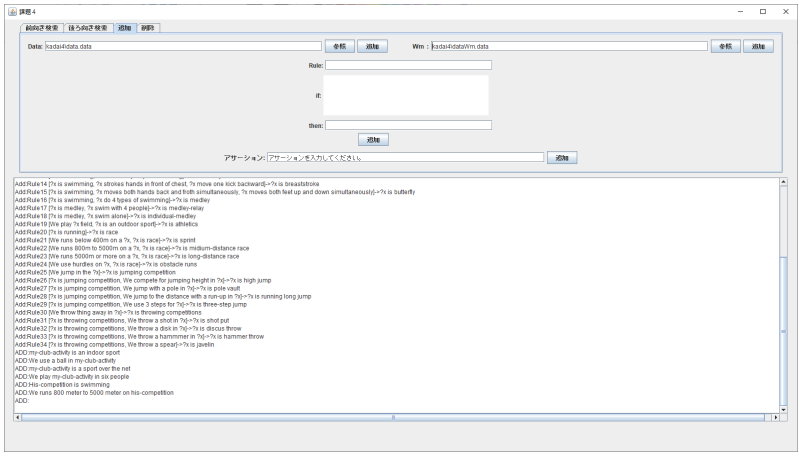

図 6: 追加例

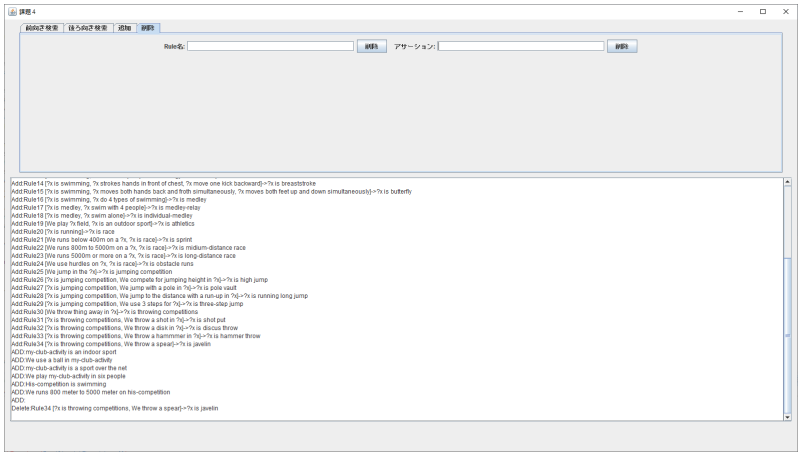

図 7: 削除例

5.4　考察 29114007-池口弘尚

追加や削除を行うときにそれが存在しているかなどの判定が甘いため、想定している通りの操作以外のことをすると上手く動かない可能性がある。

5.5　手法 29114095-野竹浩二朗

ルールを追加・削除するメソッドを作成し、それを用いて GUI からルールの追加、削除を行えるようにする。

5.6 実装 29114095-野竹浩二朗

私は、ルールを追加できるメソッド
addRules と削除できるメソッド deleteRules を作成した。まず、addRules を示す。

ソースコード 11: addRules メソッド

```
1 void addRules(String name,ArrayList<String> antecedents,String
      consequent){
2 rules.add(new Rule(name,antecedents,consequent));
3 }
4 }
```

ルール名、条件、結果を引数とし、その要素をまとめてルールを格納している ArrayList である rules に格納する。次に、ルールを削除する deleteRules を示す。

ソースコード 12: deleteRules メソッド

```
1 void deleteRules(String name){
2   for(int i=0; i<rules.size();i++){
3     if(rules.get(i).getName().equals(name)){
4       rules.remove(i);
5       i--;
6     }
7   }
8 }
9 }
```

ルールを削除する際はルール名のみを引数とする。rules.get(i).getName() により、ルール名を順に呼び出し、入力されたルール名と一致したとき、rules.remove() により、要素を削除する。

ルールを追加する際、既存のルール名と被ってはいけないため、新たに入力されたルール名と既存のルール名を比較する judgeName メソッドも追加した。以下にその中身を示す

ソースコード 13: judgename メソッド

```
1 boolean judgeName(String name){
2   boolean judge = true;
3   while(true){
4     for(int i=0; i<rules.size();i++){
5       if(rules.get(i).getName().equals(name)){
6         judge = true;
7         break;
8       }else{
9         judge = false;
```

```
10          break;
11        }
12      }
13      break;
14    }
15    return judge;
16  }
```

このメソッドでは、入力されたルール名と既存のルール名が被っていた場合に true を返すようになっている。また、メインメソッドには、ルールの追加、削除、何もしないと選べるよう、switch 文を用いて分岐を加えた。

ソースコード 14: メインメソッドに追加した文

```
1     System.out.print("add rule? delete rule? add...1/delete
          ...2/No thanks...3 : ");
2     int j = stdIn.nextInt();
3
4     switch(j){
5       case 1:
6                   ...
7       case 2:
8         ...
9       case 3:
10        ...
11    }
```

switch 文の中身をすべて記述すると無駄に長くなってしまうため省略した。case 1 では入力されたルール名と既存のルール名を比較したのちにルールを追加する。case 2 ではルール名を入力し、ルールを削除するようになっている。case 3 では何もせずに次の操作に移る。

5.7 実行例 29114095-野竹浩二朗

ルールの追加、削除するメソッドを追加した DataBaseSystem を実行した結果を以下に示す。

```
1  Enter data-filename:data.txt
2  Enter workingmemory-filename:dataWm.txt
3  ...(中略)
4  Rule33 [?x is throwing competitions, We throw a hammmer in ?x
       ]->?x is hammer throw
5  Rule34 [?x is throwing competitions, We throw a spear]->?x is
        javelin
6  add rule? delete rule? add...1/delete...2/No thanks...3 :1
7  --- Add Rule !!! ---
```

```
 8 Enter RuleName:Rule35
 9 Enter antecedent:We play ?x space
10 finish
11 Enter consequent:?x is unknown
12 ...(中略)
13 apply rule:Rule34
14 apply rule:Rule35
15 Working Memory[We play tennis outdoors, We use a ball in
       tennis, We use racket in tennis, tennis is an outdoor
       sport, tennis is a ball sport]
16 No rule produces a new assertion
17 ...(以下略)
```

ここでも、実行結果を省略して載せている。ルールの追加を選択後、3 つの要素を入力すると aplly rule にルールが反映されていることが確認できる。

5.8 考察 29114095-野竹浩二朗

ルールを削除する際、ルール名を把握しなければならない。今回では RuleXX というルールの番号が分かっていないと削除できないため、ルールの内容から検索、削除ができるようになればさらに汎用性が高いものができると考えたが、効率的にルールを検索する方法が思いつかず、今回はルール名を指定して削除する方法となってしまった。

6 感想 29114007-池口弘尚

コードの設計をどのようにするのかなどを取り決めていなかったため、中途半端な実装になってしまった。例えば、RuleBase が Forward と Backward で分かれていたために同じような機能が別のもので実装されていたり、Matcher と Unifier が存在したりするといったことである。そのため、次回からは作ったクラスやメソッドをリストアップしていく必要があると思った。

7 感想 29114031-大原拓人

今回実装が間に合わなかった要因として、自然言語での質疑応答システムが予想以上に複雑であったことと、完全な自然言語への対応を妥協し、もう少し単純なシステムで解決しようとしなかったことが挙げられる。グループのメンバーと相談し、ある程度の妥協案を出しておくべきであった。

8　感想 29114095-野竹浩二朗

今回の課題では、BackwordChain、ForwordChain の 2 つがあり、片方でできてももう片方ではできなかったりと苦労した。また、GUI を作ったわけではないが、GUI を作るうえで使うメソッドを作成した。これに関して GUI の作成を担当してくれている人からどんなメソッドの方がいいかを聞きながらの作成となったので大変だった。

参考文献

[1] ウェブインテリジェンスの演習で用いられたコードの例を参考にした

[2] 新谷虎松，講義「知識システム」スライド