知能プログラミング演習II 課題4

グループ8
29114142 湯浅範子

2019年11月26日

提出物　rep4(2914142.pdf),group08.zip

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

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．

私は必須課題 4-3 に関わるプログラムを作成したため，それについて記述する．

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，与えられたプログラムを GUI から利用できるようにするためのメソッドやファイルの追加である．

2.1 手法

GUI の動作のために MVP モデルを利用し，GUI から命令を受け動作プログラムに実行を依頼する Presenter を作成し，以下の機能を実装した．

1. 前向き推論
    - 初期状態のルールとアサーションを用いた推論を行う
    - ルールの編集を行った後もアサーションを用いた推論を行う
    - 推論が行われた手順を保持し，GUI 側に受け渡す
    - アサーションによる検索を行う (検索時の推論手順も返す)
    - ルールの一覧表示を行う
    - ルールに対する編集を行う

2. 後ろ向き推論

- アサーションによる検索を行う
- 推論探索手順を保持し，GUI に受け渡す
- ルールに対する編集を行う

1. については，前向き推論のプログラムに対して作成した Presenter に実装した．

2. については，後ろ向き推論のプログラムに対して作成した Presenter に実装した．

ルールの編集に関するメソッドの枠組みも作成したが，この部分の実装は増田君が担当したため詳細はそちらのレポートを参考にされたい．

2.2 実装

まず，前向き推論時に必要になるメソッドを順に示す．ここでは私が作成した以下の部分に関するメソッドを中心に記述する．

- 初期状態のルールとアサーションを用いた推論を行う
- ルールの編集を行った後にもう一度推論を行う
- 推論が行われた手順を保持し，GUI 側に受け渡す
- アサーションによる検索を行う (検索時の推論手順も返す)
- ルールの一覧表示を行う

始めに，前向き推論の Presenter.java で作成したメソッドをソースコード 1 に示す．

ソースコード 1: Presenter.java(略)

```
1 class Presenter {
2         private RuleBaseSystem rulebasesystem = new
              RuleBaseSystem();
3         …
4         // GUI 起動時に初期データの読み込みと出力
```

```
5         public void start(ArrayList<String>
              firstAssertions, String fileName) {
6             rulebasesystem.start(firstAssertions,
                  fileName);
7         }
8         // ルール更新後のデータから推論の再試行を行う
9         public void restart(ArrayList<String> assertions)
              {
10            ArrayList<Rule> rules = rulebasesystem.
                  getRules();
11            rulebasesystem.restart(assertions, rules);
12        }
13        // 探索を順に追った結果を返すよう指示する
14        public ArrayList<StepResult> stepResult() {
15            ArrayList<StepResult> stepresults =
                  rulebasesystem.getStepResults();
16            return stepresults;
17        }
18        …
19        // ルールの一覧表示を行うよう指示する
20        public ArrayList<Rule> fetchRules() {
21            ArrayList<Rule> ruleList = new ArrayList
                  <>();
22            ruleList = rulebasesystem.getRules();
23            return ruleList;
24        }
25        // アサーションによる検索をするよう指示する
26        public ArrayList<ArrayList<SearchStep>>
              searchAssertion(ArrayList<String> targetData)
              {
27        ArrayList<ArrayList<SearchStep>> resultList = new
              ArrayList<>();
28            for (String target : targetData) {
29                resultList.add(rulebasesystem.
                      searchAssertion(target));
30            }
31            return resultList;
32        }
33    }
```

start メソッドと restart メソッドでは，アサーションとルールを取得しそれを基に前向き推論を行うように RuleBaseSystem.java に実装を行っ

た．fetchRules メソッドは現在取得しているルールを getRule メソッドで RuleBase クラスから取得し，それを返却することで実現させた．

また stepResult メソッドでは，前向き推論を行う途中でその経路を保持し返却を行うことで実現させた．また返却のために新たに StepResult クラスを作成した．これについては後に詳しく記述する．

次にアサーションによる検索をするよう指示するメソッド searchAssertion とその関連メソッドをソースコード 2 に示す．

ソースコード 2: searchAssertion メソッド

```
1  // 検索を行い結果を返す (複数検索の場合は改良)
2  public ArrayList<SearchStep> searchAssertion(String target
       ) {
3          RuleBaseSystem.question = true;
4          answer = new ArrayList<>();
5          answerString = new ArrayList<>();
6          sss = new ArrayList<>();
7          List<StepResult> subSRs = new ArrayList<>();
8          NaturalLanguage(target);
9          for (int i = 0; i < answer.size(); i++) {
10                 if (answerString.size() > 0) {
11                         subSRs = searchSRs(answerString.get
                               (i));
12                         ss = new SearchStep(answer.get(i),
                                subSRs);
13                 } else {
14                         ss = new SearchStep(answer.get(i),
                                null);
15                 }
16                 sss.add(ss);
17                 subSRs = new ArrayList<>();
18         }
19         Matcher.answer = new ArrayList<>();
20         Matcher.answerString = new ArrayList<>();
21         return sss;
22 }
23 // 探索経路のみの抽出
24 public List<StepResult> searchSRs(String answerS) {
25         ArrayList<StepResult> SRs = getStepResults();
26         List<StepResult> subSRs = new ArrayList<>();
27         int target = -1;
```

```
28          for (int i = 0; i < SRs.size(); i++) {
29                  String success = SRs.get(i).getSuccess().
                        getName();
30                  if (success.equals(answerS)) {
31                          target = i;
32                  }
33          }
34          subSRs = SRs.subList(0, target+1);
35          return subSRs;
36  }
```

ここでは，後藤君が作成してくれた英文の然言語での検索を行う NaturalLanguage メソッドを活用して，探索結果とその時一致したアサーションを取得する．この時取得したアサーションと前向き推論で得られた探索経路を比較する．それが探索によって得られたアサーションだと明らかになった場合は，ここまでに得られる経路を subList メソッドによってコピーし返却することで，探索結果に対する経路も得られるようにした．またこれも StepResult 同様に SearchStep クラスを新たに作成して返却を行った．これについても後に詳しく記述する．

次に，後ろ向き推論時に必要になるメソッドを順に示す．

- アサーションによる検索を行う
- 推論探索手順を保持し，GUI に受け渡す

これらの機能を実装するため，後ろ向き推論の Presenter.java で作成したメソッドについてソースコード 3 に示す．

ソースコード 3: Presenter.java(略)

```
1  class Presenter {
2          private RuleBaseSystem rulebasesystem = new
               RuleBaseSystem();
3          …
4          // 初期ルールデータの読み込み
5          public void start(String filename) {
6                  rulebasesystem.start(filename);
7          }
8          // 検索を順に追った結果を返すよう指示する
9          public ArrayList<StepResult> stepResults(String
               wmname, String target) {
```

```
10                  ArrayList<StepResult> stepresults =
                        rulebasesystem.stepResult(wmname,
                        target);
11                  return stepresults;
12          }
13          …
14          // ルールの一覧表示を行うよう指示する
15          public ArrayList<Rule> fetchRules() {
16                  ArrayList<Rule> ruleList = new ArrayList
                        <>();
17                  …
18                  ruleList = rulebasesystem.fetchRules();
19                  return ruleList;
20          }
21  }
```

start メソッドでは，ファイルの読み込みによるルールの構築のみを行う．また fetchRules では現在所持しているルールを RuleBase クラスから getRules メソッドで取得し返却を行うことで一覧表示が行えるようにした．

探索を順に追った結果を返すためのメソッド stepResults は，RuleBaseSystem クラスで次のソースコード 4 のように実装した．

ソースコード 4: stepResults メソッド

```
1  // 後ろ向き推論
2  public ArrayList<StepResult> stepResult(String wmname,
       String target) {
3           ArrayList<Rule> rules = new ArrayList<>();
4           if (fileFlag == true) {
5                   rules = getFirstRules();
6                   fileFlag = false;
7           } else {
8                   rules = getRules();
9           }
10          ArrayList<String> wm = fm.loadWm(wmname);
11          rb = new RuleBase(rules,wm);
12          ArrayList<String> queries = NaturalLanguage(target
                );
13          rb.backwardChain(queries);
14          ArrayList<StepResult> answer = rb.getStepResults
```

```
            ();
15          return answer;
16  }
```

ここでも，後藤君が作成した英文の然言語での検索を行う NaturalLanguage メソッドを活用して探索を行い，その結果の経路を返却する．

2.3 実行例

GUI は青山君が作成するため，ここでは私が実装した Presenter の動作が正しく動作が行われているかどうかを Test.java を作成して確認した．このとき動作確認の簡単のため，扱うデータは始めに与えられた CarData.data と CarDataWM.data とした．

まず start メソッドと restart メソッドの動作確認を行った．Test.java でプログラムを実行すると，教科書で示されている結果と同様の出力が得られた．ここから start メソッドが正しく機能していることが確認できた．また restart メソッドについても，増田君が作成してくれたルールを編集を行った後の推論の再試行では，存在しなくなったルールについての推論は行っていないことも確認できた．このことから restart メソッドも正しく動作していることが確認できた．

次にルールの一覧を表示する fetchRules メソッドの実行結果を以下に示す．

ソースコード 5: fetchRules メソッドの実行結果

```
1  ruleName:CarRule1
2  antecedentName:?x is inexpensive
3  ruleConsequent:?x is made in Japan
4  ruleid:0
5
6  ruleName:CarRule2
7  antecedentName:?x is small
8  ruleConsequent:?x is made in Japan
9  ruleid:1
10
11 ...
12
```

```
ruleName:CarRule14
antecedentName:?x is a foreign car
antecedentName:?x is a sports car
antecedentName:?x is red
antecedentName:?x has a big engine
ruleConsequent:?x is a Ferrari F50
ruleid:13

ruleName:CarRule15
antecedentName:?x is a foreign car
antecedentName:?x is a good face
ruleConsequent:?x is a Jaguar XJ8
ruleid:14
```

確認すると，CarData.data 内にあるデータを全て正しく表示できていることが確認できた．また今回は Rule クラスのインスタンスをより簡単に操作できるよう id を加え，それも表示させた．これは 0 から始まっているため，ルールの名称の番号より全て 1 小さくなっているものの，id の割り当ても正しく行えていると確認された．

次に，推論経路を表示する stepResult メソッドを実行したときの結果を示す．

ソースコード 6: stepResult メソッドの実行結果

```
◆ADD: my-car is inexpensive
◇apply rule: CarRule1
◆success: my-car is made in Japan

◆ADD: my-car is made in Japan
◆ADD: my-car has a VTEC engine
◇apply rule: CarRule8
◆success: my-car is a Honda

◆ADD: my-car is a Honda
◆ADD: my-car is stylish
◆ADD: my-car has several color models
◆ADD: my-car has several seats
◆ADD: my-car is a wagon
◇apply rule: CarRule11
◆success: my-car is an Accord Wagon
```

この実行結果は，どのアサーションを用い (ADD)，どのルールが適用されたことで (apply rule) どのようなアサーションが新しく生成されたか (success) を示している．これは教科書や WM・ルールデータの参照により正しい推論手順が返されていると確認できた．またデータを変えても出力は正しく行われた．

更に探索結果とその経路を出力する searchAssertion メソッドの実行結果を示す．

ソースコード 7: searchAssertion メソッドの実行結果

```
1  検索質問:Is my car a Honda ?
2  答え = Yes
3  Data is new WM
4  ■ADD: my-car is inexpensive
5  ◇apply rule: CarRule1
6  ◆success: my-car is made in Japan
7  ■ADD: my-car is made in Japan
8  ■ADD: my-car has a VTEC engine
9  ◇apply rule: CarRule8
10 ◆success: my-car is a Honda
11
12 検索質問:What does she car have ?
13 答え =
14 No Data
15
16 検索質問:What does my car have ?
17 答え = color-models
18 Data is initial WM
19 答え = seats
20 Data is initial WM
21
22 検索質問:Is my car a Toyota ?
23 答え = No
24 No Data
```

この実行結果では，検索による結果と，その結果に対する導出経路を出力するようにした．ここでは，質問結果として考えられる 4 つの例について示す．

1 つ目は，答えが Yes/No で示され，出力結果が推論により得られたア

サーションだった場合である．この時,「答え」にはYesと示され，アサーションが新しく導かれたものであると示す「Data is new WM」の出力の後，そのアサーションが導かれるまでの経路が算出される．

2つ目は，質問が具体値を求めたがそれに値する具体値が発見できなかった場合である．この時は「答え」には何も表示されず，経路には「No Data」と表示される．

3つ目は，答えが具体値で与えられ，得られたアサーションは初めからWMに入っていた場合である．この時「答え」には具体値が出力され，経路は「Data is initial WM」と表示され推論によって得られたデータではないことが示されている．

4つ目は，答えがYes/Noで示され，質問に対応するアサーションが発見できなかった場合である．この時「答え」はNoと出力され，経路も「No Data」と表示される．

出力方法の全てのパターンは以上の例の組み合わせによって得られるため，探索結果の出力も正しく行えていると確認できた．

次に，後ろ向き推論についての実行結果を示す．

ルールデータの一覧表示のメソッドについては，実行したところ前向き推論と同じ結果が得られたので，ルール編集を含め正しく一覧表示が行えていることが確認された．

後ろ向き推論によって得られた探索経路の返却を行うstepResultsメソッドについては，以下に実行結果を示す．

ソースコード 8: stepResultsメソッドの実行結果

```
1  patarn = his-car is inexpensive
2  queries = [his-car is inexpensive]
3  Hypothesis:[his-car is inexpensive]
4  Success WM
5  his-car is inexpensive <=> his-car is inexpensive
6  Yes
7  {}
8  binding: {}
9  Query: his-car is inexpensive
10 Answer:his-car is inexpensive
11 【出力結果】
```

```
◇QuestionField: Target Question(No Field) ◆Question: his
    -car is inexpensive
◇AnswerField: WorkingSpace ◆Answer: his-car is
    inexpensive

patarn = ?x is an Accord Wagon
queries = [?x is an Accord Wagon]
Hypothesis:[?x is an Accord Wagon]
…
Yes
{?x38=his-car, ?x37=his-car, ?x10=his-car, ?x=his-car}
binding: {?x38=his-car, ?x37=his-car, ?x10=his-car, ?x=
    his-car}
tmp: ?x, result: his-car
Query: ?x is an Accord Wagon
Answer:his-car is an Accord Wagon
【出力結果】
◇QuestionField: Target Question(No Field) ◆Question: ?x
     is an Accord Wagon
◇AnswerField: CarRule11 ◆Answer: ?x10 is an Accord
    Wagon

◇QuestionField: CarRule11 ◆Question: ?x10 is a Honda
◇AnswerField: CarRule8 ◆Answer: ?x37 is a Honda

◇QuestionField: CarRule8 ◆Question: ?x37 is made in
    Japan
◇AnswerField: CarRule1 ◆Answer: ?x38 is made in Japan

◇QuestionField: CarRule1 ◆Question: ?x38 is inexpensive
◇AnswerField: WorkingSpace ◆Answer: his-car is
    inexpensive

◇QuestionField: CarRule8 ◆Question: ?x37 has a VTEC
    engine
◇AnswerField: WorkingSpace ◆Answer: his-car has a VTEC
    engine

◇QuestionField: CarRule11 ◆Question: ?x10 is stylish
◇AnswerField: WorkingSpace ◆Answer: his-car is stylish

◇QuestionField: CarRule11 ◆Question: ?x10 has several
```

```
    color models
45 ◇AnswerField: WorkingSpace ◆Answer: his-car has several
    color models
46
47 ◇QuestionField: CarRule11 ◆Question: ?x10 has several
    seats
48 ◇AnswerField: WorkingSpace ◆Answer: his-car has several
    seats
49
50 ◇QuestionField: CarRule11 ◆Question: ?x10 is a wagon
51 ◇AnswerField: WorkingSpace ◆Answer: his-car is a wagon
```

今回の探索経路では，検索文とそれに一致する答えの文章を探すため，出力は

- 検索文がどこから (どのルールから) 導かれたものかを表す QuestionField
- 検索文本体 Question
- 答えの文章がどこから (どのルールから) 導かれたものかを表す AnswerField
- 答えの文章本体 Answer

の 4 つに分類して出力を行った．

今回は 2 つの例を実行結果とした．1 つ目は探索結果が WM に既にあった場合である．このとき探索経路は検索文と答えの文章，それぞれの Field は無し (No Field) と WM である．2 つ目は探索結果が WM 似なかった場合である．この時，正しく結果を得られた経路のみを保持し出力できるようにした．さらに，それぞれがどのルールから得られた文章なのかが Field によって簡単に分かるようになっている．これらの結果も，教科書やルールから正しいことが確認できた．

2.4 考察

プログラムの作成にあたり，今回の課題は以前使用した MVP モデルでの実装を考えた．以前増田君が作成した Presenter を参考にしながら，RuleBaseSystem プログラムを操作し，求める動作を行うように Presenter

からの指示で操作が行えるようにプログラムを行うことを考えた．

まず始めに前向き推論に対する Presenter とその関連メソッドについて記述する．

前向き推論では探索前に推論を行い，推論可能な全アサーションを導いている．そのため推論のみを行うメソッドと，ルールの変更があった際にもう一度推論をやり直すメソッドが必要になると考えられた．さらに課題にもあるように推論過程を可視化するため，推論手順を保持するメソッドを作成することも考えた．

プログラム動作時はどのような data ファイルや初期アサーションであっても，ユーザーが簡単に変更可能にする必要があると考えた．プログラムを改良することでファイルの名前と初期アサーションのリストを引数にし，その引数を基にして推論を行えるようにした．

また，既存の Rule クラスと Assertion クラスでは推論過程を導く際にどのルールやアサーションを活用しているかを取得するのに手間がかかる．推論過程を格納する際にはどのアサーションを活用しているのかを分かりやすくする必要があると考えたため，ルールとアサーションの通し番号 id を作成することで，それぞれを番号で管理出来るようにした．

発展課題でルールの編集を行うが，ルールの編集を行ってしまうと今までに作成したアサーションが成立しなくなる場合があるため，ルールの編集後にはもう一度推論を行いアサーションを再構築する必要があると考えられた．話し合った結果，再構築の際ルールは編集後のものを用いるが，アサーションは新しく入力し直すと決めたため，アサーションリストのみ引数に持つ restart メソッドの作成を行った．始めの入力ではルールとアサーション共に String 型のリストであるが，restart ではルールは既に Rule 型のリストになっているため，始めの推論とは異なる動作をする．このことを考慮し，今回はコンストラクタを多重定義することで動作の違いに対応した．

推論経路の導出については，与えられた実行結果から容易に導けるものではなかったため，プログラムを読み解きながら修正を加えた．このとき「どのアサーションを用い・どのルールを適用させることで・どんなアサーションが作成されたか」の情報によって推論手順が導けると考

えた．そのため，これらの情報を推論順に格納できるクラス StepResult を新しく作成した．また推論方法から適用ルールと生成アサーションは 1 つであり，始めに加えられるアサーションは複数であるため，どのアサーションを用いるかのみリストを用い実装した．さらに，探索時には得られる結果 (String) とそれを導いた導出経路 (ArrayList<StepResult>) を出力するため，これらをまとめた SearchStep クラスを新たに作成し保持した．

導出経路については，推論により得られた WM と一致する答えがどこから導かれたものなのかを検索し，答えが StepResult リストの新しく得られたアサーションであった場合は，そのアサーションを導くまでの部分の経路の抜き出しを行った．抜き出しについてはシャローコピーでも良いと考え，subList メソッドを用いて実装することを考えた．

次に後ろ向き推論についての Presenter とその関連メソッドについて記述する．

後ろ向き推論では探索を行う際に推論を行うため，前向き推論と異なり探索時に導出経路の抜き出しを行う必要はない．WM が初めから与えられていたため，初期ルールデータのみの読み込みを start メソッドで行った．しかし今回の課題ではルールデータと WM のデータを得た後に実際の操作を行う RuleBase クラスのインスタンスを作成する形であった．この形では Rule の情報が start 実行直後では得られない状態になり，解決のため別の変数を用意するなど工夫が必要であった．そのため，start ではルールデータと WM データのどちらもを同時に読み込み RuleBase インスタンスを作成するか，start メソッド自体を作成しない方がプログラムをより効率よく利用できたのではないかと考えられた．

また，前向き推論ではルールやアサーションに id 番号を付与したが，プログラムを作成し改良を進める中で必要性があまり感じられなくなったため，後ろ向き推論では作成しなかった．ルールの一覧表示については，前向き推論と同じ考え方で実装した．

探索経路については,「検索しようとしている質問文・その質問文がどこに (どのルールに) 属しているか・得られた結果・結果がどこに (どのルールに) 属しているか」の 4 つの情報により構成されると考えた．そのため推論中で常にこれらの情報を保持し，4 つが正しく格納された段階で

データをまとめたStepResultクラスのインスタンスとして作成した．しかしプログラムをそのまま実行すると間違えた探索経路も格納されてしまう．そこで探索に失敗した場合はその情報を保持し，データを格納する際に失敗して得られた結果の上に上書きする形で書き込むことで，正しい推論手順のみを保持した経路を導けるようにした．

さらに変数に通し番号がつけられていることを利用して，リストに通し番号に対応するルールを格納することを考えた．これにより探索失敗時や探索データをWMに発見したことによる変更時など，探索が戻った際の質問文がどのルールから得られているかを簡単に導けるようにした．

また班員と相談した結果，GUIへの表示を行う際には結果の経路保持の際にリンクを用い，直前や直後のStepResultの情報が得られるようにしたほうが良いとなり，実装方法を変更した．このとき変更した定義についてのソースコードなどを以下に示す．後ろ向き推論については，結果の出力を行えるようにしていなかったため，結果と経路を格納するSearchStepクラスも作成した．

ソースコード 9: 実装方法の変更部

```
1  // 前向き推論
2  class StepResult {
3          private ArrayList<StepResult> addSR; // リンク
4          private Rule apply;
5          private Assertion success;
6  }
7  // リンク作成メソッド
8  private ArrayList<StepResult> findLink(ArrayList<
       StepResult> saveALL, ArrayList<Assertion> add) {
9    ArrayList<StepResult> addLink = new ArrayList<>();
10   for(int i = 0; i < add.size(); i++) {
11     for (int j = 0; j < saveALL.size(); j++) {
12       if (add.get(i).getName().equals(saveALL.get(j).
             getSuccess().getName())) {
13         addLink.add(saveALL.get(j));
14       }
15     }
16   }
17   return addLink;
18 }
19 // 後ろ向き推論
```

```
20 class StepResult {
21         private ArrayList<StepResult> addSR; // リンク
22         private Rule AnswerField;
23         private String Answer;
24 }
25 // リンクを所持したStepResult インスタンスの作成
26 private ArrayList<StepResult> changeData() {
27   int remember = -1;
28   ArrayList<StepResult> Link = new ArrayList<>(); // 返却
       するものの逆順 (Link 格納のため)
29   ArrayList<StepResult> reLink = new ArrayList<>(); // 返
       却するもの
30   for (int i = srs.size()-1; i >= 0; i--) {
31     ArrayList<StepResult> addLink = new ArrayList<>(); //
         子リンクを表すリンクリスト
32     if (srs.get(i).getAF() == null) {
33       StepResult change = new StepResult(null, null, srs.
           get(i).getA());
34       Link.add(change);
35       reLink.add(0, change);
36     } else {
37       remember = i;
38       int count = 0;
39       for (int j = srs.size()-1; j > remember; j--) {
40         if (srs.get(j).getQF().getName().equals(srs.get(i
             ).getAF().getName())) {
41           addLink.add(Link.get(count));
42         }
43         count++;
44       }
45       StepResult change = new StepResult(addLink, srs.get(
           i).getAF(), srs.get(i).getA());
46       Link.add(change);
47       reLink.add(0, change);
48     }
49   }
50   ArrayList<StepResult> addLink = new ArrayList<>();
51   addLink.add(Link.get(Link.size()-1));
52   StepResult change = new StepResult(addLink, null, srs.
       get(0).getQ());
53   Link.add(change);
54   reLink.add(0, change);
```

```
55   return reLink;
56 }
```

これらを実行した実行結果を示す (変更部は表示方法のみ).

ソースコード 10: 実装方法変更後の実行結果

```
1  // 前向き推論
2  ◇ADD Link rule: No Link(Data is WM)
3  ◆ADD: my-car is inexpensive
4  □apply rule: CarRule1
5  ■success: my-car is made in Japan
6
7  ◇ADD Link rule: CarRule1 [?x is inexpensive]->?x is made
       in Japan
8  ◆ADD: my-car is made in Japan
9  ◇ADD Link rule: No Link(Data is WM)
10 ◆ADD: my-car has a VTEC engine
11 □apply rule: CarRule8
12 ■success: my-car is a Honda
13
14 ◇ADD Link rule: CarRule8 [?x is made in Japan, ?x has a
     VTEC engine]->?x is a Honda
15 ◆ADD: my-car is a Honda
16 ◇ADD Link rule: No Link(Data is WM)
17 ◆ADD: my-car is stylish
18 ◇ADD Link rule: No Link(Data is WM)
19 ◆ADD: my-car has several color models
20 ◇ADD Link rule: No Link(Data is WM)
21 ◆ADD: my-car has several seats
22 ◇ADD Link rule: No Link(Data is WM)
23 ◆ADD: my-car is a wagon
24 □apply rule: CarRule11
25 ■success: my-car is an Accord Wagon
26
27 // 後ろ向き推論
28 【出力結果】
29 Answer: his-car is an Accord Wagon
30 ◇Link rule: CarRule11 [?x10 is a wagon]->?x10 is an
     Accord Wagon
31 ◆Link: ?x10 is an Accord Wagon
32 □Answer Field: No rule
33 ■Answer: ?x is an Accord Wagon
34
```

```
35 ◇Link rule: CarRule8 [?x35 has a VTEC engine]->?x35 is a
       Honda
36 ◆Link: ?x35 is a Honda
37 □Answer Field: CarRule11
38 ■Answer: ?x10 is an Accord Wagon
39
40 ◇Link rule: CarRule1 [?x36 is inexpensive]->?x36 is made
       in Japan
41 ◆Link: ?x36 is made in Japan
42 □Answer Field: CarRule8
43 ■Answer: ?x35 is a Honda
44
45 ◇Link rule: No Link(Data is WM)
46 ◆Link: his-car is a wagon
47 ···
48 ◇Link rule: No Link(Data is WM)
49 ◆Link: his-car has a VTEC engine
50 ◇Link rule: No Link(Data is WM)
51 ◆Link: his-car is inexpensive
52 □Answer Field: CarRule1
53 ■Answer: ?x36 is made in Japan
54
55 ◆Link: No Link(Data is WM)
56 □Answer Field: No rule
57 ■Answer: his-car is inexpensive
58
59 ···
60
61 ◆Link: No Link(Data is WM)
62 □Answer Field: No rule
63 ■Answer: his-car is a wagon
```

得られた実行結果では，自分の 1 つ前 (1 つ後) のリンク情報のルールとその時のデータについて表示を行った．ここからリンク情報が正しく格納され，経路の関係性が明確化したことが分かった．

最後にプログラム全体に対する考察を行う．

今回は初めての 2 週間課題であったが，出来るだけ早くプログラムを作成させるため，プログラムの全体像を意識した．具体的にはプログラムの大枠を作成した段階でそれを班員と共有し，出来るだけ全員が共通

の認識を持ってプログラムが作成できるように努めた．しかし今回この方法はあまり上手くいかなかった．理由としては，大枠を作成したメソッドの引数や目的まで共有せずプログラムを作成し始めてしまったことにある．大まかなメソッドの内容を見て，各々が自分のプログラムに使いやすいよう使おうとプログラムを作成したため，メソッドの使用方法の認識が少しずつ異なってしまった．また大枠の変更が正しく伝わっておらず，実装方式が異なってしまったということもあった．

情報を早くから共有すること自体は悪くない手段であったと考えるが，早めに詳細についても相談をしたり，変更があった場合は報告する必要性を強く感じる課題となった．

また今回の課題では，一度作ったプログラムに修正を何度も重ねて求める結果を得た．そのため機能が修正前のものの拡張になってしまい，自由度が下がり冗長になったと感じられた．期間が短いため新しくプログラムを構築し直すことは難しいと考えられるが，一度本当にこの実装方法で効率よく実現できるのかを考えてプログラムする必要があると考えられた．

今回は実装期間もあり「このような入力はない」や「このような状態にはならない」といった仮定のもと実装が行われた部分があったため，実装期間が長ければ，あらゆる可能性に耐えられるプログラムを作成する必要があると考えた．

私が担当したプログラムは，他の班員に対してデータを渡したり受け取ったりする指示役の部分であったため，他の人が作成したプログラムと自身のプログラムの引数やデータ形式を合わせる必要があった．これらの調節をする中で，相手にとって使いやすい形式と自分が考えていた形式が異なることが多々あった．複数人でプログラムを行う際には，相手にとって使いやすく見やすいプログラムを作成する必要性を今まで以上に感じたので，コメントアウトや分かりやすい変数名・メソッド名なども今後気に掛けられるようにしたい．

3　感想

今回の課題では，与えられたプログラムで出力される内容とGUIで出力したい内容が異なっていたため，新しくメソッドや変数を用意しプログラム本体の改良を行わなければならなかったのが大変だった．また，受け渡しのプログラムが完成しなければ最終的なGUIの調整をすることが出来ないため，早く私のプログラムを完成させ，GUI側にプログラムを渡したいと考えていたが，デバッグ作業に時間がかかったため早く渡すことが出来なかった．今後は出来るだけ計画的にプログラムの作成を進めていけるようにしたい．

また，今回のプログラムは他にもより効率の良い実装方法があると感じたので，他の人がどのような実装を行ったのか知りたいと感じた．

MVPモデルについて今までより深く知ることが出来たが，Viewをあまり有効活用できなかった．今後の課題の中で機会があれば，さらに学んでいきたいと感じた．

参考文献

[1] 新谷虎松『Javaによる知能プログラミング入門』コロナ社，2002年.

[2] java 美しい MVP の具体的なコード例, https://code-examples.net/ja/q/ad7352 （2019年11月18日アクセス）.

[3] Hatena Blog Konifar's WIP, http://konifar.hatenablog.com/entry/2015/04/17/010606 （2019年11月18日アクセス）.

[4] Java Master Java クラスのインスタンス化・初期化とは？, http://carey.link/java/java-instance（2019年11月18日アクセス）.

[5] Eclipseではじめるプログラミング（7）: Javaの「クラス」と「インスタンス」を理解する (1/2), https://www.atmarkit.co.jp/ait/articles/0503/19/news020.html （2019年11月18日アクセス）.

[6] 【Java入門】Listのコピーとcloneメソッドの使い方, https://www.sejuku.net/blog/19769 （2019年11月25日アクセス）.

[7] [Java共通部品] 数値チェック, https://blog.java-reference.com/common-check-number/ （2019年11月25日アクセス）.