

知能プログラミング演習 II 課題 2

グループ 8

29114060 後藤 拓也

2019 年 11 月 11 日

■提出物 rep4

■グループ グループ 8

■グループメンバー

学生番号	氏名	貢献度比率
29114003	青山周平	no
29114060	後藤拓也	no
29114116	増田大輝	no
29114142	湯浅範子	no
29119016	小中祐希	no

■自分の役割 必須課題 4.2

「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」

1 課題の説明

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．

2 手法

1. 後ろ向き推論における英文を用いた質問
2. 前向き推論における英文を用いた質問

2.1 後ろ向き推論に関して

後ろ向き推論は、ある仮説を与え、それが現在のアサーション集合において成り立つかをルールを用いて検証していくため、英語による質問を仮説に当てはめればよい。

ここで問題となるのは、英語の質問の内容である。後ろ向き推論はルールの後件部を見て、その前件部がワーキングメモリにあるかとバックトラックしていくので、質問内容が後件部の内容でない場合は処理されない。以下に具体例を用いる。

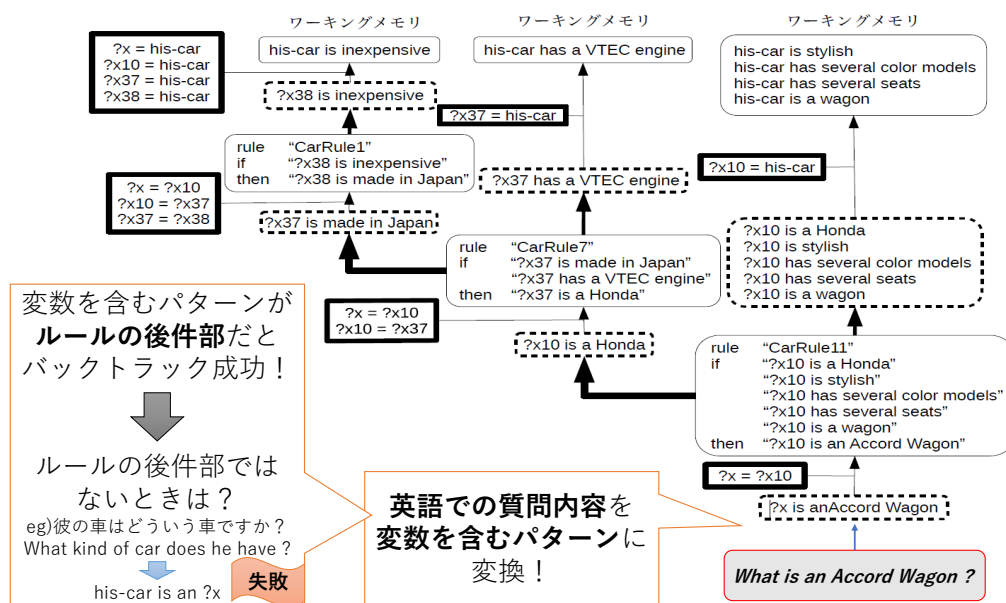


図1 英語の質問を仮説に変換させる

質問の内容が「アコードワゴンは何ですか?」と場合、「`?x is an Accord Wagon`」に変換でき、それはルールの後件部の内容なので、そのままバックトラックを行えるが、「彼が持っている車は何ですか?」という場合、「`his-car is ?x`」というパターンに変換でき、これはルールの後件部にはないので、バックトラックに失敗する。(正しくは、`his-Car is`

inexpensive と出力されてしまう.)

また、質問内容が後件部の内容ではなく、前件部の内容に関わる内容の場合、ワーキングメモリ内のアサーションを上から照らし合わせていき、照合するアサーションで出力される。上記で述べた「his-Car is ?x」も、ワーキングメモリの1番上に「his-Car is inexpensive」が存在するため間違えて出力されるのである。また、これらより、同じ文の形を持っている場合、1つ目のアサーションにヒットすると、2つ目まで進まないということも同様に成り立つ。以下を見てもらいたい。

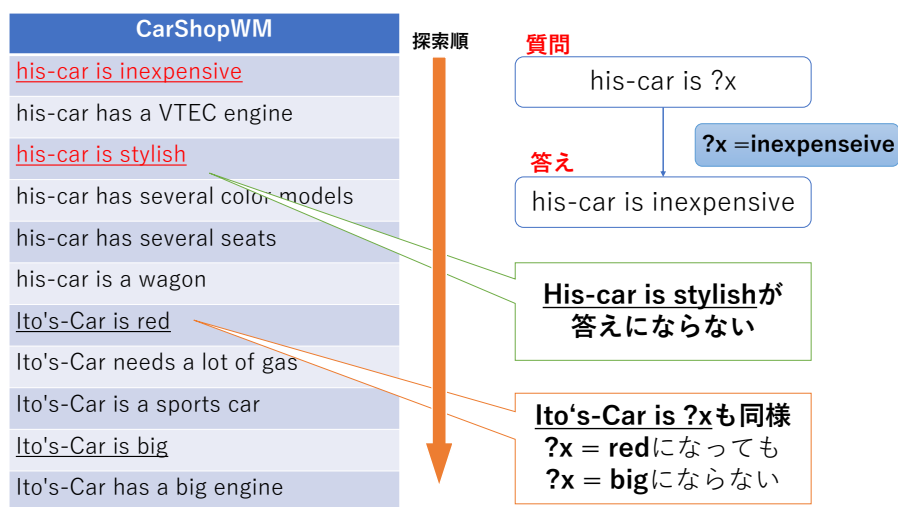


図2 WM内の探索順序

質問の内容によっては、「his-car is stylish」に基づく内容を問いたくなるだろう。そういった場合、「his-car is ?x」という形をしているアサーションが複数ある場合に、何らかの制限を加える必要がある。

ここで、ワーキングメモリにおけるアサーションに対する質問内容を具体化してみる。「彼の車は安価です。」と「彼の車はスタイリッシュです。」というアサーションが存在する際、質問とは何であろうか? 「彼の車は安価ですか?」と「彼の車はスタイリッシュですか?」の Yes/No が返事となる質問ではないだろうか? 逆に、「彼の車は何ですか?」と質問して「彼の車は安価です。」と返ってきたら、むむむと思うのではないだろうか。だが、「伊藤の車は何色ですか?」と質問したら、「伊藤の車は赤色です。」と返答がほしい。

質問内容を、

1. Yes/No で答えることができる質問
2. 具体的な答えを返す質問

に分類した際、2 の”具体的な答えを返す質問”に応えられるように、ワーキングメモリ内のアサーションの並び方や内容など、工夫する必要がある。

”英語の質問”を”変数を含むパターン”に落とし込む方法は前回の課題のようにクエリーに分けて、適切に並び替える方法をとる。

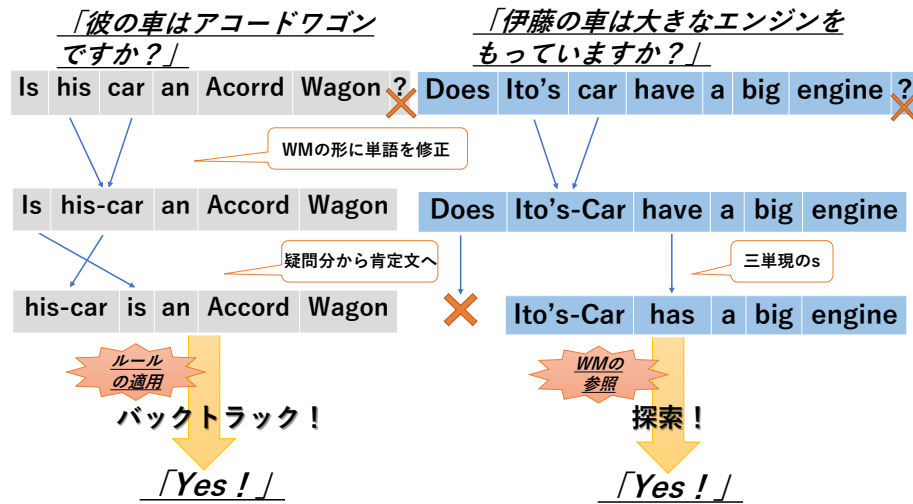


図3 Yes/No で回答する疑問文の処理

疑問分を肯定文に並び変えるのは、SVC 構文の場合は、先頭の 2 文字を並び替えだけで、SVO 構文の場合は、先頭文字を削除するだけである。三人称単数の置き換え (have を has や need を needs など) など細かい処理もある。図 3 における左の例文と右の例文の大きな処理の違いは、肯定文のパターンに並び変えた後である。左の例文では、ルールの後件部に該当する内容なので、バックトラックが行われるが、右の例文では、ルールの後件部には存在しない内容なので、ワーキングメモリを参照される。ワーキングメモリの参照のされ方は、上記で述べたように、上から処理される。この Yes/No の答えを返す内容では、この左右の例文に大きな違いは存在しないが、次の”具体的な答えを返す質問”の際には、これが大きく関わってくる。

具体的な答えを返す質問も似たような処理で英語から変数を含むパターンに落とし込むことができる。

具体的な答えを返す質問、つまり、質問文に What が含まれるものに対しては、疑問分から肯定文に変換する際に変数?x を適切な場所に配置する。図 4 の左の質問は、ルールの後件部の内容なので、バックトラックが行われ、右の質問ではワーキングメモリが参照される。どちらも変数が具体化され、答えが出力される。ここで問題になってくるのは、図 4 の

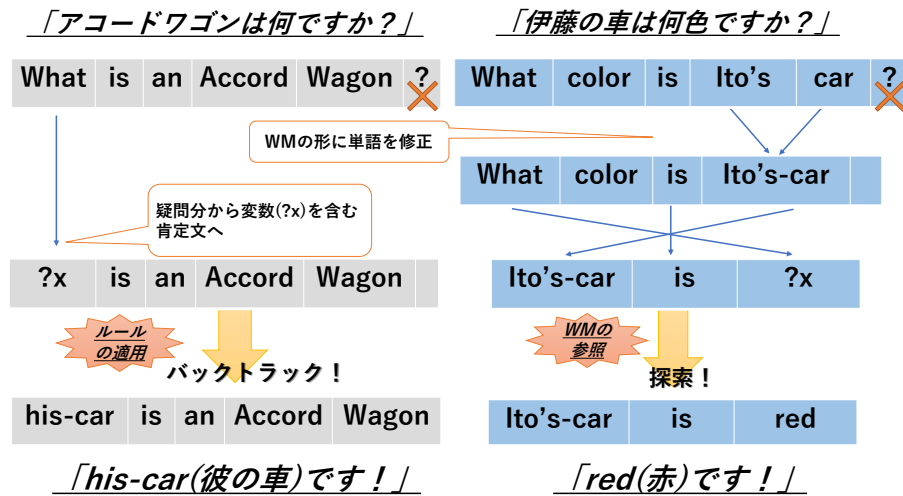


図 4 具体的な答えを返す質問の処理

左側の質問のような、変数を入れた場合のルールの適用によるバックトラックシステムである。バックトラックを行うためには、ルールの後件部の内容でなければならない、それ以外では、バックトラックが行われない。つまり、「?x is an Accord Wagon」は質問できて、「his-car is ?x」は使えないと述べたこのレポートの一番始めの内容の問題が生じる。これはこの推論システムの欠点ともいえる。

2.2 前向き推論に関して

前向き推論では、ワーキングメモリのアサーション集合とルールにより生成される新たなアサーションを、ワーキングメモリに次々と格納していくので、質問に答える時には、質問内容と最終的に出来上がったワーキングメモリに格納されているアサーションを1つ1つ照らし合わせればよい。質問の種類も Yes/No と答えるものと、具体的な値を答える方法の2種類であり、後ろ向き推論と同じ方法が使える。また、前向き推論は、最終的なワーキングメモリだけを参照することから、後ろ向き推論のように、質問がルールの後件部に当てはまるかどうかを考える必要もない。

3 実装

英語の質問を変数を含むパターンに変換し、解析をするメソッド NaturalLanguage メソッドを作成。メイン文で入力された英語の質問 (String 型) に対して実行し、答えを出力

する。英語の質問内容を変数を含むパターンに落とす流れは、後ろ向き推論でも前向き推論でも内容は同じであるが、後ろ向き推論では、実際に推論を行う backwardChain メソッドを最後に使うのに対し、前向き推論では、先に forwardChain メソッドを呼び出して推論を行ってから、NaturalLanguage メソッドを呼び出す。そのメソッドを機能ごとに分けて以下に示す。

3.1 手法 1

まずは、NaturalLanguage メソッドの始めの処理をソースコード 1 に示す。

ソースコード 1 NaturalLanguage メソッドその 1

```
1 public static void NaturalLanguage(String equestion) {
2     /**
3     * 1. "英語の質問:s"を"変数含むパターン"に置き換える
4     * 2. その"変数を含むパターン"を解析する:rb.backwardChain()を実行
5     */
6     //解
7     ArrayList<String> tokenList = new ArrayList<>();
8     //今どこを指しているか
9     int tokenPoint = 0;
10
11     //前処理
12     if(equestion.contains("'s c")) {
13         equestion = equestion.replace("'s c", "'s-C");
14     }
15     else if(equestion.contains("his")) {
16         equestion = equestion.replace("his c", "his-c");
17     }
18     else if(equestion.contains("my")) {
19         equestion = equestion.replace("my c", "my-c");
20     }
21
22     //1.まずはトークンに分解して,
23     StringTokenizer stoken = new StringTokenizer(equestion);
24     // トークンの数を保存
25     int tokenSize = stoken.countTokens() - 1; //最後の?は除く！
```

NaturalLanguage メソッドでは、まず前処理として、英文とアサーションの違いを直す。「Ito's-Car」は「Ito's car」に分け、「his-car」は「his car」に、「my-car」は「my car」に分けている。その後、String 型のアサーションをトークンに分解し、1 つのアサーションがいくつかのトークンに分解できたかを保持しておく。

次に, NaturalLanguage メソッドの中間部, トークンの並び替えの処理をソースコード 2 に示す.

ソースコード 2 NaturalLanguage メソッドその 2

```
1      //2.トークンの並び替え,
2      String firstToken = token.nextTok();
3      tokenPoint ++;
4      String secondToken = token.nextTok();
5      tokenPoint ++;
6      if(firstToken.equals("Is")) {
7          tokenList.add(secondToken);
8          tokenList.add("is");
9      }
10     else if(firstToken.equals("What")) {
11         rb.qFlag = 1;
12         if(secondToken.equals("color")) {
13             String thirdToken = token.nextTok();
14             tokenPoint ++;
15             tokenList.add(token.nextTok());
16             tokenPoint ++;
17             tokenList.add(thirdToken);
18             tokenList.add(" ?x");
19         }
20         else if(secondToken.equals("does")) {
21             tokenList.add(token.nextTok());
22             tokenPoint ++;
23             String thirdToken = token.nextTok();
24             tokenPoint ++;
25             if(thirdToken.equals("have")) {
26                 tokenList.add("has");
27             }
28             tokenList.add("?x");
29         }
30         else if(secondToken.equals("is")) {
31             tokenList.add("?x");
32             tokenList.add("is");
33         }
34     }
35     else if(firstToken.equals("Does")) {
36         String thirdToken = token.nextTok();
37         tokenPoint ++;
38         //三単現のs
39         if(thirdToken.equals("have")) {
40             thirdToken = "has";
41         }
42         else {
43             thirdToken = thirdToken.replace(thirdToken, thirdToken+"s");
```

```

44         }
45         tokenList.add(secondToken);
46         tokenList.add(thirdToken);
47     }

```

質問文ははじめのトークンが何かで大きく分岐できる。What で始まれば”具体的な値を問う質問”であり, Is で始まれば”Yes/No 返事の質問”の SVC 型, Does で始まれば”Yes/No 返事の質問”の SVO 型となる。もちろん What のあとで, 同様に”is”と”does”で分けられる。今回, 分解したトークンのどこを指しているかを int 型の tokenPoint で保存しておく。これは, 以下で述べられる最後の処理をする際に用いられる。

最後に, NaturalLanguage メソッドの最終部の処理を示す。ここは, 後ろ向き推論と前向き推論で異なる点があるので, まずは後ろ向き推論に関して, ソースコード 3 に示す。

ソースコード 3 NaturalLanguage メソッドその 3(後ろ向き推論の場合)

```

1      //3.toString で最後に合体させる(今回はあくまでStrig の文字列にする.)
2      // 格納
3      for(int i = tokenPoint; i < tokenSize; i++) {
4          tokenList.add(stoken.nextToken());
5      }
6      // ArrayList → String 文字へ
7      String patarn = tokenList.toString();
8      patarn = patarn.replace("[", "");
9      patarn = patarn.replace("]", "");
10     patarn = patarn.replace(", ", "");
11     System.out.println("patarn = " + patarn);
12
13     //解析
14     StringTokenizer patarns = new StringTokenizer(patarn, ",");
15     ArrayList<String> queries = new ArrayList<String>();
16     for(int i = 0 ; i < patarns.countTokens();){
17         queries.add(patarns.nextToken());
18         System.out.println("queries = " + queries);
19     }
20     rb.backwardChain(queries); //このqueries は String の文字列
21 }

```

今回は, 前回の SemanticNet プログラムのように, Tail, Lavel, Head の 3 つに分けて当てはめるのではなく, 1 つの String 文として解析をするので, 分解してトークンを 1 つの String 文にもどす必要がある。tokenList に各トークンを格納していくが, ソースコード 1,2 で述べた現在のトークンの位置を保存する tokenPoint を用い, まだ tokenList に代入されていない残りのトークンを代入する。List を 1 つの String にするには, toString

メソッドとリストに表現される”[かぎカッコ]”と”, 区切り文字”を取り除くことで実現している。

次に, NaturalLanguage メソッドの最終部の処理の前向き推論に関して, 後ろ向き推論と異なる箇所をソースコード 4 に示す。

ソースコード 4 NaturalLanguage メソッドその 3(前向き推論の場合)

```
1  /*(注意)
2    * Yes/No 返事の場合は, そのままでいいんだけど,
3    * 「my-car has ?x」のときに, 「?x= a VTEC engine」とかで複数トークンになる
4    * ので, What の場合は別にしないといけない
5    */
6
7  //what に質問
8  if(patarn.contains("?x")) {
9
10     ArrayList<String> newWorkingMemory = new ArrayList<>();
11
12     for(int i = 0; i < rb.wm.memorySize(); i++) {
13         ArrayList<String> newAssertion = new ArrayList<>();
14         StringTokenizer wmtoken = new StringTokenizer(rb.wm.getValue(i));
15
16         String wmfirstToken = wmtoken.nextToken();
17         newAssertion.add(wmfirstToken);
18
19         String wmsecondToken = wmtoken.nextToken();
20         String wmthirdToken = wmtoken.nextToken();
21         if(wmsecondToken.equals("is") && wmthirdToken.equals("a")) {
22             newAssertion.add("is-a");
23         }
24         else if(wmsecondToken.equals("is") && wmthirdToken.equals("an")) {
25             newAssertion.add("is-an");
26         }
27         else if(wmsecondToken.equals("has") && wmthirdToken.equals("a")) {
28             newAssertion.add("has-a");
29         }
30         else if(wmsecondToken.equals("has") && wmthirdToken.equals("an")) {
31             newAssertion.add("has-an");
32         }
33         else if(wmthirdToken.equals("several")) {
34             newAssertion.add(wmsecondToken);
35         }
36         else {
37             newAssertion.add(wmsecondToken);
38             newAssertion.add(wmthirdToken);
```

```

39     }
40     //System.out.println("残りトークンの数 = " + wmtoken.countTokens());
41     if(wmtoken.countTokens() > 1) {
42         String wmforthToken = wmtoken.nextToken();
43         String wmfifthToken = wmtoken.nextToken();
44         newAssertion.add(wmforthToken + "-" + wmfifthToken);
45     }
46     else if(wmtoken.countTokens() == 1){
47         newAssertion.add(wmtoken.nextToken());
48     }
49     //System.out.println("newAssertion = " + newAssertion);
50     // ArrayList → String 文字へ
51     String stringAssertion = newAssertion.toString();
52     stringAssertion = stringAssertion.replace("[", "");
53     stringAssertion = stringAssertion.replace("]", "");
54     stringAssertion = stringAssertion.replace(",", "");
55     //System.out.println("stringAssertion = " + stringAssertion);
56     newWorkingMemory.add(stringAssertion);
57
58     //System.out.println("newWorkingMemory = " + newWorkingMemory);
59     //解析
60     for(int i = 0; i < newWorkingMemory.size(); i++) {
61         (new Matcher()).matching(patarn, newWorkingMemory.get(i));
62     }
63 }
64
65 //Yes/No で答える質問
66 else {
67     //解析
68     boolean flag = false;
69     for(int i = 0; i < rb.wm.memorySize(); i++) {
70         if((new Matcher()).matching(patarn, rb.wm.getValue(i))) {
71             System.out.println("答え = Yes");
72             flag = true;
73         }
74     }
75     if(!flag) {
76         System.out.println("答え = No");
77     }
78 }

```

前向き推論では、リスト化されたワーキングメモリ内のアサーションを1つずつ検索していき、Matcher クラスの matching メソッドを呼び出すことで答えを出力させている。ただし、ここで”What で聞かれた具体的な値を返す質問”と”Yes/No で返す質問”で大きな違いが生じる。What で聞かれた質問、つまり、変数 (?x) をマッチングで具体化させると

き、これまでの matching メソッドと同様に、1 つのトークン単位で行っているので、「?x = a VTEC engine」のような、複数のトークンをまとめて出力させることはできない。なぜ”Yes/No で答える質問”ではそれが生じないのかというと、Yes/No ではすでにトークンが指定数存在するからである。以下をみてもらいたい。

(具体例)

肯定文の場合、

「my car has a VTEC engine.」

Yes/No で返す場合の質問の場合、

「Does my car have a VTEC engine ?」 → 「my car has a VTEC engine」 と変換可能
疑問文の場合、

「What does my car have ?」 → 「my car have ?x」 としか変換できない...

そのため、「VTEC engine」のような 2 つのトークンにまたがるものは、1 つにまとめないといけない。その時に、「has」と「a」は「has-a」のようにまとめ、「several」といった複数の意味をもつ前置詞は取り除いている。

4 実行例

4.1 後ろ向き推論:その 1

後ろ向き推論システムにおいて、日本語で言うと、「彼の車はアコードワゴンですか?」と質問したときの実行結果が以下ようになる。

-
- 1 質問を入力してください
 - 2 Is his-car an Accord Wagon ?
 - 3 質問内容 = Is his-car an Accord Wagon ?
 - 4 patarn = his-car is an Accord Wagon
 - 5 queries = [his-car is an Accord Wagon]
 - 6 Hypothesis:[his-car is an Accord Wagon]
 - 7 Success RULE
 - 8 Rule:CarRule11 [?x40 is a Honda, ?x40 is stylish, ?x40 has several color models, ?x40 has several seats, ?x40 is a wagon]→?x40 is an Accord Wagon <=> his-car is an Accord Wagon
 - 9 Success RULE
 - 10 Rule:CarRule7 [?x47 is made in Japan, ?x47 has Honda's logo]→?x47 is a Honda <=> ?x40 is a Honda
 - 11 Success RULE

```

12 Rule:CarRule1 [?x48 is inexpensive]—>?x48 is made in Japan <=> ?x47 is made in
    Japan
13 Success WM
14 his-car is inexpensive <=> ?x48 is inexpensive
15 tmpPoint: 13
16 Success:?x47 is made in Japan
17 tmpPoint: -1
18 Success RULE
19 Rule:CarRule8 [?x66 is made in Japan, ?x66 has a VTEC engine]—>?x66 is a Honda
    <=> ?x40 is a Honda
20 Success RULE
21 Rule:CarRule1 [?x67 is inexpensive]—>?x67 is made in Japan <=> ?x66 is made in
    Japan
22 Success WM
23 his-car is inexpensive <=> ?x67 is inexpensive
24 tmpPoint: 13
25 Success:?x66 is made in Japan
26 Success WM
27 his-car has a VTEC engine <=> ?x66 has a VTEC engine
28 tmpPoint: 20
29 Success:?x40 is a Honda
30 Success WM
31 his-car is stylish <=> ?x40 is stylish
32 tmpPoint: 3
33 Success:?x40 is stylish
34 Success WM
35 his-car has several color models <=> ?x40 has several color models
36 tmpPoint: 4
37 Success:?x40 has several color models
38 Success WM
39 his-car has several seats <=> ?x40 has several seats
40 tmpPoint: 5
41 Success:?x40 has several seats
42 Success WM
43 his-car is a wagon <=> ?x40 is a wagon
44 qFlag = 0
45 Yes

```

この質問内容はルールの後件部の内容に関わるので、バックトラックが行われていることがわかる。結果正しく出力されていることが確認できる。

4.2 後ろ向き推論:その2

後ろ向き推論において、日本語で言うと、「伊藤の車は赤いですか?」と質問したときの実行結果が以下のようになる。

```

1 Is Ito's car red ?
2 質問内容 = Is Ito's car red ?
3 patarn = Ito's-Car is red
4 queries = [Ito's-Car is red]
5 Hypothesis:[Ito's-Car is red]
6 Success WM
7 Ito's-Car is red <=> Ito's-Car is red
8 qFlag = 0
9 Yes

```

この質問内容はルールの後件部には存在しないので、ワーキングメモリだけが参照されていることがわかる。その結果、正しく出力されていることがわかる。

4.3 前向き推論:その 1

前向き推論において、日本語で言うと、「私の車は何を持っていますか」と質問したときの実行結果が以下ようになる。

```

1 質問を入力してください
2 What does my car have ?
3 質問内容 = What does my car have ?
4 patarn = my-car has ?x
5 newWorkingMemory = [my-car is inexpensive, my-car has-a VTEC-engine, my-
    car is stylish, my-car has color-models, my-car has seats, my-car is-a
    wagon, my-car is made in-Japan, my-car is-a Honda, my-car is-an Accord
    -Wagon]
6 答え = color-models
7 答え = seats

```

英語の質問文が、変数を含むパターンに正しく変換された後、「has-a」や「VTEC-engine」など通常のワーキングメモリから改良した新しいワーキングメモリが生成され、それにより正しい答えが出力されている。後ろ向き推論とは異なり、リスト構造のワーキングメモリを 1 つずつ参照しているだけなので、複数の回答の出力も容易に実現されていることがわかる。

4.4 前向き推論:その 2

前向き推論において、日本語で言うと、「私の車は VTEC エンジンを持っていますか」と質問したときの実行結果が以下ようになる。

```

1 質問を入力してください

```

- 2 Does my car have a VTEC engine ?
 - 3 質問内容 = Does my car have a VTEC engine ?
 - 4 secondToken = my-car
 - 5 patarn = my-car has a VTEC engine
 - 6 答え = Yes
-

「前向き推論:その 1」の実行結果とは異なり, 新しいワーキングメモリを作成する必要がないので, そのまま出力させている. 正しく Yes の答えが返ってきていることが確認できる.

5 考察

今回は前向き推論と後ろ向き推論において, 同じ質問内容でも, その推論システムの機能の違いから, 違うアプローチが必要になることがわかった. 後ろ向き推論における質問の内容には大きく 4 パターンに分けることができた. 以下を参考にしてもらいたい.

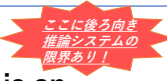
	ルールの適用 (バックトラック)	ワーキングメモリ (WM)のみ参照
質問の内容 Yes/ No返事	eg) Is his car an Acorrd Wagon ?	eg) Does Ito's car have a big engine ?
質問の内容 具体的な値の 返事	eg)  What is an Acorrd Wagon ?	eg) What color is Ito's car ?

図 5 後ろ向き推論の質問の分類

これまでさんざん述べてきたように, 後ろ向き推論が仮説を検証することにおいて, 全ての文章に対応していない. あくまで, 仮説にできる内容はルールの後件部の内容であり, それ以外はワーキングメモリを参照するだけである.

また, 今回の自分のように, 英語の質問文をトークンに分けて処理する流れだと, 「?x = a VTEC engine」のような複数のトークンにまたがってしまう答えに答えることがとても大変になってしまった. 実際に前向き推論では新しくワーキングメモリを作らざるを得なくなり, 後ろ向き推論に当たってはうまく答えを出力できていない状態である.

6 感想

人間の認識システムをプログラムに落としこんだとされるルールベースモデルにおける後ろ向き推論システム, これにおいて, 検証できる仮説の内容が限定されてしまうことに驚きを隠せない. 自分の質問のさせ型に全く別のアプローチを用いるか, 新たな知識(ルール)を構築させることで, この現状を打破できるのかどうかについては今後に期待していきたい.

また, 2つの課題にわたり英語の自然言語を扱ってきたが, 今回のトークンの問題のようにまだまだ再現性が低い状態が続いた. 英語でこの大変さなので, 日本語にいたっては, いくら時間を使っても足りないであろうと感じた. 大量のコーパスを用い, 研究を進める自然言語学者は本当に大変な研究をしていると思った.

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] HashMap –著: Java コード入門
<http://java-code.jp/232>
- [3] Google 翻訳 –著: Google
<https://translate.google.com/?hl=ja>
- [4] java substring メソッド –著: Engineer.club
<https://engineer-club.jp/java-substring>