知能プログラミング演習 II 課題 3

グループ 8
29114116 増田大輝

2019 年 10 月 25 日

■提出物　rep3

■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 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.

本課題では, 実装を行わないため,「実装」の節を省略することとする. 代わりに, 前向き推論と後ろ向き推論について説明する節を設ける．また, 実行例を示した上でルールが選択される過程を説明するため, 通常「実行例」の節を「実行例とルールが選択される過程の説明」として強調する.

2.1　手法

最初に, 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」のプログラムの動作確認を行い, 教科書を参考にそれぞれの推論の違いについて説明する. 次に, 両者のプログラムの実行例を示しつつ,LibreOffice の Draw によって作成した図を交えながらルールが選択される過程の説明を行う.

2.2　前向き推論と後ろ向き推論の違い

前向き推論では, ルールインタプリタ内部の照合過程において, ルールの前件を満たすようなアサーションがワーキングメモリ内に存在する場合にルールが発火し, 後件の評価やアクションの実行が行われる. この一連の流れが, 追加されるアサーションがなくなるまで繰り返し行われる. 前向き推論は, 以下のステップにより行われる.

1. ワーキングメモリ内のアサーションと選択されたルールの前件がマッチするかチェックする.
2. マッチしたルールの後件を具体化してアサーションとしてワーキングメモリに追加/アクションを実行する.
3. ルールが実行されなくなるまで 1 に戻って照合過程を繰り返し行う.
4. 新たに実行可能なルールが無くなった場合は終了する.

すなわち, ワーキングメモリ内部に蓄えられたアサーションは知識に当たり, これらの知識を用いて実行可能なルールから新たな知識が導出されることにより, 逐次的な推論が行われる. したがって, 蓄えられた知識とルールから新たな知識を獲得し, その知識によって実行可能なルールがあれば, そこからさらなる知識を獲得することができるのである. 前向き推論では, 与えられたアサーションとルールのパターンを照合することによって得られる全ての知識をワーキングメモリに追加する. この特徴から, 前向き推論では, 網羅的に知識を獲得することができるというメリットがある. その一方で, 特定の知識のみを得たい際には, 必要のない知識の照合も含めた推論を行う可能性が十分にあるため, 非効率であるというデメリットがある.

後ろ向き推論では, 与えられた仮説が現在のワーキングメモリ内のアサーション集合, すなわち保持している全ての知識において成立するか否かをルールを用いて調べることによって駆動する. すなわち, 以下の手順で推論が行われる.

1. 与えられた仮説とマッチするルールの後件を探索する.
2. マッチしたルールの前件とワーキングメモリ内部のアサーションのマッチングを行う.
3. 成功した場合は仮説を真として判定.

失敗した場合はマッチングに失敗した前件を新たな仮説とし,1 から繰り返し, 真偽

判定を行う.

4. 全ての仮説が真と判断できなかった場合は最初の仮説を偽として判定.

以上をまとめると, 後向き推論では得たい情報を仮説として, その仮説を導くためのルールを順に辿って行くことで, 最終的に現在の知識から真偽や具体化を得る. この特徴から, 後向き推論では, 特定の知識を導き出す際に, 必要な知識のみを推論によって獲得するため効率的であるといえる. その一方で, 全ての知識についてルールが適用されるとは限らないので, 知識獲得における網羅性はない.

以上が, 前向き推論と後向き推論の特徴であり, それぞれの違いは推論の駆動のさせ方とその違いにより生じるメリット・デメリットが互いに相反している点にある. 具体的には, 前向き推論ではワーキングメモリ内のアサーションの変化がルールが発火する原因となっており, 後向き推論では仮説の変化がルールを発火させる原因となる. 以下にそれぞれの代表的な特徴を比較して違いがわかる表を示す.

	照合過程でチェックする部分	ルール発火の原因
前向き推論	ルールの前件とアサーション	ワーキングメモリの変化
後向き推論	ルールの後件と仮説	仮説の変化

表 1　前向き推論と後向き推論の駆動方法の違い

	メリット	デメリット
前向き推論	網羅的に知識を獲得	余分な知識も獲得するため非効率
後向き推論	効率的な知識獲得	得られる知識が限定的

表 2　前向き推論と後向き推論のメリット・デメリット

2.3　実行例とルールが選択される過程の説明

2.3.1　前向き推論の実行例について

以下に前向き推論の実行結果を示す.

ソースコード 1　前向き推論の実行結果

```
1    ~/Programming2/Work4/ForwardChain
2    ●java RuleBaseSystem【 masuda-branch 】
```

```
3   ADD:my-car is inexpensive
4   ADD:my-car has a VTEC engine
5   ADD:my-car is stylish
6   ADD:my-car has several color models
7   ADD:my-car has several seats
8   ADD:my-car is a wagon
9   CarRule1 [?x is inexpensive]->?x is made in Japan
10  CarRule2 [?x is small]->?x is made in Japan
11  CarRule3 [?x is expensive]->?x is a foreign car
12  CarRule4 [?x is big, ?x needs a lot of gas]->?x is a foreign car
13  CarRule5 [?x is made in Japan, ?x has Toyota's logo]->?x is a
        Toyota
14  CarRule6 [?x is made in Japan, ?x is a popular car]->?x is a
        Toyota
15  CarRule7 [?x is made in Japan, ?x has Honda's logo]->?x is a Honda
16  CarRule8 [?x is made in Japan, ?x has a VTEC engine]->?x is a
        Honda
17  CarRule9 [?x is a Toyota, ?x has several seats, ?x is a wagon]->?
        x is a Carolla Wagon
18  CarRule10 [?x is a Toyota, ?x has several seats, ?x is a hybrid
        car]->?x is a Prius
19  CarRule11 [?x is a Honda, ?x is stylish, ?x has several color
        models, ?x has several seats, ?x is a wagon]->?x is an Accord
         Wagon
20  CarRule12 [?x is a Honda, ?x has an aluminium body, ?x has only 2
        seats]->?x is a NSX
21  CarRule13 [?x is a foreign car, ?x is a sports car, ?x is stylish,
         ?x has several color models, ?x has a big engine]->?x is a
        Lamborghini Countach
22  CarRule14 [?x is a foreign car, ?x is a sports car, ?x is red, ?x
         has a big engine]->?x is a Ferrari F50
23  CarRule15 [?x is a foreign car, ?x is a good face]->?x is a Jaguar
         XJ8
24  apply rule:CarRule1
25  Success: my-car is made in Japan
26  ADD:my-car is made in Japan
27  apply rule:CarRule2
28  apply rule:CarRule3
```

```
29      apply rule:CarRule4
30      apply rule:CarRule5
31      apply rule:CarRule6
32      apply rule:CarRule7
33      apply rule:CarRule8
34      Success: my-car is a Honda
35      ADD:my-car is a Honda
36      apply rule:CarRule9
37      apply rule:CarRule10
38      apply rule:CarRule11
39      Success: my-car is an Accord Wagon
40      ADD:my-car is an Accord Wagon
41      apply rule:CarRule12
42      apply rule:CarRule13
43      apply rule:CarRule14
44      apply rule:CarRule15
45      Working Memory[my-car is inexpensive, my-car has a VTEC engine, my
            -car is stylish, my-car has several color models, my-car has
            several seats, my-car is a wagon, my-car is made in Japan, my-
            car is a Honda, my-car is an Accord Wagon]
46      apply rule:CarRule1
47      apply rule:CarRule2
48      apply rule:CarRule3
49      apply rule:CarRule4
50      apply rule:CarRule5
51      apply rule:CarRule6
52      apply rule:CarRule7
53      apply rule:CarRule8
54      apply rule:CarRule9
55      apply rule:CarRule10
56      apply rule:CarRule11
57      apply rule:CarRule12
58      apply rule:CarRule13
59      apply rule:CarRule14
60      apply rule:CarRule15
61      Working Memory[my-car is inexpensive, my-car has a VTEC engine, my
            -car is stylish, my-car has several color models, my-car has
            several seats, my-car is a wagon, my-car is made in Japan, my-
```

```
                car is a Honda, my-car is an Accord Wagon]
62      No rule produces a new assertion
```

実行結果の前半では, 各推論システムの構成要素の初期化が行われている. 具体的には,3〜8 行目においてワーキングメモリ内のアサーション集合が初期化され,9〜23 行目においてはルールベースが初期化されている.

次に,24 行目以降の推論の実行が行われている後半部分について説明する. 説明のポイントとして, ルールが実行されている部分に注目する.

まず, はじめにルールが実行されているのは,24〜26 行目の部分である. 具体的には, ワーキングメモリ内のアサーション”my-car is inexpensive”によって CarRule1 が実行された形になっている. このルールの実行は以下の図の左側のようになる.

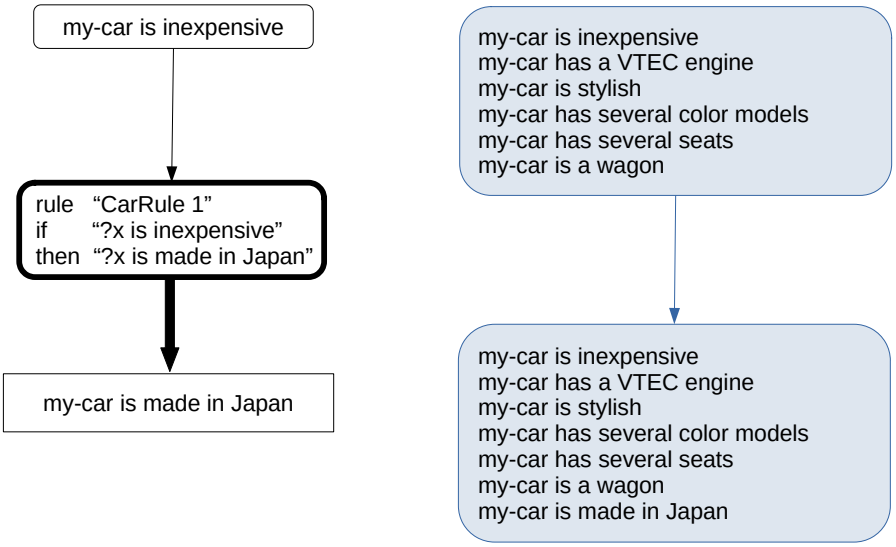

図 1　CarRule1 の実行とワーキングメモリの変化

また, 上図の右側のワーキングメモリの変化からも読み取れるように, ルールの実行結果として得られたアサーション”my-car is made in Japan”が実行後のワーキングメモリに追加されている. このことは, ルールを実行する際の前件にマッチングできる可能性のある知識が増えたことを意味する.

続いて,33〜35 行目に注目する. ここでは, ワーキングメモリ内のアサーション”my-car is made in Japan”と”my-car has a VTEC engine”によって CarRule8 が実行されている. したがって,CarRule1 の実行結果によって得られた知識を元に推論が駆動されている. このルールの実行は以下の図の左側のようになる.

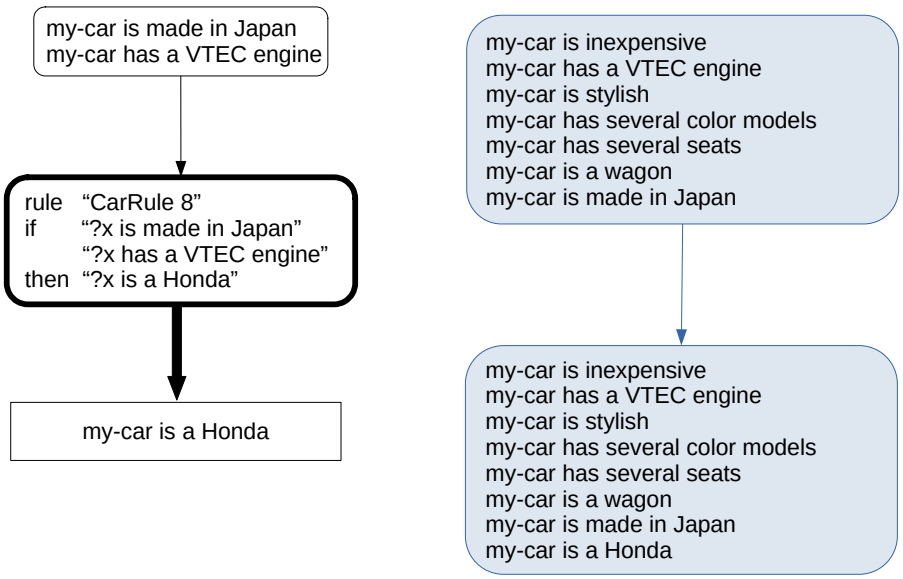

図 2　CarRule8 の実行とワーキングメモリの変化

ルールの実行結果として新たなアサーション”my-car is a Honda”がワーキングメモリに追加される.

同様に,38〜40 行目に着目する．ここでは, ワーキングメモリ内のアサーション 5 つを前件にマッチングさせて CarRule11 を実行している．このルールの実行は以下の図の左側のようになる.

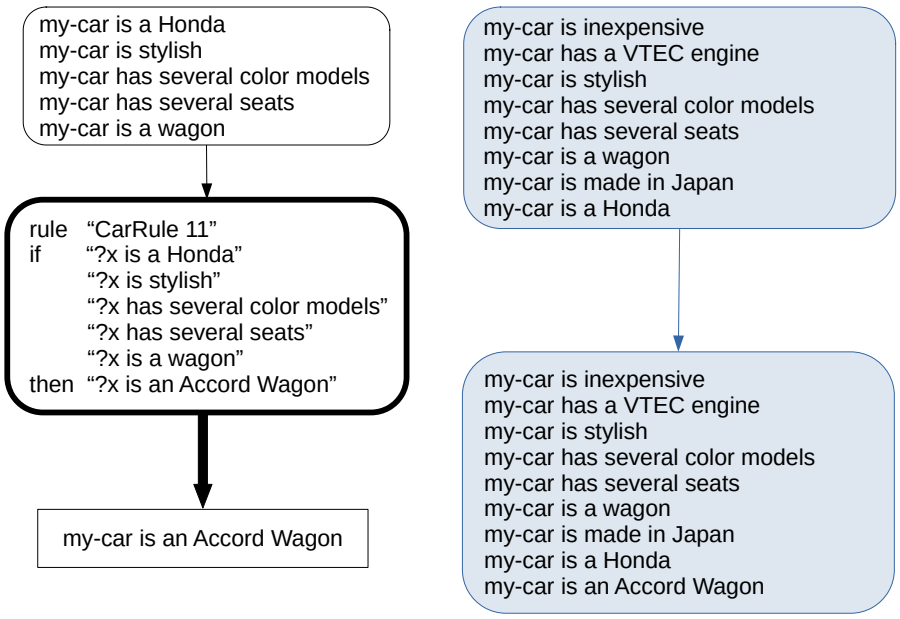

図 3　CarRule11 の実行とワーキングメモリの変化

この推論の結果として, 新たなアサーション”my-car is an Accord Wagon”が得られた.

その後, いかなるルールもマッチングせず,44 行目で照合過程が一度終了する. 続く 45 行目では, 現在のワーキングメモリ内部のアサーション集合の確認が行われている.
しかし, これで前向き推論のプログラムは終了しない. 46〜60 行目にあるように, もう一度照合過程が行われている. これは, 前の照合過程において, あるルールを実行することによって得られた知識を用いて, そのルールより以前にチェックされたルールが実行される可能性が残されているためである. この操作は, 前向き推論によって網羅的に知識が獲得されていくことを担保する. ただし, 今回の場合は, 一周目で全ての知識が獲得されたために, 二周目において新たに実行されるルールが存在せず, 62 行目にあるように推論が終了している.
以上の前向き推論における全ルールの実行過程を示したものを以下に示す.

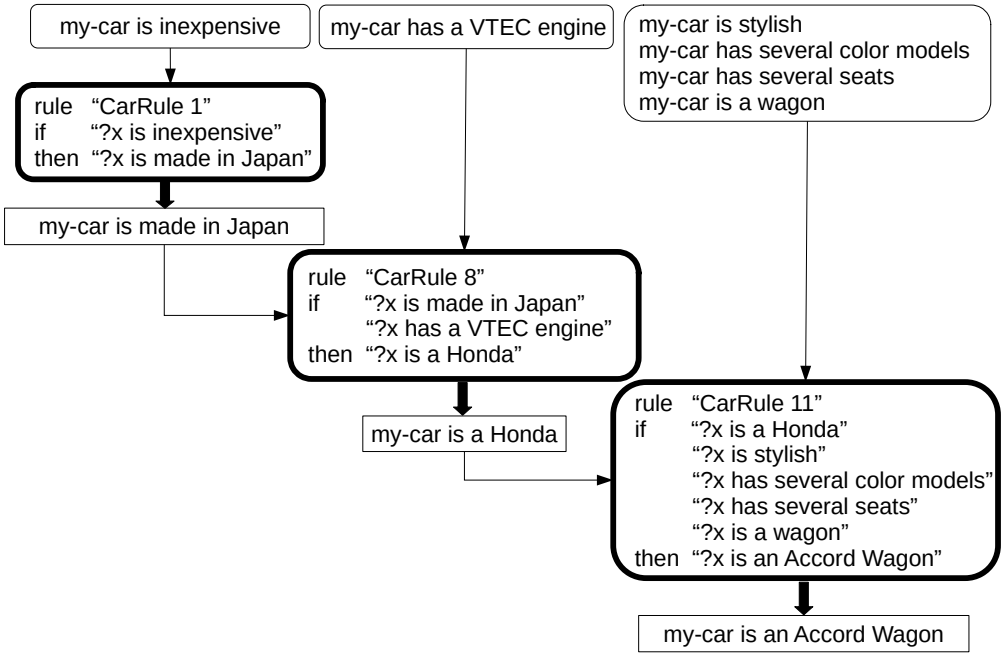

図 4　前向き推論の実行

2.3.2 後向き推論の実行例について

以下に, 後向き推論の実行結果を示す.

ソースコード 2　後向き推論の実行結果

```
1   ~/Programming2/Work4/BackwordChain
2   ●java RuleBaseSystem "?x is an Accord Wagon"【 masuda-branch 】
3   Hypothesis:[?x is an Accord Wagon]
4   Success RULE
5   Rule:CarRule11 [?x10 is a Honda, ?x10 is stylish, ?x10 has several
        color models, ?x10 has several seats, ?x10 is a wagon]->?x10
    is an Accord Wagon <=> ?x is an Accord Wagon
6   Success RULE
7   Rule:CarRule7 [?x17 is made in Japan, ?x17 has Honda's logo]->?x17
        is a Honda <=> ?x10 is a Honda
8   Success RULE
9   Rule:CarRule1 [?x18 is inexpensive]->?x18 is made in Japan <=> ?
    x17 is made in Japan
10  Success WM
11  his-car is inexpensive <=> ?x18 is inexpensive
12  tmpPoint: 12
13  Success:?x17 is made in Japan
14  tmpPoint: -1
15  Success RULE
16  Rule:CarRule8 [?x37 is made in Japan, ?x37 has a VTEC engine]->?
    x37 is a Honda <=> ?x10 is a Honda
17  Success RULE
18  Rule:CarRule1 [?x38 is inexpensive]->?x38 is made in Japan <=> ?
    x37 is made in Japan
19  Success WM
20  his-car is inexpensive <=> ?x38 is inexpensive
21  tmpPoint: 12
22  Success:?x37 is made in Japan
23  Success WM
24  his-car has a VTEC engine <=> ?x37 has a VTEC engine
25  tmpPoint: 19
26  Success:?x10 is a Honda
27  Success WM
```

```
28      his-car is stylish <=> ?x10 is stylish
29      tmpPoint: 3
30      Success:?x10 is stylish
31      Success WM
32      his-car has several color models <=> ?x10 has several color models
33      tmpPoint: 4
34      Success:?x10 has several color models
35      Success WM
36      his-car has several seats <=> ?x10 has several seats
37      tmpPoint: 5
38      Success:?x10 has several seats
39      Success WM
40      his-car is a wagon <=> ?x10 is a wagon
41      Yes
42      {?x38=his-car, ?x37=his-car, ?x10=his-car, ?x=his-car}
43      binding: {?x38=his-car, ?x37=his-car, ?x10=his-car, ?x=his-car}
44      tmp: ?x, result: his-car
45      Query: ?x is an Accord Wagon
46      Answer:his-car is an Accord Wagon
```

まず,2 行目では, コマンドとして java RuleBaseSystem "?x is an Accord Wagon" を実行しており, 条件を満たすような?x を問う形式になっている. この後向き推論では, この?x を変数束縛として具体化することによって得ることを目的とする. ゆえに, 実行例の 3 行目では, 大元の仮説として"?x is an Accord Wagon"が設定されている.
後向き推論では, 仮説とルールの後件をマッチングさせることによって次の仮説を導き出し, 仮説とワーキングメモリ内のアサーションのマッチングを行うことで変数束縛を行う. したがって, この 2 点に注目して実行例の解説を行うものとする. なお, バックトラック等の複雑な処理の解説も行うため, 説明量がやや多くなることを断っておく.

実行例の 4・5 行目では,CarRule11 の後件と仮説のマッチングが成功している. すなわち,"?x10 is an Accord Wagon"と"?x is an Accord Wagon"の間でマッチングが成功し,?x=?x10 として束縛される. 以下の図 5 に, このマッチングの様子を示す. その結果として,CarRule11 の前件 [?x10 is a Honda, ?x10 is stylish, ?x10 has severalcolor models, ?x10 has several seats, ?x10 is a wagon] が新たな仮定として導出される. これらの仮定が真となれば, 元の仮定も真となる.

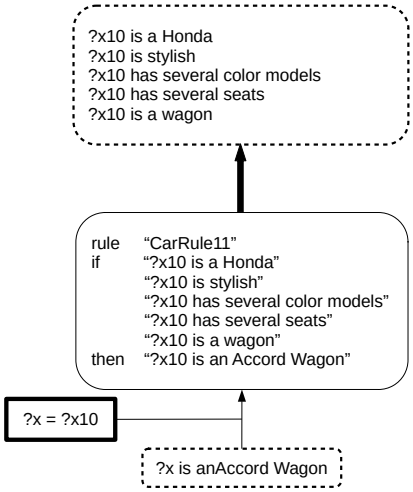

図 5　CarRule11 の後件と仮説のマッチング

次に, 実行例 6・7 行目において CarRule7 の後件と新たに導出された仮説の一つ”?x10 is a Honda”のマッチングが成功している. 詳細については以下の図 6 に示す. 結果とし

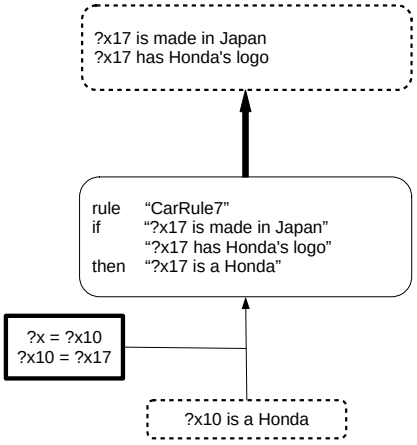

図 6　CarRule7 の後件と仮説のマッチング

て, 仮説”?x10 is a Honda”は [?x17 is made in Japan, ?x17 has Honda’s logo] に置き換わった.

同様にして, 実行例 8・9 行目においても以下の図 7 のように仮説が置き換わる.

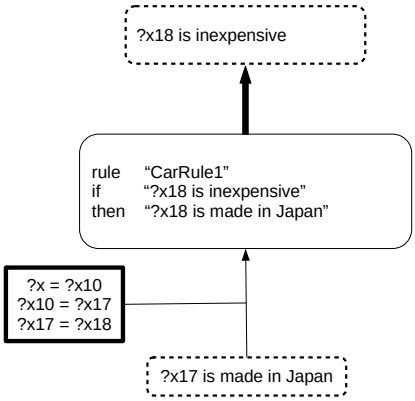

図 7　CarRule1 の後件と仮説のマッチング

次に,10〜12 行目において, 先ほどのマッチングの結果によって得られた仮説の具体化が行われている. すなわち,”?x18 is inexpensive”とワーキングメモリ内のアサーション”his-car is inexpensive” が一致することによって, 変数?x18=?x17=?x10=?x=his-car として具体化が得られる. 以下の図 8 に具体化の様子を示す. これにより, 他の仮説についても?x18=?x17=?x10=?x=his-car として具体化されたものが, ワーキングメモリ内のアサーション集合に存在すれば元の仮説が真となる. 13 行目では, この具体化を受けて仮説”?x17 is made in Japan”が成功したことが示されている.

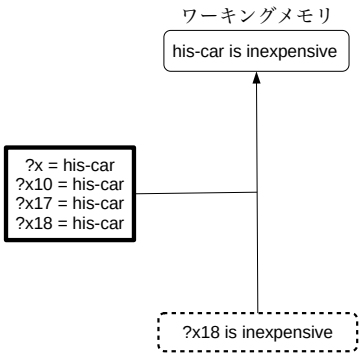

図 8　”?x18 is inexpensive”と”his-car is inexpensive”による具体化

しかし, ここで問題が発生する. 先ほど,?x18=?x17=?x10=?x=his-car による具体化が得られたが, CarRule7 により導かれた仮説の一つである”?x17 has Honda’s logo” について,?x17=his-car とした時のアサーション”his-car has Honda’s logo”が存在しないのである. したがって, 仮説”?x17 has Honda’s logo”は偽となり,tmpPoint は-1 を返すこととなる. 以下の図 9 に仮説のマッチングの失敗を図示する. この図から, ワーキングメモ

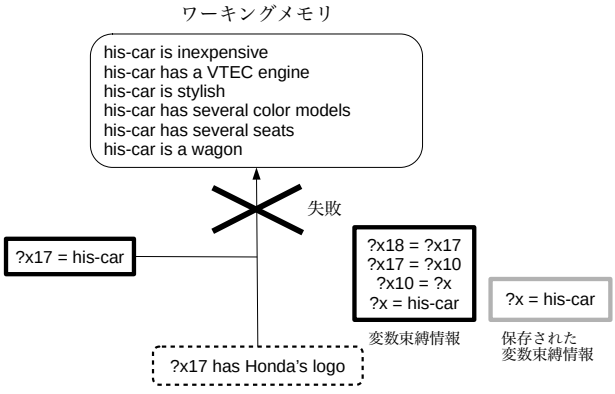

図 9　仮説”his-car has Honda’s logo”の失敗

リ内に該当アサーション”his-car has Honda’s logo”が存在しないことが確認できる．これにより,CarRule7 により, 仮説”?x10 is Honda”を導くことが不可能であることが示された.

仮説のチェックの失敗により, 一つ前の推論ステップに戻って CarRule7 の代わりとなる新たなルールを探す必要が生じる．この一連の処理のことをバックトラックと呼び, 先ほどユニフィケーションされた変数束縛は元に戻される．すなわち, 変数束縛情報は?x10=?x の状態に再び戻る．以下の図 10, バックトラックの実行を示す．バックトラッ

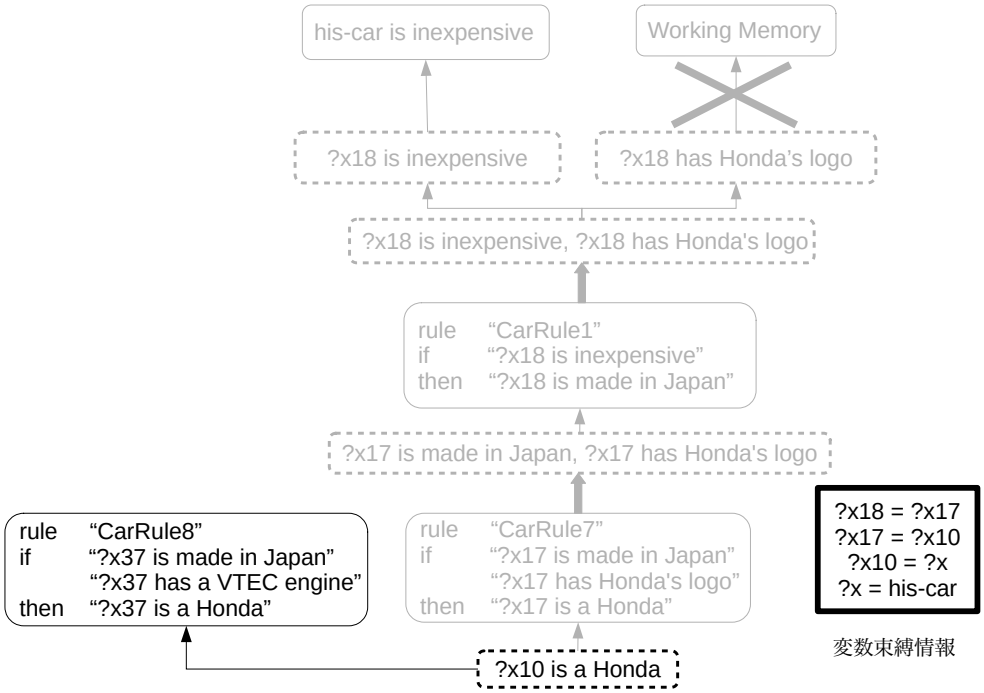

図 10　バックトラックの実行

クの実行によって, 再び仮説”?x10 is a Honda”を真にするような仮説を導出するルールを探すこととなった.

そして,15・16 行目において,CarRule8 の後件と”?x10 is a Honda”のパターンマッチングが成功する. これについても詳細は以下の図 11 に示すものとする.

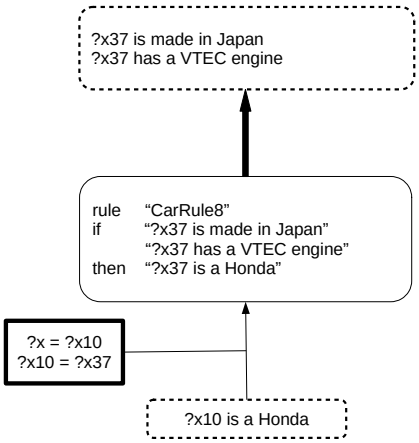

図 11　CarRule8 の後件と仮説のマッチング

同様に,17・18 行目でも CarRule1 の後件と”?x37 is made in Japan”のマッチングを示す. 図 12 を参照する.

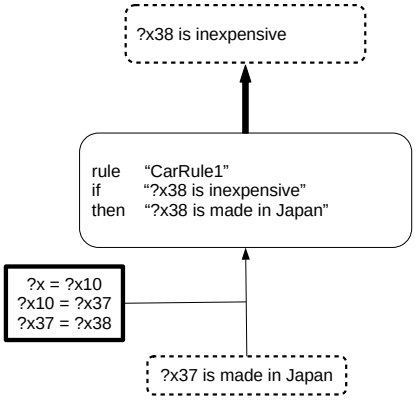

図 12　CarRule1 の後件と仮説のマッチング

実行例 19・20 行目でようやく”his-car is inexpensive”と”?x38 is inexpensive”によって具体化が得られる. すなわち,?x38=?x37=?x10=?x=his-car が得られる.

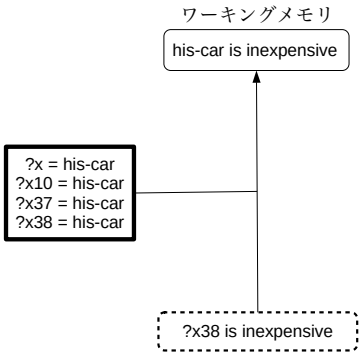

図 13 ”his-car is inexpensive”と”?x38 is inexpensive”によるユニフィケーション

先ほどのユニフィケーションの結果を受けて,?x37=his-car となるので, 仮説”?x37 has a VTEC engine”が真となるためには, ワーキングメモリ内のアサーション集合に”his-car has a VTEC engine”が含まれれば良い. ワーキングメモリ内を確認すると,”his-car has a VTEC engine”は存在するので, この仮説は真である. 以下, 図 14 に仮説のチェックの過程を示す.

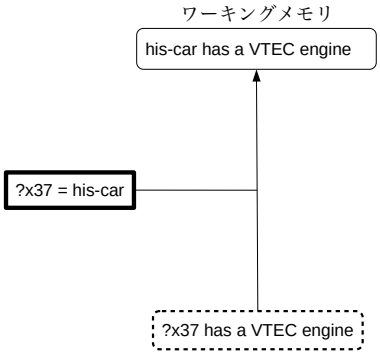

図 14　”his-car has a VTEC engine”のチェック

同様にして, 残りの仮説 [his-car is stylish, his-car has several color models, his-car has several seats, his-car is a wagon] についてもワーキングメモリ内を確認するとそれぞれに該当するアサーションが存在するために仮説が真となることが示される. 以下にこの様子を図 15 に示す.

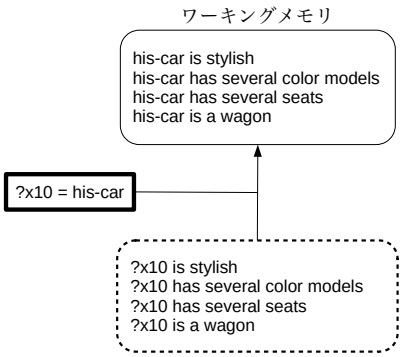

図 15　その他の仮説のチェック

以上によって,?x10=his-car としてユニフィケーションを行うことで, CarRule11 によって導出された全ての仮説が真となることが示されたので, 元の仮説,”?x is an Accord Wagon”について,?x=his-car が示された.

この後向き推論の成功例の一連の流れを以下の図 16 に示す.

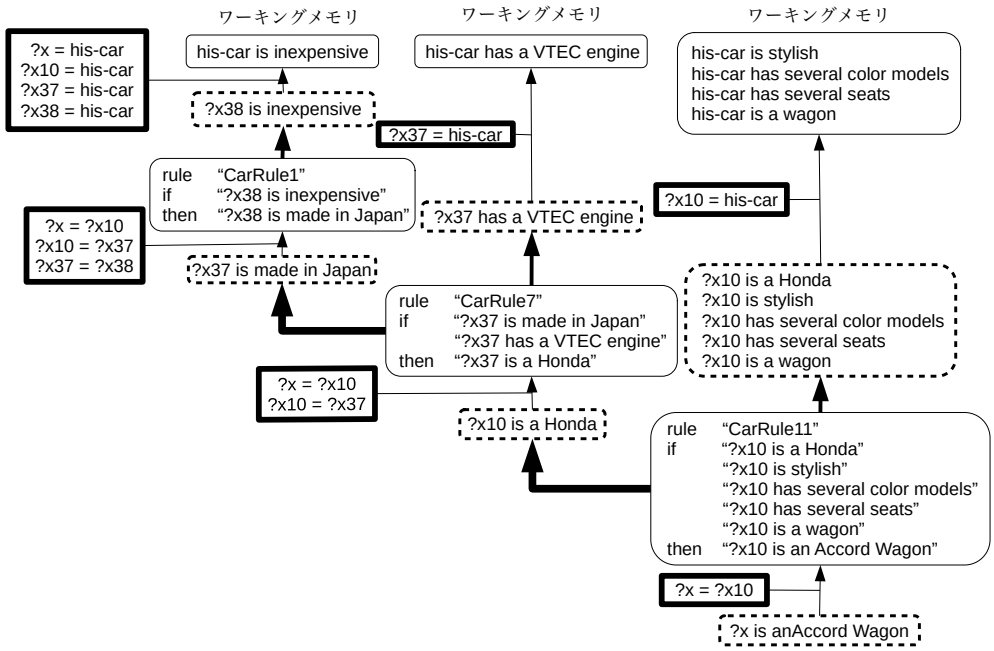

図 16　後向き推論の成功

2.4　考察

前向き推論と後向き推論の違いやそれぞれのメリット・デメリットを比較することによって, それぞれの特徴が鮮明になった. 具体的には, 前向き推論では網羅的な知識の獲得がなされ, 自動的に知識を導き出すといった点での実用がなされるのではないかと考えられる. 対照的に, 後向き推論では, その効率性の良さから欲しい知識を応答形式で得ることに特化しているように感じられた.
また, 前向き推論では, 単純な総探索のような形式を取ることによって知識を獲得するので, 応答形式を取る際には, あらかじめ下処理として前向き推論を行い, データ検索の際には駆動させない方向で運用すべきであると感じた. 対して, 後ろ向き推論では必ず応答形式による利用となることが考えられるため, あらかじめ質問者がどのような知識をどのような形式で質問可能であるかを知っておく必要がある.
また, 利用されない際には前向き推論による下処理を行い, 利用時には後向き推論を利用することにより, さらに効率の良いシステムになるのでは無いかと考えた. これは, 知識量やルールの規模が大きくなるほどに効果を増すのでは無いかと考えられる.
知識システムは, 知識の量や範囲によって運用の際に制限が大きいのでは無いかと考えられる上に, データ駆動型のシステムとは異なり, 全く新しい知識の発見を行うことができない（入力された知識に依存した範囲でのみの限定的なシステム）であるため, 実用的には, こうしたアクセシビリティの面でのブラッシュアップを考える必要があると感じた.

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.

3.1　手法

ルールの編集を実装するために, 内部処理として二つの段階に処理を分割する. まず, ルールベース内のルール集合への編集作業を行う. その次に, 変更されたルールベースの内容を, データファイルに上書きする処理を行う. 以上の手順に沿って, 実装を行う.

3.2 実装

ルールの編集を実装するために,RuleBase クラス内部に 4 つのメソッドを新たに追加した.

まず, ルールベース内のルールを編集する処理を行うメソッドを作成する. 今回, ルールの編集には, 追加, 削除, 変更の 3 つが存在するので, それぞれに対応したメソッドを作成する必要がある. 以下に, 今回作成した 3 つのメソッドとその説明を示す.

insertRule()　引数で指定されたルールをルールベースに追加する.
deleteRule()　引数で指定されたルールをルールベースから削除する.
updateRule()　引数で指定されたルールを変更する.

以上のメソッドによって, ルールベース内部のルール集合に対しての操作がなされる.

次に, 変更後のルールベースの内容をデータファイルに反映するメソッド writeFile() を定義する. このメソッドは, 上記の 3 つのメソッドから共通で利用されるものとする.

以上によって, ルールの編集を行う内部実装を行った.

3.3 実行例

私（増田）は, ルールの追加・削除・変更を行う実装を担当したため,CUI 上での実行結果を示す. まずは, 元のデータファイルを以下に示す.

ソースコード 3　初期の EditRuleTest.data

```
1 rule  "CarRule10"
2 if  "?x is a Toyota"
3     "?x has several seats"
4     "?x is a hybrid car"
5 then "?x is a Prius"
```

次に, ルール CarRule2 を追加する処理を行った際のデータファイルを示す.

ソースコード 4　ルール追加後の EditRuleTest.data

```
1 rule "CarRule10"
```

```
2 if "?x is a Toyota"
3         "?x has several seats"
4         "?x is a hybrid car"
5 then "?x is a Prius"
6
7 rule "CarRule2"
8 if "?x is small"
9 then "?x is made in Japan"
```

続いて, 先ほど追加したルール CarRule2 を削除したデータファイルを示す.

ソースコード 5　ルール削除後の EditRuleTest.data

```
1 rule "CarRule10"
2 if "?x is a Toyota"
3     "?x has several seats"
4     "?x is a hybrid car"
5 then "?x is a Prius"
```

最後に, ルール carRule10 を変更するプログラムを実行した結果を示す. ただし, 二番目の前件を削除する変更を行うとする.

ソースコード 6　ルール変更後の EditRuleTest.data

```
1 rule "CarRule10"
2 if "?x is a Toyota"
3         "?x has several seats"
4 then "?x is a Prius"
```

3.4　考察

ルールの編集を実装するに当たって 2 通りの実装方法を考案した. 一つ目の手法は, 上述した通り, ルールベース内部のルール集合への編集を行い, 操作後のルールベースの内容をデータファイルに上書きする手法である. 二つ目の手法は, データファイルを直接編集することによって, ルールを更新し, 再び読み込んでルールベースを構築する手法である. そこで, データファイルを直接変更することは複雑かつ非効率であるのに対して, ルールベース内部のルール集合を直接編集する方が容易であると考えたため, 前者を選ぶこととした.

また, 処理を二段階に切り分けることによって, 同様の処理を writeFile() メソッドに統合

し, 機能の差分をそれぞれ 3 つのメソッドに任せることで柔軟な構成を作ることができたと考えている.

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] LibreOffice Draw を用いた図版作成 –基礎プログラミング演習 I 資料 (京都産業大学) (2019 年 11 月 12 日アクセス)
http://www.cse.kyoto-su.ac.jp/~oomoto/lecture/program/LibreOffice/draw/index-j.html

[3] LaTeX の表を生成できるサイト Tables Generator –muscle_keisuke の日記 (2019 年 11 月 12 日アクセス)
http://muscle-keisuke.hatenablog.com/entry/2016/07/02/170205

[4] LaTeX Tables Generator –Tables Generator (2019 年 11 月 12 日アクセス)
http://www.tablesgenerator.com/latex_tables