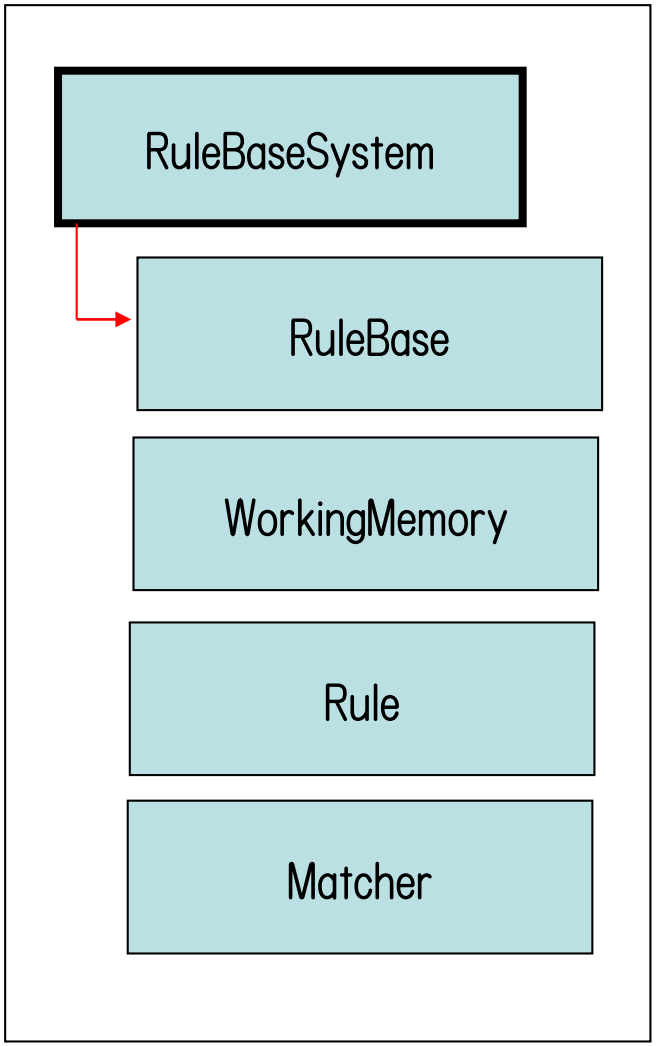
ルールベースシステム：前向き推論

クラス構成




```
 8:public class RuleBaseSystem {
 9: static RuleBase rb;
10: public static void main(String args[]){
11:   rb = new RuleBase();
12:   rb.forwardChain();
13: }
14:}

21:class WorkingMemory {
22: Vector assertions;
23:
24: WorkingMemory(){
25:   assertions = new Vector();
26: }
```

1) Apple is red
2) Sky is blue

ルールベースシステム：前向き推論

1) ?x is ?z
2) ?x is red

```
35: public Vector matchingAssertions(Vector theAntecedents){
36:  Vector bindings = new Vector();
37:  return matchable(theAntecedents, 0, bindings);
38: }
39:
40: private Vector matchable(Vector theAntecedents,
                            int n, Vector bindings){
41:  if(n == theAntecedents.size()){
42:   return bindings;
43:  } else if (n == 0){
44:   boolean success = false;
45:   for(int i = 0 ; i < assertions.size() ; i++){
46:    Hashtable binding = new Hashtable();
47:    if((new Matcher()).matching(
48:      (String)theAntecedents.elementAt(n),
49:      (String)assertions.elementAt(i),
50:       binding)){
51:      bindings.addElement(binding);
52:      success = true;
53:    }
54:   }
55:   if(success){
56:    return matchable(theAntecedents, n+1, bindings);
57:   } else {
58:    return null;
59:   }
60:  } else {
61:   boolean success = false;
62:   Vector newBindings = new Vector();
63:   for(int i = 0 ; i < bindings.size() ; i++){
64:    for(int j = 0 ; j < assertions.size() ; j++){
65:     if((new Matcher()).matching(
66:           (String)theAntecedents.elementAt(n),
67:      (String)assertions.elementAt(j),
68:      (Hashtable)bindings.elementAt(i))){
69:      newBindings.addElement(bindings.elementAt(i));
70:      success = true;
71:     }
72:    }
73:   }
74:   if(success){
75:    return matchable(theAntecedents,n+1,newBindings);
76:   } else {
77:    return null;
78:   }
79:  }
80: }
```

複数のマッチング→複数の変数束縛

n=1

2

2

2

1つ目のパターン
「?x is ?z」

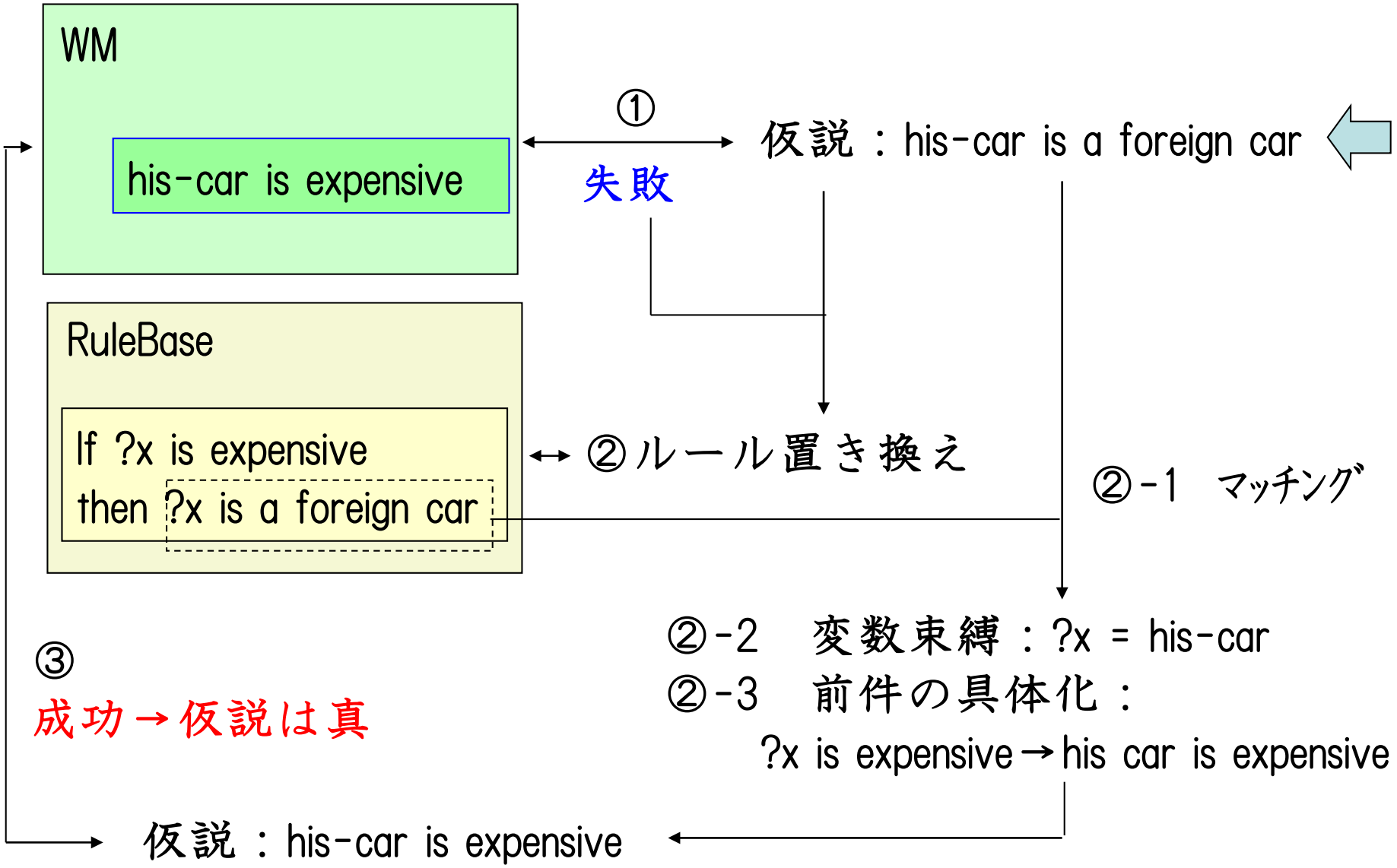
1) ?x = Apple, ?z = red
2) ?x = Sky, ?z = blue

2つ目のパターン
「?x is red」

1) ?x = Apple, ?z = red
~~2) ?x = Sky, ?z = blue~~

n+1=2

ルールベースシステム：後向き推論



ルールベースシステム：後向き推論

```
69: private boolean matchingPatterns(Vector
                        thePatterns, Hashtable theBinding){
70:  String firstPattern;
71:  if(thePatterns.size() == 1){
72:   firstPattern = (String)thePatterns.elementAt(0);
73:   if(matchingPatternOne(firstPattern,theBinding,0) != -1){
74:    return true;
75:   } else {
76:    return false;
77:   }
78:  } else {
79:   firstPattern = (String)thePatterns.elementAt(0);
80:   thePatterns.removeElementAt(0);
81:
82:   int cPoint = 0;
83:   while(cPoint < wm.size() + rules.size()){
            ......
122:   return false;
123:  }
124: }
```

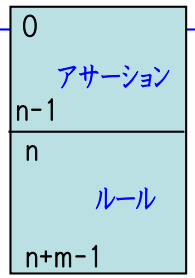
複数の仮説

仮説とマッチする変数束縛情報

仮説が１つの場合

整合を試行している対象を指示

仮説が２つ以上の場合




```
83:  while(cPoint < wm.size() + rules.size()){
84:   // 元のバインディングを取っておく
85:   Hashtable orgBinding = new Hashtable();
86:   for(Enumeration e = theBinding.keys() ;
            ......
91:   }
92:   int tmpPoint =
93:    matchingPatternOne(firstPattern,theBinding,cPoint);
94:   if(tmpPoint != -1){
95:    System.out.println("Success:"+firstPattern);
96:    if(matchingPatterns(thePatterns,theBinding)){
97:     //成功
98:     return true;
99:    } else {
100:    //失敗
101:    //choiceポイントを進める
102:    cPoint = tmpPoint;
103:    // 失敗したのでバインディングを戻す
104:    theBinding.clear();
            ......
110:   }
111:  } else {
112:   // 失敗したのでバインディングを戻す
113:   theBinding.clear();
            ......
119:   return false;
120:  }
121: }
```

ユニフィケーションが成功した対象の番号

ユニフィケーション成功

２つめ以降の全ての仮説のユニフィケーションが成功

２つめ以降のいずれかの仮説のユニフィケーションが失敗

元の仮説のマッチングに戻る

対象の残り（続き）を用いてマッチングを継続

ユニフィケーション失敗

仮説が２つ以上の場合に1つ目の仮説の束縛が失敗

ルールベースシステム：後向き推論

1つの仮説を満たす変数束縛を返す

```
126: private int matchingPatternOne(String thePattern,
                                    Hashtable theBinding,int cPoint){
127:  if(cPoint < wm.size() ){
128:   // WME(Working Memory Elements) と Unify してみる.
129:   for(int i = cPoint ; i < wm.size() ; i++){
130:    if((new Unifier()).unify(thePattern,
131:                             (String)wm.elementAt(i),
132:                             theBinding)){
133:     System.out.println("Success WM");
134:     System.out.println((String)wm.elementAt(i)+" <=>
                                                  "+thePattern);
135:     return i+1;
136:    }
137:   }
138:  }
139:  if(cPoint < wm.size() + rules.size() ){
140:   // Ruleの後件と Unify してみる.
141:   for(int i = cPoint ; i < rules.size() ; i++){
142:    Rule aRule = rename((Rule)rules.elementAt(i));
143:    // 元のバインディングを取っておく
144:    Hashtable orgBinding = new Hashtable();
145:    for(Enumeration e = theBinding.keys() ; e.hasMoreElements();){
146:     String key = (String)e.nextElement();
147:     String value = (String)theBinding.get(key);
148:     orgBinding.put(key,value);
149:    }
```

Ruleの後件

```
150:    if((new Unifier()).unify(thePattern,
151:                             (String)aRule.getConsequent(),
152:                             theBinding)){
153:     System.out.println("Success RULE");
154:     System.out.println("Rule:"+aRule+" <=> "+thePattern);
155:     // さらにbackwardChaining
156:     Vector newPatterns = (Vector)aRule.getAntecedents();
157:     if(matchingPatterns(newPatterns,theBinding)){
158:      return wm.size()+i+1;
159:     } else {
160:      // 失敗したら元に戻す.
161:      theBinding.clear();
162:      for(Enumeration e=orgBinding.keys();e.hasMoreElements();){
163:       String key = (String)e.nextElement();
164:       String value = (String)orgBinding.get(key);
165:       theBinding.put(key,value);
166:      }
167:     }
168:    }
169:   }
170:  }
171:  return -1;
172: }
```

Ruleの前件を仮説として再帰的に変数束縛を求める

ルールベースシステム：後向き推論

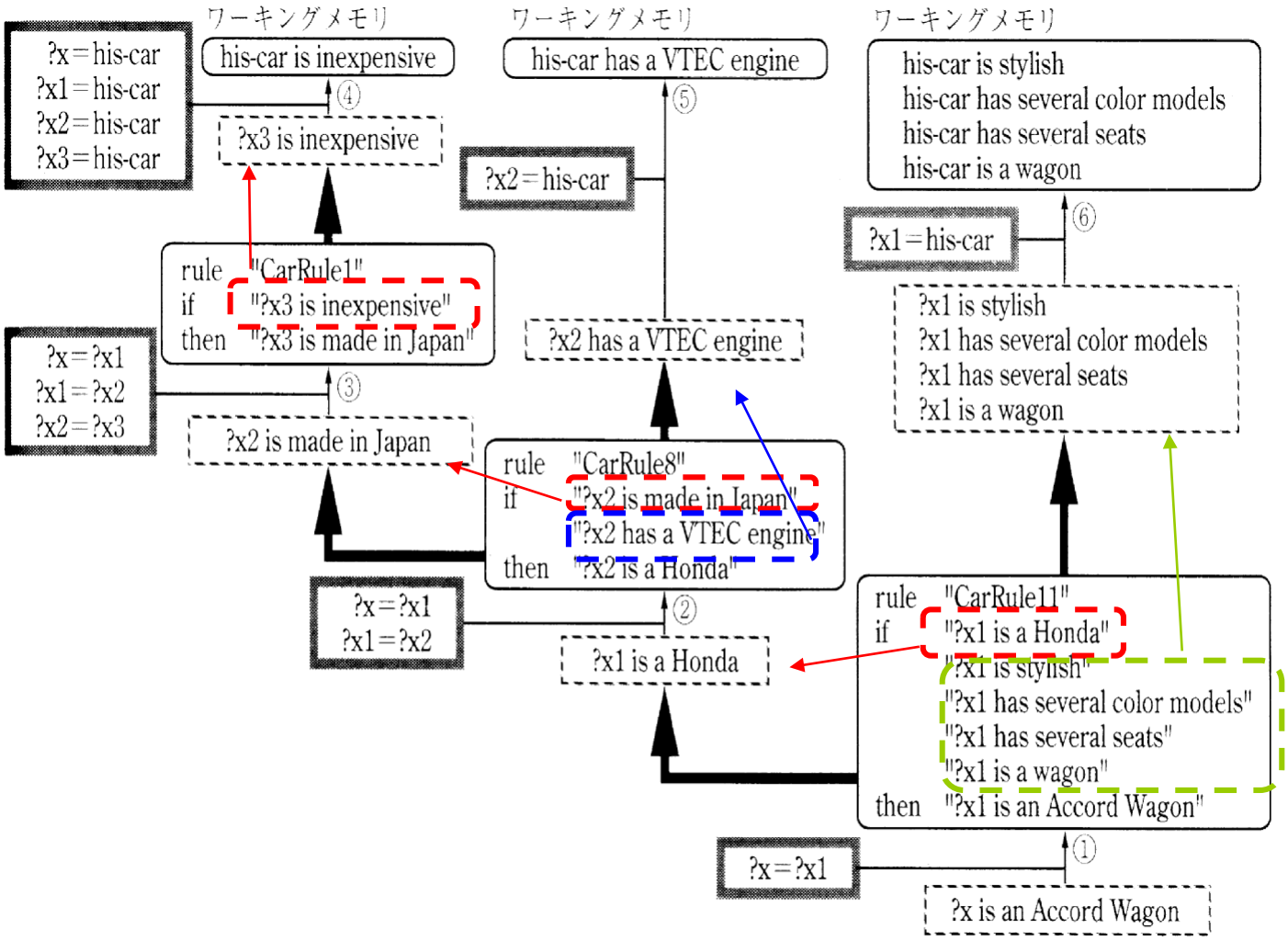


図 *3.12*　「?x is an Accord Wagon」に関する推論（①～⑥：仮説の検証の順番）