PARIS-SACLAY UNIVERSITY

Master 2 Data & Knowledge

Année 2017-2018

Report of Semantic Web (Lab 1)

by

Mengzi ZHAO

Horn rules reasoning

Professor : *MA Yue, Paris-Sud University*

I - Introduction

This practical exercise is to make us familiar with the reasoning mechanism of Horn Rule and we need to code the algorithms in Java.

My report includes tests of 4 algorithms:

1 – Forward Chaining algorithm
2 – Forward Chaining optimized algorithm
3 – Backward Chaining algorithm
4 – Backward Chaining with question

and some explications to make you understand better my ideas to code.

II – Test

To do the test for all algorithms, I wrote some codes in the document ReasoningHorn.java.

ReasoningForwardChaining.java

This algorithm permits to deduce new facts according to rules in the base from a base of initial facts. If conditions of the rule which is in the rule base all exist in the fact base, it means all of conditions of this rule are correct, so we can define this rule is correct, conclusions of this rule are correct too, so we can add these conclusions to the fact base. We apply this algorithm for all of the rules.

The function match is to verify if the condition is in the fact base. ´To test the **match(Variable condition, Formalism fb)** function :

```
Variable qq = new Variable("transoceanic_race");
System.out.println("Test match : " + reasoner.match(qq, fb));
```

I obtained the result :

```
Test match : false
```

The function eval is to verify if this rule is correct or not according to the fact base. To test the **eval(HornRule rule, FactBase fb)** function :

```
HornRule rule = kb.getRules().get(0);
System.out.println("Test the function eval : " + reasoner.eval(rule, fb));
```

I obtained the result :

```
Test the function eval : true
```

This function forwardchaining is to affect the algorithm of forward chaining. To test the **forwardChaining(Formalism ruleBase, Formalism factBase)** function :

```
System.out.println("Test forward chaining : " + reasoner.forwardChaining(kb,fb));
```

I obtained the result :

Test forward chaining : facts=[cruise_offshore, sailboat, longer_than_13, sailboat_cruise, transoceanic_race, portable, longer_than_10, sailing_dinghy, not_keel, not_portable, habitable, boat, sport, racing_can, sail, longer_than_8]

ReasoningForwardChainingOptimised.java

The optimized version for forward chaining don't need to repeat the operation of making correspondences between facts and conditions. We can delete the conditions of the rules which are in the fact base, if conditions of the rule are empty, it means this rule is correct and we can add conclusions of this rule to the fact base. We apply this algorithm for all the rules.

To test the **forwardChainingOptimised(Formalism ruleBase, Formalism factBase)** function :

```
ReasoningForwardChainingOptimised reasoner2 = new ReasoningForwardChainingOptimised();
System.out.println("Test the algorithm of forward chaining optimised");
System.out.println(reasoner2.forwardChainingOptimise(rb2, f2));
```

I obtained the result :

```
Test the algorithm of forward chaining optimised
facts=[cruise_offshore, sailboat, longer_than_13, sailboat_cruise, transoceanic_race, portable, longer_than_10, sailing_dinghy,
not_keel, not_portable, habitable, boat, sport, racing_can, sail, longer_than_8]
```

ReasoningBackwardChaining.java

This algorithm is to replace a goal by sub-goals by matching a current goal with the conclusion atoms of rules. To test the **backwardChaining(Formalism ruleBase, Formalism factBase)** function :

```
Formalism f3 = fb;
Formalism rb3 = kb;
Variable query = new Variable();
Variable query2 = new Variable();
Variable query3 = new Variable();
Variable query4 = new Variable();
Variable query5 = new Variable();

query.setNomVariable("gaff_rig");
query2.setNomVariable("transoceanic_race");
query3.setNomVariable("sailboat");
query4.setNomVariable("longer_than_13");
query5.setNomVariable("sailboat_cruise");

Formalism q = (Formalism) query;
Formalism q2 = (Formalism) query2;
Formalism q3 = (Formalism) query3;
Formalism q4 = (Formalism) query4;
Formalism q5 = (Formalism) query5;

ReasoningBackwardChaining reasoner3 = new ReasoningBackwardChaining();
System.out.println(reasoner3.backwardChaining(rb3, f3, q));
System.out.println(reasoner3.backwardChaining(rb3, f3, q2));
System.out.println(reasoner3.backwardChaining(rb3, f3, q3));
System.out.println(reasoner3.backwardChaining(rb3, f3, q4));
System.out.println(reasoner3.backwardChaining(rb3, f3, q5));
```

I obtained the result :

```
false
true
true
true
true
```

ReasoningBackwardChainingwithQuestions.java

This version permit to ask the user to obtain the response of the query which doesn't exist in the conclusions of rules. I use the "Scanner" to realize the input of the response of user.

To test the **backwardChainingwithQuestions(Formalism ruleBase, Formalism factBase)** function :

```
Formalism f4 = fb;
Formalism rb4 = kb;
ReasoningBackwardChainingwithQuestions reasoner4 = new ReasoningBackwardChainingwithQuestions();
Variable query6 = new Variable();
query6.setNomVariable("gaff_rig");
Formalism q6 = (Formalism) query6;
System.out.println("Test backward chaining with questions");
System.out.println(reasoner4.backwardChainingwithQuestions(rb4, f4, q6));
```

I obtained the result :

```
Test backward chaining with questions
This query is demandable, please enter your answer (true/false) :

true
This query is : true
True
```

III – Conclusion

After doing this exercise, I can know better the knowledge of Horn rules reasoning, this is also a good chance to remind the syntax of java.